ROM BIOS LISTING

AT&T Personal Computer 6300

ROM BIOS Listing

This manual contains the ROM BIOS listing for the AT&T Personal Computer 6300. The Basic Input/Output System (BIOS) is located in the Read-Only Memory (ROM) on the PC 6300 motherboard. The ROM BIOS serves as an interface between the computer system and the input/output devices connected to the system ports. The information supplied by the ROM BIOS controls these devices. During normal operation, the ROM BIOS operates much like a driver that is resident in the PC 6300 memory space.

```
                        ;.xlist


                        page 60,132
                        ;===============================================================
                        ;       Filename:flags.src
                        ;
                        ;       This file contains the temporary conditional assembly flag
                        ;       definitions and includes of the other assembly modules.
                        ;
                        ;===============================================================

                        ;---------------------------------------------------------------
                        ;       Macro Definitions
                        ;---------------------------------------------------------------

                        ; This macro is to prevent 'overORGing' one area of code with another.
                        ; It is used the same as ORG.

                        XORG macro    loc
                             $$$x = $ - start           ;; Strange var name to avoid possible
                                                        ;; conflict.

                             if ($$$x gt loc)

                               if1

                                    %out ERROR:  ORG conflict at loc

                               endif

                                    ORG error at loc
                        else

                             org      loc

                        endif
                             endm

                        ;
                        ; MASM does not let you code a 'jump intersegment direct' instruction, so
                        ; this macro simulates that instruction.

                        jmpf macro    arg1,arg2         ;; USAGE:         jmpf     seg,off
                             db       0EAh
                             dw       arg2
                             dw       arg1
                             endm

                        ;---------------------------------------------------------------------------
                        ;       Code Declaration
                        ;---------------------------------------------------------------------------

0000                    code segment  public 'ROM'      ; link code segments first
                             assume   cs:code, ds:nothing, es:nothing, ss:nothing
```

```
0000                            startlabel      near

C000                                 ORG        0C000h

C000                            flags_data1     proc

C000  00                        chk_lo          db      0       ; space for checksum of F000:C000 to F000:DFFF
C001  00                                        db      0       ; Filler.
C002  E23C R                    rom_mt          dw      mastab  ; offset of mastab in ROM.

                                flags_data1     endp

C004                            far_calls       proc    far     ; far call table: the user does a far call to
                                                                ; F000:C0XX, a near call is done to the proper
                                                                ; routine, and a far return back to the user.

C004  E8 E009 R                      call       DString         ; F000:C004     (3 bytes per near call)
C007  CB                             ret                        ;               (1 byte per far return)
C008  E8 E57C R                      call       DCrLf           ; F000:C008
C00B  CB                             ret
C00C  E8 E589 R                      call       DColon          ; F000:C00C
C00F  CB                             ret
C010  E8 E595 R                      call       DHexLong        ; F000:C010
C013  CB                             ret
C014  E8 E5A6 R                      call       DHexWord        ; F000:C014
C017  CB                             ret
C018  E8 E5AD R                      call       DHexByte        ; F000:C018
C01B  CB                             ret
C01C  E8 E5C1 R                      call       DHexNib         ; F000:C01C
C01F  CB                             ret
C020  E8 E5D6 R                      call       DNum            ; F000:C020
C023  CB                             ret
C024  E8 E5DE R                      call       DNumW           ; F000:C024
C027  CB                             ret
C028  E8 E566 R                      call       rom_checksum    ; F000:C028
C02B  CB                             ret
C02C  E8 E165 R                      call       rtc_chk         ; F000:C02C
C02F  CB                             ret
C030  E8 E1D4 R                      call       memtst          ; F000:C030
C033  CB                             ret
C034  E8 D09F R                      call       h_init          ; F000:C034
C037  CB                             ret
C038  E8 D5DE R                      call       h_fmt           ; F000:C038
C03B  CB                             ret

C03C  D9CA R                         dw         offset banner_m         ; pointer to banner
C03E  0000                           dw         0                       ; For aligning the copyright message.
C040  43 4F 50 59 52 49 47           db         'COPYRIGHT (C)'
      48 54 20 28 43 29 20
      20 20
C050  4F 4C 49 56 45 54 54           db         'OLIVETTI 1984'
      49 20 31 39 38 34 20
      20 20
```

```
                        far_calls     endp

C060                    code ends

                        ;--------------------------------------------------------------------
                        ;    Includes of Assembly Modules
                        ;--------------------------------------------------------------------

                        ;include flags.asm                              (this module)
                      C include sysdata.asm
                      C
                      C ;====================================================================
                      C ;    Filename:sysdata.src
                      C ;
                      C ;    This is the port equate and system data definition module.
                      C ;    (See flags.src for conditional assembly flags...)
                      C ;
                      C ;====================================================================
                      C
                      C ;--------------------------------------------------------------------
                      C ;    Global Constants
                      C ;--------------------------------------------------------------------
                      C
= 0000                C abs0_seg      equ     00000h
= 0030                C stack_seg     equ     00030h
= 0040                C data_seg      equ     00040h
= B000                C para_mono     equ     0B000h
= B800                C para_graph    equ     0B800h
= F000                C code_seg      equ     0F000h
                      C
= 0007                C BEL           equ     007h
= 0008                C BS            equ     008h
= 000D                C CR            equ     00Dh
= 000A                C LF            equ     00Ah
                      C
= 0000                C NUL           equ     0
                      C
                      C ;--------------------------------------------------------------------
                      C ;    PC1050 Addresses
                      C ;--------------------------------------------------------------------
                      C
                      C ;--------------------------------------------------------------------
                      C ;    i8237A p_dma Controller Port Addresses
                      C ;--------------------------------------------------------------------
                      C
= 0000                C dma_addr_0    equ     00h     ; 16-bit address register - channel 0 - refresh
= 0001                C dma_count_0   equ     01h     ; 16-bit count register
= 0002                C dma_addr_1    equ     02h     ; 16-bit address register - channel 1 - not used
= 0003                C dma_count_1   equ     03h     ; 16-bit count register
= 0004                C dma_addr_2    equ     04h     ; 16-bit address register - channel 2 - FDU
= 0005                C dma_count_2   equ     05h     ; 16-bit count register
= 0006                C dma_addr_3    equ     06h     ; 16-bit address register - channel 3 - display
= 0007                C dma_count_3   equ     07h     ; 16-bit count register
                      C
= 0008                C dma_status    equ     08h     ;  8-bit read status register
```

```
= 0008                  C  dma_command    equ     08h     ;  8-bit write command register
= 0009                  C  dma_request    equ     09h     ;  4-bit write request register
= 000A                  C  dma_mask_bit   equ     0Ah     ;  4-bit (write) set/clear one mask register bit
= 000B                  C  dma_mode       equ     0Bh     ;  6-bit write mode register
= 000C                  C  dma_ff_clr     equ     0Ch     ;        (write) clear byte pointer flip/flop
= 000D                  C  dma_temp       equ     0Dh     ;  8-bit read temporary register
= 000D                  C  dma_master_clrequ      0Dh     ;        (write) master clear command
= 000E                  C  dma_mask_clr   equ     0Eh     ;  4-bit (write) clear all mask register bits
= 000F                  C  dma_mask_writeequ      0Fh     ;  4-bit write all mask register bits at once
                        C
= 0080                  C  dma_segm_0     equ     080h    ; RAM refresh - 4x4-bit high nibble segment port
= 0082                  C  dma_segm_1     equ     082h    ; not used
= 0081                  C  dma_segm_2     equ     081h    ; FDU
= 0083                  C  dma_segm_3     equ     083h    ; display                TEMP
                        C
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;    i8237A p_dma controller constants
                        C  ;---------------------------------------------------------------------
                        C
                        C  ; dma_command port:                      ; bit #0: memory-to-memory/~I/O enable
                        C                                           ; bit #2: controller disable
                        C                                           ; bit #3: compressed/~normal timing
                        C                                           ; bit #4: rotating/~fixed priority
                        C                                           ; bit #5: extended/~late write selection
                        C                                           ; bit #6: DREQ active low/~high
                        C                                           ; bit #7: DACK active high/~low
                        C
= 0004                  C  dma_cmd_disable        equ     004h    ; controller disable (bit #2) command
= 0000                  C  dma_cmd_enableequ      000h            ; memory-to-I/O.controller enable.normal
                        C                                         ; fixed priority. late write. DREQ/~DACK
                        C
= 0058                  C  dma_mode_0     equ     058h            ; channel 0. read. autoinitialize. inc-
                        C                                         ; rement. single mode for RAM refresh.
= 0041                  C  dma_mode_1     equ     041h            ; channel 1. verify. autoinit disabled.
                        C                                         ; increment. single mode for not used.
= 0056                  C  dma_mode_2     equ     056h            ; channel 2. write. autoinitialize. inc-
                        C                                         ; rement. single mode for FDU.
= 0043                  C  dma_mode_3     equ     043h            ; channel 3. verify. autoinit disabled.
                        C                                         ; increment. single mode for display.
                        C
                        C  ; dma_mask_bit port:                     ; bits #0-1: channel select
                        C                                           ; bit  #  2: set/~clr mask bit (off/~on)
                        C
= 0000                  C  dma_unmask_0   equ     000h            ; turn on channel 0 for RAM refresh.
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;    i8259A Programmable Interrupt Controller Port Addresses
                        C  ;---------------------------------------------------------------------
                        C
= 0020                  C  pic_0          equ     020h    ; 8259A 'control' port (A0 = 0)
= 0021                  C  pic_1          equ     021h    ; 8259A 'data'    port (A0 = 1)
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;    i8259A Programmable Interrupt Controller Commands
```

```
                        C  ;--------------------------------------------------------------------
                        C
= 0013                  C  pic_icw1      equ      013h     ; ICW1 for both master & slave pic's
                        C                                  ; bit #0 = 1: ICW4 to follow (w/vector base)
                        C                                  ; bit #1 = 1: single mode (no slaves or icw3)
                        C                                  ; bit #2 = 0: call address interval of 8 bytes
                        C                                  ; (don't care if 8086 mode -- always vectors 4
                        C                                  ; byte interval)
                        C                                  ; bit #3 = 0: edge triggered
= 0008                  C  pic_icw2      equ      008h     ; interrupt vector base address (INTs 08h - 0Fh)
= 0008                  C  pic_icw3        equ     008h    ; if cascade mode , and IR3 is a
                        C                                  ; slave, 8259A is reprogrammed including icw3
= 000D                  C  pic_icw4      equ      00Dh     ; bit #0 = 1: 8086 mode
                        C                                  ; bit #1 = 0: normal end_of_int
                        C                                  ; bit #2 = 1: specify master for buffered mode
                        C                                  ; (  specifies slave for buffered mode )
                        C                                  ; bit #3 = 1: buffered mode
                        C                                  ; bit #4 = 0: not special fully nested
= 00FF                  C  pic_off_msk   equ      0FFh     ; pic interrupt mask bits (all interrupts off)
                        C
= 0020                  C  pic_neoi      equ      020h     ; non-specific end-of-interrupt
                        C
= 0060                  C  pic_seoi_0    equ      060h     ; specific end-of-interrupt for IR0: i8253 p_timer
= 0061                  C  pic_seoi_1    equ      061h     ; specific end-of-interrupt for IR0: i8041A kb
= 0066                  C  pic_seoi_6    equ      066h     ; specific end-of-interrupt for IR6: fdu
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    i8253 p_timer Port Addresses
                        C  ;--------------------------------------------------------------------
                        C
= 0040                  C  p_8253_0      equ      040h     ; 8253 p_timer 0 - rtc interrupt - IR0 = INT 08h
= 0041                  C  p_8253_1      equ      041h     ; 8253 p_timer 1 - memory refresh p_dma
= 0042                  C  p_8253_2      equ      042h     ; 8253 p_timer 2 - tone generator for speaker
= 0043                  C  p_8253_ctrl   equ      043h     ; 8253 p_timer control port
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    i8253 p_timer Control Bytes
                        C
                        C  ; bit  #0     -> Binary Code Decimal (BCD) Enable
                        C  ; bits #1-3   -> Mode (0-5)     000      Mode 0: Interrupt on Terminal Count
                        C  ;                               001      Mode 1: Programmable One-Shot
                        C  ;                               x10      Mode 2: Rate Generator
                        C  ;                               x11      Mode 3: Square Wave Rate Generator
                        C  ;                               100      Mode 4: Software Triggered Strobe
                        C  ;                               101      Mode 5:  Hardware Triggered Strobe
                        C  ; bits #4-5   -> Read/Load Instruction (0-3)
                        C  ; bits #6-7   -> Select Counter (0-2)
                        C
                        C  ;--------------------------------------------------------------------
= 0036                  C  t0cmd         equ      036h     ; 00 11 011 0 -> p_8253_0, lsb 1st, mode 3, no BCD
= 0074                  C  t1cmd         equ      074h     ; 01 11 010 0 -> p_8253_1, lsb 1st, mode 2, no BCD
= 00B6                  C  t2cmd         equ      0B6h     ; 10 11 011 0 -> p_8253_2, lsb 1st, mode 3, no BCD
= 0054                  C  t3cmd         equ      054h     ; 01 11 010 0 -> p_8253_1, lsb 1sonly, mode 2, no BCD
                        C
                        C  ;--------------------------------------------------------------------
```

```
                        C  ;    i8253 p_timer Counts
                        C
                        C  ; 8253 input is 1.2288 MHz (3.6864/3) or a period of 813.8 nsec = 0.814 usec
                        C  ; Note:     PC input is 1.19318 MHz or a period of 838.1 nsec = 0.838 usec
                        C
                        C  ;---------------------------------------------------------------------
= 0000                  C  t0count equ   0        ; = 65,536  -> (1,228,800 Hz)/(65,536) = 18.75 ints/sec
                        C                         ;          -> (1,193,180 Hz)/(65,536) = 18.21 ints/sec
                        C
                        C  ;t1count equ  9        ; OLD refresh cycle =  9*(813.8 nsec) = 7.32 usec
                        C
= 0013                  C  t1count       equ      19       ; REAL refresh cycle = 19*(813.8 nsec) = 15.5 usec
                        C                         ; < 15.625 usec minimum required. (  is 18 - safety??)
                        C
= 0266                  C  t2count       equ      614      ; (1.2288 MHz)/(2*614) = 1.00 kHz tone
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;    Z8530 Serial Communication Controller
                        C  ;
                        C  ;    (scc_data_x port addresses are indexed from the scc_ctl_x
                        C  ;    port addresses.  See com.src.)
                        C  ;---------------------------------------------------------------------
                        C
= 0050                  C  scc_ctl_a     equ      050h     ; write to SCC pointer register (0-Fh),
= 0052                  C  scc_ctl_b     equ      052h     ; then read or write from selected register.
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;    8041 Keyboard Controller
                        C  ;---------------------------------------------------------------------
                        C
= 0060                  C  p_kscan               equ      060h
= 0061                  C  p_kctrl               equ      061h     ; bit  #7:       reset interrupt pending
                        C                                         ; bit  #6:       kb clock reset
                        C                                         ; bit  #5:       I/O channel (NMI) ~enable
                        C                                         ; bit  #4:       RAM parity (NMI) ~enable
                        C                                         ; bits #3 & #2:  not used
                        C                                         ; bit  #1:       speaker data
                        C                                         ; bit  #0:       speaker gate to p_8253_2
                        C
= 0020                  C  I_O_parity_disable equ 020h    ; disable bit for I/O parity (memory board)
= 0010                  C  RAM_parity_disable equ 010h    ; disable bit for RAM parity (motherboard)
= 0030                  C  parity_disableequ     030h     ; disable bits for parity NMIs
                        C
= 0064                  C  kb_status     equ      064h     ; bit  #1:       input buffer (ok to write byte)
                        C                                         ; bit  #0:       output buffer (byte to be read)
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;    General Control Ports
                        C  ;---------------------------------------------------------------------
                        C
= 0062                  C  ControlC      equ      062h     ; bit  #7:       Ram parity check.
                        C                                         ; bit  #6:       I/O channel parity check.
                        C                                         ; bit  #1:       8087 installed
= 0065                  C  CommControl   equ      065h
                        C
```

```
                        C  ; Switch at 7T on Motherboard:
                        C
= 0066                  C  sys_conf_a    equ      066h     ; bit  #7:        Bank 1 Populated
                        C                                  ; bit  #6:        not used
                        C                                  ; bit  #5:        SCC 8530 chip installed
                        C                                  ; bit  #4:        8087 installed
                        C                                  ; bit  #3:        256k x 1 RAM used
                        C                                  ; bits #2 - #0:   RAM configuration
                        C
= 0010                  C  switch_8087   equ      010h     ; 8087 NPU presencs bit in dip switch port
                        C
                        C  ; Switch at 7W on Motherboard:
                        C
= 0067                  C  sys_conf_b    equ      067h     ; bits #7 - #6:   (number of FDUs)-1
                        C                                  ; bits #5 - #4:   reserved for monitor type
                        C                                  ; bit  #3:  1 = non-Olivetti Video Controller
                        C                                  ; bit  #2:  0 = use indiginous HDU code.
                        C                                  ;           1 = do not use indiginous HDU code.
                        C                                  ; bit  #1:       " fast" FDU start up
                        C                                  ; bit  #0:       96 tpi FDU
                        C
= 0008                  C  NON_OLI_VID   equ      08       ; Non-Olivetti Video Controller Installed
                        C
                        C
= 0080                  C  nmi_enable    equ      80h
= 00A0                  C  nmi_enable_port        equ      0A0h
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;    58174A Clock Calendar
                        C  ;
                        C  ;    (See calendar.src)
                        C  ;---------------------------------------------------------------------
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;    FDU & HDU Disk Driver Error Codes
                        C  ;---------------------------------------------------------------------
                        C
= 0080                  C  time_out      equ      80h
= 0040                  C  seek_error    equ      40h
= 0020                  C  fdc_error     equ      20h
= 0010                  C  crc_error     equ      10h
= 0009                  C  dma_seg_error equ      09h
= 0008                  C  dma_error     equ      08h
= 0004                  C  sect_not_foundequ      04h
= 0003                  C  write_protect equ      03h
= 0002                  C  addr_mark_error        equ      02h
= 0001                  C  cmd_error     equ      01h
                        C
                        C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; FDU EQUATES  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                        C  ;             Controller constants
                        C
= 000C                  C  f_srt_48               equ      1100b    ; 48TPI Step Rate Time (6 ms @ 4 Mhz)
= 000E                  C  f_srt_96               equ      1110b    ; 96TPI Step Rate Time (4 ms @ 4 Mhz)
= 000F                  C  f_hut                  equ      1111b    ; Head Unload Time (480 ms @ 4 Mhz)
= 0001                  C  f_hlt                  equ      1        ; Head Load Time (4 ms @ 4 Mhz)
```

```
= 0000                  C  f_ndma                       equ     0       ; Not DMA bit (0 = dma on)
                        C
= 0025                  C  f_motor_wait         equ     37      ; no. of RTC ticks before turning
                        C                                       ; motor off. (f_motor_wait x 55ms)
                        C
=                       C  f_drive              equ     [bp+0]                  ; byte pointer.
=                       C  f_head               equ     [bp+1]                  ; byte pointer.
=                       C  f_numsecs    equ     [bp+2]                  ; byte pointer.
=                       C  f_command    equ     [bp+3]                  ; byte pointer.
=                       C  f_bufoff     equ     [bp+4]                  ; word pointer.
=                       C  f_secnum     equ     [bp+6]                  ; byte pointer.
=                       C  f_cyl        equ     [bp+7]                  ; byte pointer.
=                       C  f_real_drive equ     [bp+8]                  ; byte pointer.
                        C
                        C  ;    Floppy Disk port addresses
                        C
= 03F2                  C  f_motor_port         equ     03F2h   ; drive select port
= 03F4                  C  f_nec_status         equ     03F4h   ; disk controller status port
= 03F5                  C  f_nec_data           equ     03F5h   ; disk controller data port
                        C
                        C  ;    Floppy Disk commands
                        C
= 00E6                  C  f_read_cmd           equ     0E6h    ; read data
= 00C5                  C  f_write_cmd          equ     0C5h    ; write data
= 004D                  C  f_format_cmd         equ     04Dh    ; format
= 0007                  C  f_recal_cmd          equ     007h    ; recalibrate
= 0008                  C  f_snsint_cmd         equ     008h    ; sense interrupt
= 0004                  C  f_snsdrv_cmd         equ     004h    ; sense drive
= 0003                  C  f_specify_cmd        equ     003h    ; specify
= 000F                  C  f_seek_cmd           equ     00Fh    ; seek
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;    Game Card
                        C  ;---------------------------------------------------------------------
                        C
= 0201                  C  game_card    equ     201h
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;    Parallel Printer Interface
                        C  ;
                        C  ;    (prt_stat_x & prt_cmd_x port addresses are indexed from the
                        C  ;    prt_data_x port addresses.  See prt.src.)
                        C  ;---------------------------------------------------------------------
                        C
= 03BC                  C  prt_data_a   equ     03BCh
                        C  ; prt_stat_a equ     03BDh
                        C  ; prt_cmd_a  equ     03BEh
                        C
= 0378                  C  prt_data_b     equ     0378h   ; on mother board
                        C  ; prt_stat_b equ     0379h
                        C  ; prt_cmd_b  equ     037Ah
                        C
= 0278                  C  prt_data_c   equ     0278h
                        C  ; prt_stat_c equ     0279h
                        C  ; prt_cmd_c  equ     027Ah
```

```
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Color and Monochrome Video Controller
                        C  ;
                        C  ;    (xxxxx_data, xxxxx_mode, xxxxx_status, xxxxx_LPclear, and
                        C  ;    xxxxx_LPPreset port addresses are indexed from the xxxxx_Pointer
                        C  ;    port addresses for color & display.  See vid.src and graph.src.)
                        C  ;--------------------------------------------------------------------
                        C
                        C  ; Color Controller.
                        C
= 03D4                  C  color_pointer equ     03D4h          ; 6845 pointer to internal regs
                        C
                        C  ; Monochrome Controller.
                        C
= 03B4                  C  v_pointer     equ     03B4h          ; 6845 pointer to internal regs
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    INS8250 Asynchronous Communication Chip
                        C  ;
                        C  ;    (com_int_x, com_lctl_x, com_mctl_x, com_lstat_x, and
                        C  ;    com_mstat_x port addresses are indexed from the com_data_x
                        C  ;    port addresses.  See com.src.)
                        C  ;--------------------------------------------------------------------
                        C
= 03F8                  C  com_data_a    equ     03F8h    ; channel A 8250 data register/low byte baud
                        C  ; com_int_a   equ     03F9h    ; channel A 8250 high byte baud count register
= 03FA                  C  com_id_a      equ     03FAh    ; channel A 8250 check for presence register
                        C  ; com_lctl_a  equ     03FBh    ; channel A 8250 line control register
                        C  ; com_mctl_a  equ     03FCh    ; channel A 8250 modem control register
                        C  ; com_lstat_a equ     03FDh    ; channel A 8250 line status register
                        C  ; com_mstat_a equ     03FEh    ; channel A 8250 modem status register
                        C
= 02F8                  C  com_data_b    equ     02F8h    ; channel B 8250 data register/low byte baud
                        C  ; com_int_b   equ     02F9h    ; channel B 8250 high byte baud count register
= 02FA                  C  com_id_b      equ     02FAh    ; channel B 8250 check for presence register
                        C  ; com_lctl_b  equ     02FBh    ; channel B 8250 line control register
                        C  ; com_mctl_b  equ     02FCh    ; channel B 8250 modem control register
                        C  ; com_lstat_b equ     02FDh    ; channel B 8250 line status register
                        C  ; com_mstat_b equ     02FEh    ; channel B 8250 modem status register
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Keyboard Constants
                        C  ;--------------------------------------------------------------------
                        C
                        C  ;---- shift flag equates within kb_flag_1
                        C
= 0080                  C  insert_mode     equ   80h     ; insert state in action
= 0040                  C  caps_lock_mode  equ   40h     ; caps lock state toggled
= 0020                  C  num_lock_mode   equ   20h     ; num lock state toggled
= 0010                  C  scrl_lock_mode  equ   10h     ; scroll lock state toggled
= 0008                  C  pause_mode      equ   08h     ; pause toggled
= 0001                  C  dlx_kb          equ   01h     ; deluxe keyboard
                        C
                        C
```

```
                        C  ;---- shift flag equates within kb_flag
                        C
= 0080                  C  insert_shift    equ     80h     ; insert key depressed
= 0040                  C  caps_lock_shift equ     40h     ; caps lock key depressed
= 0020                  C  num_lock_shift  equ     20h     ; num lock key depressed
= 0010                  C  scrl_lock_shift equ     10h     ; scroll lock key depressed
= 0008                  C  alt_shift       equ     08h     ; alternate shift key depressed
= 0004                  C  cntrl_shift     equ     04h     ; control shift key depressed
= 0002                  C  left_shift      equ     02h     ; left shift key depressed
= 0001                  C  right_shift     equ     01h     ; right shift key depressed
                        C
                        C
                        C  ;---- Scan codes for special function keys
                        C
= 001D                  C  cntrl_key       equ     29      ; control key scan code
= 002A                  C  left_shift_key  equ     42      ; left shift scan code
= 0036                  C  right_shift_key equ     54      ; right shift scan code
= 0038                  C  alt_key         equ     56      ; alt shift key scan code
= 003A                  C  caps_lock_key   equ     58      ; shift lock scan code
= 0045                  C  num_lock_key    equ     69      ; number lock scan code
= 0046                  C  scrl_lock_key   equ     70      ; scroll lock key scan code
= 0052                  C  insert_key      equ     82      ; insert key scan code
= 0053                  C  delete_key      equ     83      ; delete key scan code
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Data Declarations
                        C  ;--------------------------------------------------------------------
                        C
                        C  ;--------------------------------------------------------------------------
                        C  ;    Interrupt Locations (dummy data segment to define constant offsets)
                        C  ;--------------------------------------------------------------------------
                        C
0000                    C  abs0 segment  public 'RAM'              ; at abs0_seg
                        C       assume   cs:nothing, ds:nothing, es:nothing, ss:nothing
                        C
                        C  ;--------------------------------------------------------------------------
                        C  ; CPU Interrupt Routines
                        C  ;--------------------------------------------------------------------------
                        C
0000  ????????          C  int00locn     dd      ?              ; divide by zero
0004  ????????          C  int01locn     dd      ?              ; single step trap
0008  ????????          C  int02locn     dd      ?              ; nmi parity trap
000C  ????????          C  int03locn     dd      ?              ; break interrupt
0010  ????????          C  int04locn     dd      ?              ; divide overflow
0014  ????????          C  int05locn     dd      ?              ; print screen
                        C
0018  ????????          C  int06locn     dd      ?
001C  ????????          C  int07locn     dd      ?
                        C
                        C  ;--------------------------------------------------------------------------
                        C  ; i8259A Hardware Interrupt Routines
                        C  ;--------------------------------------------------------------------------
                        C
0020  ????????          C  int08locn     dd      ?              ; i8253 rtc interrupt
0024  ????????          C  int09locn     dd      ?              ; i8041 kb interrupt
```

```
                        C
0028  ????????          C  int0Alocn     dd       ?
002C  ????????          C  int0Blocn     dd       ?
0030  ????????          C  int0Clocn     dd       ?
                        C
0034  ????????          C  int0Dlocn     dd       ?               ; hard disk interrupt
0038  ????????          C  int0Elocn     dd       ?               ; floppy disk interrupt
                        C
003C  ????????          C  int0Flocn     dd       ?
                        C
                        C  ;--------------------------------------------------------------------------
                        C  ; Software Interrupt Routines
                        C  ;--------------------------------------------------------------------------
                        C
0040  ????????          C  int10locn     dd       ?               ; display request
0044  ????????          C  int11locn     dd       ?               ; equipment request
0048  ????????          C  int12locn     dd       ?               ; memory size request
004C  ????????          C  int13locn     dd       ?               ; disk I/O request
0050  ????????          C  int14locn     dd       ?               ; serial communication request
0054  ????????          C  int15locn     dd       ?               ; cassette request
0058  ????????          C  int16locn     dd       ?               ; kb request
005C  ????????          C  int17locn     dd       ?               ; printer request
0060  ????????          C  int18locn     dd       ?               ; cassette BASIC pointer
0064  ????????          C  int19locn     dd       ?               ; boot-strap request
0068  ????????          C  int1Alocn     dd       ?               ; time of day request
006C  ????????          C  int1Blocn     dd       ?               ; kb break pointer
0070  ????????          C  int1Clocn     dd       ?               ; p_timer break pointer
0074  ????????          C  int1Dlocn     dd       ?               ; display parameter pointer
0078  ????????          C  int1Elocn     dd       ?               ; disk parameter pointer
007C  ????????          C  int1Flocn     dd       ?               ; graphics character extensions pointer
                        C
0080                    C  abs0 ends
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    RAM stack
                        C  ;--------------------------------------------------------------------
                        C
0000                    C  stack_ram     segment  public 'RAM'      ; at stack_seg
                        C
0000                    C  stack_ram     ends
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    System Data Area
                        C  ;--------------------------------------------------------------------
                        C
0000                    C  data segment  public 'RAM'              ; at data_seg
                        C       assume   cs:nothing, ds:nothing, es:nothing, ss:nothing
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;     Data Area
                        C  ;--------------------------------------------------------------------
                        C
                        C  ;     ROM Bios Data Area
                        C
0000  0004[             C  rs232_addr    dw       4 dup (?)  ; 0040:0000     addresses of rs232 adapters
```

```
          ????            C
                        ]   C
                            C
0008  0004[                 C  printer_addr  dw      4 dup (?)  : 0040:0008     addresses of printers
          ????              C
                        ]   C
                            C
0010  ????                  C  switch_bits   dw      ?          : 0040:0010     state of DIP switches
0012  00                    C  mfg_tst                db      0          : 0040:0012     initialization flag
0013  ????                  C  memory_size   dw      ?          : 0040:0013     memory size in kbytes
0015  0002[                 C  mfg_err_flag  db      2 dup (?)  : 0040:0015     error codes for manufacturing
          ??                C
                        ]   C
                            C
                            C
                            C  :      Keyboard Data Area
                            C
0017  ??                    C  kb_flag               db      ?          : 0040:0017     keyboard shift flag status byte
0018  ??                    C  kb_flag_1     db      ?          : 0040:0018     second byte of keyboard status
0019  ??                    C  alt_input     db      ?          : 0040:0019     alternate keypad entry
001A  ????                  C  buffer_head   dw      ?          : 0040:001A     keyboard output pointer offset
001C  ????                  C  buffer_tail   dw      ?          : 0040:001C     keyboard input pointer offset
                            C
001E  0010[                 C  kb_buffer     dw      16 dup (?) : 0040:001E     room for 15 entries: head =
          ????              C
                        ]   C
                            C
                            C                                   :               tail implies buffer is empty
                            C
                            C  :      Floppy Diskette Data Area
                            C
003E  ??                    C  seek_status   db      ?          : 0040:003E     floppy disk restore status bits
003F  ??                    C  motor_status  db      ?          : 0040:003F     floppy disk motor status bits
0040  ??                    C  motor_count   db      ?          : 0040:0040     floppy disk turn off counter
0041  ??                    C  diskette_status       db      ?          : 0040:0041     floppy disk driver status byte
                            C
0042                        C  cmd_block     label   byte       : 0040:0042     HDU command block buffer
0042                        C  hd_error      label   byte       : 0040:0042     HDU sense byte buffer
0042  0007[                 C  nec_status    db      7 dup (?)  : 0040:0042     status bytes from NEC controller
          ??                C
                        ]   C
                            C
                            C
                            C  :        Video Display Data Area
                            C
                            C
0049  ??                    C  v_mode                db      ?          : 0040:0049       CRT mode
004A  ????                  C  v_width               dw      ?          : 0040:004A       CRT number of columns (often db)
004C  ????                  C  v_height      dw      ?          : 0040:004C       CRT length of video ram in bytes
004E  ????                  C  v_top                 dw      ?          : 0040:004E       CRT video ram buffer address
0050  0008[                 C  v_curpos      dw      8 dup (?)  : 0040:0050  cursor for each of up to 8 pages
          ????              C
                        ]   C
                            C
                            C
```

```
0060  ????              C  v_cursize     dw      ?        ; 0040:0060      v_curs_type sets cursor value
0062  ??                C  v_apage               db      ?        ; 0040:0062
0063  ????              C  v_base6845    dw      ?        ; 0040:0063
0065  ??                C  v_3x8                 db      ?        ; 0040:0065
0066  ??                C  v_colorpal    db      ?        ; 0040:0066
                        C
                        C  ;     Optional Post Data Area
                        C
0067  ????              C  io_rom_init   dw      ?          ; 0040:0067    option ROM init routine offset
0069  ????              C  io_rom_seg    dw      ?          ; 0040:0069    option RON init routine segment
006B  ??                C  intr_flag     db      ?          ; 0040:006B    occurrence of interrupt flag
                        C
                        C  ;      i8253 p_timer Data Area
                        C
006C  ????              C  t_low_order   dw      ?          ; 0040:006C    low word of i8253 p_timer count
006E  ????              C  t_hi_order    dw      ?          ; 0040:006E    high word of i8253 p_timer count
0070  ??                C  t_overflow    db      ?          ; 0040:0070    time rolled over flag
                        C
                        C  ;     System Data Area
                        C
0071  ??                C  bios_break    db      ?          ; 0040:0071    bit #7 set if break key hit
0072  0000              C  reset_flag    dw      0          ; 0040:0072    = 1234h if keyboard reset hit
                        C
                        C  ;     Fixed Disk Data Area
                        C
0074  ??                C  disk_status   db      ?          ; 0040:0074    fixed disk driver status byte
0075  ??                C  hf_num                db      ?          ; 0040:0075    fixed disk drive count
0076  ??                C  control_byte  db      ?          ; 0040:0076    fixed disk control byte options
0077  ??                C  port_off      db      ?          ; 0040:0077    fixed disk port offset
                        C
                        C  ;     Printer & RS-232 Time-Out Data Areas
                        C
0078  0004[              C  printer_t_out db      4 dup (?)  ; 0040:0078     printer time-out variables
          ??            C
                      ] C
                        C
007C  0004[              C  serial_t_out  db      4 dup (?)  ; 0040:007C     RS-232 time-out variables
          ??            C
                      ] C
                        C
                        C
                        C  ;     Additional Keyboard Data Area
                        C
0080  ????              C  buffer_start  dw      ?          ; 0040:0080    offset of kb_buffer        = 001E
0082  ????              C  buffer_end    dw      ?          ; 0040:0082    offset of kb_buffer_end    = 003E
                        C
                        C  ;--------------------------------------------------------------------------
                        C  ;    Data Area
                        C  ;--------------------------------------------------------------------------
                        C
                        C  ;    Master Table Pointer
                        C
                        C  ;    40:84 -> 40:88 is Also used by Enhanced Graphics Adaptor ROM
                        C
0084  ????????          C  master_tbl_ptrdd      ?          ; 0040:0084    pointer to master table
```

```
                        C
                        C
0088  ??                C  EGA_switches  db      ?          ; 0040:0088 Used by EGA ROM
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    END of location-specific data area. These variables are TEMPORARY
                        C  ;
                        C  ;    kb_here is used only during pwrup0.
                        C  ;    control is used only during pwrup1 for 8087 test
                        C  ;--------------------------------------------------------------------
                        C
                        C
0089  ????              C  control              dw      ?          ; TEMP. 8087 value
                        C
008B  ??                C  kb_here              db      ?          ; TEMP. keyboard presence flag
                        C
0098                    C       org 98h
0098  ??                C  disable_nmi_flag db    ?          ;power up flag to keep track of
                        C                                    ;wheter to disable nmi.
0099                    C  data ends
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;       Video RAM
                        C  ;--------------------------------------------------------------------
                        C
0000                    C  v_ramsegment  public 'RAM'      ; at para_mono
                        C
0000                    C  v_ramends

C060                       code segment  public 'ROM'
                           assume   cs:code, ds:nothing, es:nothing, ss:nothing

C060                       font_lo_8x16  label    byte                    ; 2048 bytes
                        C  include       fontlo16.asm
                        C
C060                    C  fontlo16 proc near      ;   System Font Table for M24
                        C
C060  00 00 00 00 00 00 00 C       DB  000h,000h,000h,000h,000h,000h,000h,000h
      00                C
C068  00 00 00 00 00 00 00 C       DB  000h,000h,000h,000h,000h,000h,000h,000h  ;     0
      00                C
C070  00 00 7E 81 A5 81 81 C       DB  000h,000h,07eh,081h,0a5h,081h,081h,0bdh
      BD                C
C078  99 81 7E 00 00 00 00 C       DB  099h,081h,07eh,000h,000h,000h,000h,000h  ;     1
      00                C
C080  00 00 7E FF DB FF FF C       DB  000h,000h,07eh,0ffh,0dbh,0ffh,0ffh,0c3h
      C3                C
C088  E7 FF 7E 00 00 00 00 C       DB  0e7h,0ffh,07eh,000h,000h,000h,000h,000h  ;     2
      00                C
C090  00 00 00 36 7F 7F 7F C       DB  000h,000h,000h,036h,07fh,07fh,07fh,07fh
      7F                C
C098  3E 1C 08 00 00 00 00 C       DB  03eh,01ch,008h,000h,000h,000h,000h,000h  ;     3
      00                C
C0A0  00 00 00 08 1C 3E 7F C       DB  000h,000h,000h,008h,01ch,03eh,07fh,03eh
      3E                C
```

```
C0A8  1C 08 00 00 00 00 00  C       DB  01ch,008h,000h,000h,000h,000h,000h,000h  ;      4
      00                    C
C0B0  00 00 18 3C 3C E7 E7  C       DB  000h,000h,018h,03ch,03ch,0e7h,0e7h,0e7h
      E7                    C
C0B8  18 18 3C 00 00 00 00  C       DB  018h,018h,03ch,000h,000h,000h,000h,000h  ;      5
      00                    C
C0C0  00 00 18 3C 7E FF FF  C       DB  000h,000h,018h,03ch,07eh,0ffh,0ffh,07eh
      7E                    C
C0C8  18 18 3C 00 00 00 00  C       DB  018h,018h,03ch,000h,000h,000h,000h,000h  ;      6
      00                    C
C0D0  00 00 00 00 00 18 3C  C       DB  000h,000h,000h,000h,000h,018h,03ch,03ch
      3C                    C
C0D8  18 00 00 00 00 00 00  C       DB  018h,000h,000h,000h,000h,000h,000h,000h  ;      7
      00                    C
C0E0  FF FF FF FF FF E7 C3  C       DB  0ffh,0ffh,0ffh,0ffh,0ffh,0e7h,0c3h,0c3h
      C3                    C
C0E8  E7 FF FF FF FF FF FF  C       DB  0e7h,0ffh,0ffh,0ffh,0ffh,0ffh,0ffh,0ffh  ;      8
      FF                    C
C0F0  00 00 00 00 3C 24 42  C       DB  000h,000h,000h,000h,03ch,024h,042h,042h
      42                    C
C0F8  24 3C 00 00 00 00 00  C       DB  024h,03ch,000h,000h,000h,000h,000h,000h  ;      9
      00                    C
C100  FF FF FF FF C3 DB BD  C       DB  0ffh,0ffh,0ffh,0ffh,0c3h,0dbh,0bdh,0bdh
      BD                    C
C108  DB C3 FF FF FF FF FF  C       DB  0dbh,0c3h,0ffh,0ffh,0ffh,0ffh,0ffh,0ffh  ;      a
      FF                    C
C110  00 00 1F 07 0D 19 78  C       DB  000h,000h,01fh,007h,00dh,019h,078h,0cch
      CC                    C
C118  CC CC 78 00 00 00 00  C       DB  0cch,0cch,078h,000h,000h,000h,000h,000h  ;      b
      00                    C
C120  00 00 3C 66 66 66 3C  C       DB  000h,000h,03ch,066h,066h,066h,03ch,018h
      18                    C
C128  7E 18 18 00 00 00 00  C       DB  07eh,018h,018h,000h,000h,000h,000h,000h  ;      c
      00                    C
C130  00 00 0C 0A 09 09 09  C       DB  000h,000h,00ch,00ah,009h,009h,009h,00ah
      0A                    C
C138  08 38 78 78 30 00 00  C       DB  008h,038h,078h,078h,030h,000h,000h,000h  ;      d
      00                    C
C140  00 00 1F 11 1F 11 11  C       DB  000h,000h,01fh,011h,01fh,011h,011h,011h
      11                    C
C148  13 37 77 72 20 00 00  C       DB  013h,037h,077h,072h,020h,000h,000h,000h  ;      e
      00                    C
C150  00 00 18 18 DB 3C E7  C       DB  000h,000h,018h,018h,0dbh,03ch,0e7h,03ch
      3C                    C
C158  DB 18 18 00 00 00 00  C       DB  0dbh,018h,018h,000h,000h,000h,000h,000h  ;      f
      00                    C
C160  00 00 40 60 70 7C 7F  C       DB  000h,000h,040h,060h,070h,07ch,07fh,07ch
      7C                    C
C168  70 60 40 00 00 00 00  C       DB  070h,060h,040h,000h,000h,000h,000h,000h  ;     10
      00                    C
C170  00 00 01 03 07 1F 7F  C       DB  000h,000h,001h,003h,007h,01fh,07fh,01fh
      1F                    C
C178  07 03 01 00 00 00 00  C       DB  007h,003h,001h,000h,000h,000h,000h,000h  ;     11
      00                    C
C180  00 00 18 3C 7E 18 18  C       DB  000h,000h,018h,03ch,07eh,018h,018h,018h
```

```
      18                        C
C188  7E 3C 18 00 00 00 00       C        DB  07eh,03ch,018h,000h,000h,000h,000h,000h  ;      12
      00                        C
C190  00 00 33 33 33 33 33       C        DB  000h,000h,033h,033h,033h,033h,033h,033h
      33                        C
C198  00 33 33 00 00 00 00       C        DB  000h,033h,033h,000h,000h,000h,000h,000h  ;      13
      00                        C
C1A0  00 00 7F DB DB DB 7B       C        DB  000h,000h,07fh,0dbh,0dbh,0dbh,07bh,01bh
      1B                        C
C1A8  1B 1B 1B 00 00 00 00       C        DB  01bh,01bh,01bh,000h,000h,000h,000h,000h  ;      14
      00                        C
C1B0  00 3E 63 30 1C 36 63       C        DB  000h,03eh,063h,030h,01ch,036h,063h,063h
      63                        C
C1B8  36 1C 06 63 3E 00 00       C        DB  036h,01ch,006h,063h,03eh,000h,000h,000h  ;      15
      00                        C
C1C0  00 00 00 00 00 00 00       C        DB  000h,000h,000h,000h,000h,000h,000h,000h
      00                        C
C1C8  7F 7F 7F 00 00 00 00       C        DB  07fh,07fh,07fh,000h,000h,000h,000h,000h  ;      16
      00                        C
C1D0  00 00 18 3C 7E 18 18       C        DB  000h,000h,018h,03ch,07eh,018h,018h,018h
      18                        C
C1D8  7E 3C 18 FF 00 00 00       C        DB  07eh,03ch,018h,0ffh,000h,000h,000h,000h  ;      17
      00                        C
C1E0  00 00 18 3C 7E 18 18       C        DB  000h,000h,018h,03ch,07eh,018h,018h,018h
      18                        C
C1E8  18 18 18 00 00 00 00       C        DB  018h,018h,018h,000h,000h,000h,000h,000h  ;      18
      00                        C
C1F0  00 00 18 18 18 18 18       C        DB  000h,000h,018h,018h,018h,018h,018h,018h
      18                        C
C1F8  7E 3C 18 00 00 00 00       C        DB  07eh,03ch,018h,000h,000h,000h,000h,000h  ;      19
      00                        C
C200  00 00 00 00 0C 06 7F       C        DB  000h,000h,000h,000h,00ch,006h,07fh,006h
      06                        C
C208  0C 00 00 00 00 00 00       C        DB  00ch,000h,000h,000h,000h,000h,000h,000h  ;      1a
      00                        C
C210  00 00 00 00 18 30 7F       C        DB  000h,000h,000h,000h,018h,030h,07fh,030h
      30                        C
C218  18 00 00 00 00 00 00       C        DB  018h,000h,000h,000h,000h,000h,000h,000h  ;      1b
      00                        C
C220  00 00 00 00 60 60 60       C        DB  000h,000h,000h,000h,060h,060h,060h,060h
      60                        C
C228  7F 7F 00 00 00 00 00       C        DB  07fh,07fh,000h,000h,000h,000h,000h,000h  ;      1c
      00                        C
C230  00 00 00 00 24 42 FF       C        DB  000h,000h,000h,000h,024h,042h,0ffh,042h
      42                        C
C238  24 00 00 00 00 00 00       C        DB  024h,000h,000h,000h,000h,000h,000h,000h  ;      1d
      00                        C
C240  00 00 00 00 00 00 00       C        DB  000h,000h,000h,000h,000h,000h,000h,018h
      18                        C
C248  3C 7E FF 00 00 00 00       C        DB  03ch,07eh,0ffh,000h,000h,000h,000h,000h  ;      1e
      00                        C
C250  00 00 00 00 00 FF 7E       C        DB  000h,000h,000h,000h,000h,0ffh,07eh,03ch
      3C                        C
C258  18 00 00 00 00 00 00       C        DB  018h,000h,000h,000h,000h,000h,000h,000h  ;      1f
      00                        C
```

```
C260  00 00 00 00 00 00 00  C       DB  000h,000h,000h,000h,000h,000h,000h,000h
      00                    C
C268  00 00 00 00 00 00 00  C       DB  000h,000h,000h,000h,000h,000h,000h,000h  ;' ' 20
      00                    C
C270  00 00 18 3C 3C 3C 18  C       DB  000h,000h,018h,03ch,03ch,03ch,018h,018h
      18                    C
C278  00 18 18 00 00 00 00  C       DB  000h,018h,018h,000h,000h,000h,000h,000h  ;'!' 21
      00                    C
C280  00 66 66 66 24 00 00  C       DB  000h,066h,066h,066h,024h,000h,000h,000h
      00                    C
C288  00 00 00 00 00 00 00  C       DB  000h,000h,000h,000h,000h,000h,000h,000h  ;'" ' 22
      00                    C
C290  00 00 36 36 7F 36 36  C       DB  000h,000h,036h,036h,07fh,036h,036h,036h
      36                    C
C298  7F 36 36 00 00 00 00  C       DB  07fh,036h,036h,000h,000h,000h,000h,000h  ;'#' 23
      00                    C
C2A0  08 08 3E 63 60 60 3E  C       DB  008h,008h,03eh,063h,060h,060h,03eh,003h
      03                    C
C2A8  03 63 3E 08 08 00 00  C       DB  003h,063h,03eh,008h,008h,000h,000h,000h  ;'$' 24
      00                    C
C2B0  00 00 00 61 63 06 0C  C       DB  000h,000h,000h,061h,063h,006h,00ch,018h
      18                    C
C2B8  30 63 43 00 00 00 00  C       DB  030h,063h,043h,000h,000h,000h,000h,000h  ;'%' 25
      00                    C
C2C0  00 00 1C 36 36 1C 3B  C       DB  000h,000h,01ch,036h,036h,01ch,03bh,06eh
      6E                    C
C2C8  66 66 3B 00 00 00 00  C       DB  066h,066h,03bh,000h,000h,000h,000h,000h  ;'&' 26
      00                    C
C2D0  00 30 30 30 60 00 00  C       DB  000h,030h,030h,030h,060h,000h,000h,000h
      00                    C
C2D8  00 00 00 00 00 00 00  C       DB  000h,000h,000h,000h,000h,000h,000h,000h  ;''' 27
      00                    C
C2E0  00 00 0C 18 30 30 30  C       DB  000h,000h,00ch,018h,030h,030h,030h,030h
      30                    C
C2E8  30 18 0C 00 00 00 00  C       DB  030h,018h,00ch,000h,000h,000h,000h,000h  ;'(' 28
      00                    C
C2F0  00 00 30 18 0C 0C 0C  C       DB  000h,000h,030h,018h,00ch,00ch,00ch,00ch
      0C                    C
C2F8  0C 18 30 00 00 00 00  C       DB  00ch,018h,030h,000h,000h,000h,000h,000h  ;')' 29
      00                    C
C300  00 00 00 00 66 3C 7E  C       DB  000h,000h,000h,000h,066h,03ch,07eh,03ch
      3C                    C
C308  66 00 00 00 00 00 00  C       DB  066h,000h,000h,000h,000h,000h,000h,000h  ;'*' 2a
      00                    C
C310  00 00 00 00 18 18 7E  C       DB  000h,000h,000h,000h,018h,018h,07eh,018h
      18                    C
C318  18 00 00 00 00 00 00  C       DB  018h,000h,000h,000h,000h,000h,000h,000h  ;'+' 2b
      00                    C
C320  00 00 00 00 00 00 00  C       DB  000h,000h,000h,000h,000h,000h,000h,000h
      00                    C
C328  18 18 18 30 00 00 00  C       DB  018h,018h,018h,030h,000h,000h,000h,000h  ;',' 2c
      00                    C
C330  00 00 00 00 00 00 7E  C       DB  000h,000h,000h,000h,000h,000h,07eh,000h
      00                    C
C338  00 00 00 00 00 00 00  C       DB  000h,000h,000h,000h,000h,000h,000h,000h  ;'-' 2d
```

```
      00                        C
C340  00 00 00 00 00 00 00      C       DB  000h,000h,000h,000h,000h,000h,000h,000h
      00                        C
C348  00 18 18 00 00 00 00      C       DB  000h,018h,018h,000h,000h,000h,000h,000h  ;'.' 2e
      00                        C
C350  00 00 01 03 06 0C 18      C       DB  000h,000h,001h,003h,006h,00ch,018h,030h
      30                        C
C358  60 40 00 00 00 00 00      C       DB  060h,040h,000h,000h,000h,000h,000h,000h  ;'/' 2f
      00                        C
                                C
C360  00 00 3E 63 67 6F 7B      C       DB  000h,000h,03eh,063h,067h,06fh,07bh,073h
      73                        C
C368  63 63 3E 00 00 00 00      C       DB  063h,063h,03eh,000h,000h,000h,000h,000h  ;'0' 30
      00                        C
                                C
C370  00 00 0C 1C 3C 0C 0C      C       DB  000h,000h,00ch,01ch,03ch,00ch,00ch,00ch
      0C                        C
C378  0C 0C 3F 00 00 00 00      C       DB  00ch,00ch,03fh,000h,000h,000h,000h,000h  ;'1' 31
      00                        C
C380  00 00 3E 63 03 06 0C      C       DB  000h,000h,03eh,063h,003h,006h,00ch,018h
      18                        C
C388  30 63 7F 00 00 00 00      C       DB  030h,063h,07fh,000h,000h,000h,000h,000h  ;'2' 32
      00                        C
C390  00 00 3E 63 03 03 1E      C       DB  000h,000h,03eh,063h,003h,003h,01eh,003h
      03                        C
C398  03 63 3E 00 00 00 00      C       DB  003h,063h,03eh,000h,000h,000h,000h,000h  ;'3' 33
      00                        C
C3A0  00 00 06 0E 1E 36 66      C       DB  000h,000h,006h,00eh,01eh,036h,066h,07fh
      7F                        C
C3A8  06 06 0F 00 00 00 00      C       DB  006h,006h,00fh,000h,000h,000h,000h,000h  ;'4' 34
      00                        C
C3B0  00 00 7E 60 60 60 7E      C       DB  000h,000h,07eh,060h,060h,060h,07eh,003h
      03                        C
C3B8  03 63 3E 00 00 00 00      C       DB  003h,063h,03eh,000h,000h,000h,000h,000h  ;'5' 35
      00                        C
C3C0  00 00 1C 30 60 60 7E      C       DB  000h,000h,01ch,030h,060h,060h,07eh,063h
      63                        C
C3C8  63 63 3E 00 00 00 00      C       DB  063h,063h,03eh,000h,000h,000h,000h,000h  ;'6' 36
      00                        C
C3D0  00 00 7F 63 03 06 0C      C       DB  000h,000h,07fh,063h,003h,006h,00ch,018h
      18                        C
C3D8  18 18 18 00 00 00 00      C       DB  018h,018h,018h,000h,000h,000h,000h,000h  ;'7' 37
      00                        C
C3E0  00 00 3E 63 63 63 3E      C       DB  000h,000h,03eh,063h,063h,063h,03eh,063h
      63                        C
C3E8  63 63 3E 00 00 00 00      C       DB  063h,063h,03eh,000h,000h,000h,000h,000h  ;'8' 38
      00                        C
C3F0  00 00 3E 63 63 63 3F      C       DB  000h,000h,03eh,063h,063h,063h,03fh,003h
      03                        C
C3F8  03 06 1C 00 00 00 00      C       DB  003h,006h,01ch,000h,000h,000h,000h,000h  ;'9' 39
      00                        C
C400  00 00 00 00 18 18 00      C       DB  000h,000h,000h,000h,018h,018h,000h,000h
      00                        C
C408  00 18 18 00 00 00 00      C       DB  000h,018h,018h,000h,000h,000h,000h,000h  ;':' 3a
      00                        C
```

```
C410  00 00 00 00 18 18 00  C       DB  000h,000h,000h,000h,018h,018h,000h,000h
      00                    C
C418  00 18 18 30 00 00 00  C       DB  000h,018h,018h,030h,000h,000h,000h,000h  ;';' 3b
      00                    C
C420  00 00 06 0C 18 30 60  C       DB  000h,000h,006h,00ch,018h,030h,060h,030h
      30                    C
C428  18 0C 06 00 00 00 00  C       DB  018h,00ch,006h,000h,000h,000h,000h,000h  ;'<' 3c
      00                    C
C430  00 00 00 00 7E 00 00  C       DB  000h,000h,000h,000h,07eh,000h,000h,000h
      00                    C
C438  7E 00 00 00 00 00 00  C       DB  07eh,000h,000h,000h,000h,000h,000h,000h  ;'=' 3d
      00                    C
C440  00 00 60 30 18 0C 06  C       DB  000h,000h,060h,030h,018h,00ch,006h,00ch
      0C                    C
C448  18 30 60 00 00 00 00  C       DB  018h,030h,060h,000h,000h,000h,000h,000h  ;'>' 3e
      00                    C
C450  00 00 3E 63 63 06 0C  C       DB  000h,000h,03eh,063h,063h,006h,00ch,00ch
      0C                    C
C458  00 0C 0C 00 00 00 00  C       DB  000h,00ch,00ch,000h,000h,000h,000h,000h  ;'?' 3f
      00                    C
C460  00 00 3E 63 63 6F 6F  C       DB  000h,000h,03eh,063h,063h,06fh,06fh,06fh
      6F                    C
C468  6E 60 3E 00 00 00 00  C       DB  06eh,060h,03eh,000h,000h,000h,000h,000h  ;'@' 40
      00                    C
C470  00 00 08 1C 36 63 63  C       DB  000h,000h,008h,01ch,036h,063h,063h,07fh
      7F                    C
C478  63 63 63 00 00 00 00  C       DB  063h,063h,063h,000h,000h,000h,000h,000h  ;'A' 41
      00                    C
C480  00 00 7E 33 33 33 3E  C       DB  000h,000h,07eh,033h,033h,033h,03eh,033h
      33                    C
C488  33 33 7E 00 00 00 00  C       DB  033h,033h,07eh,000h,000h,000h,000h,000h  ;'B' 42
      00                    C
C490  00 00 1E 33 60 60 60  C       DB  000h,000h,01eh,033h,060h,060h,060h,060h
      60                    C
C498  60 33 1E 00 00 00 00  C       DB  060h,033h,01eh,000h,000h,000h,000h,000h  ;'C' 43
      00                    C
C4A0  00 00 7C 36 33 33 33  C       DB  000h,000h,07ch,036h,033h,033h,033h,033h
      33                    C
C4A8  33 36 7C 00 00 00 00  C       DB  033h,036h,07ch,000h,000h,000h,000h,000h  ;'D' 44
      00                    C
C4B0  00 00 7F 33 30 34 3C  C       DB  000h,000h,07fh,033h,030h,034h,03ch,034h
      34                    C
C4B8  30 33 7F 00 00 00 00  C       DB  030h,033h,07fh,000h,000h,000h,000h,000h  ;'E' 45
      00                    C
C4C0  00 00 7F 33 30 34 3C  C       DB  000h,000h,07fh,033h,030h,034h,03ch,034h
      34                    C
C4C8  30 30 78 00 00 00 00  C       DB  030h,030h,078h,000h,000h,000h,000h,000h  ;'F' 46
      00                    C
C4D0  00 00 1E 33 60 60 60  C       DB  000h,000h,01eh,033h,060h,060h,060h,06fh
      6F                    C
C4D8  63 33 1D 00 00 00 00  C       DB  063h,033h,01dh,000h,000h,000h,000h,000h  ;'G' 47
      00                    C
C4E0  00 00 63 63 63 63 7F  C       DB  000h,000h,063h,063h,063h,063h,07fh,063h
      63                    C
C4E8  63 63 63 00 00 00 00  C       DB  063h,063h,063h,000h,000h,000h,000h,000h  ;'H' 48
```

```
      00                        C
C4F0  00 00 3C 18 18 18 18      C       DB  000h,000h,03ch,018h,018h,018h,018h,018h
      18                        C
C4F8  18 18 3C 00 00 00 00      C       DB  018h,018h,03ch,000h,000h,000h,000h,000h  ;'I' 49
      00                        C
C500  00 00 0F 06 06 06 06      C       DB  000h,000h,00fh,006h,006h,006h,006h,006h
      06                        C
C508  66 66 3C 00 00 00 00      C       DB  066h,066h,03ch,000h,000h,000h,000h,000h  ;'J' 4a
      00                        C
C510  00 00 73 33 36 36 3C      C       DB  000h,000h,073h,033h,036h,036h,03ch,036h
      36                        C
C518  36 33 73 00 00 00 00      C       DB  036h,033h,073h,000h,000h,000h,000h,000h  ;'K' 4b
      00                        C
C520  00 00 78 30 30 30 30      C       DB  000h,000h,078h,030h,030h,030h,030h,030h
      30                        C
C528  30 33 7F 00 00 00 00      C       DB  030h,033h,07fh,000h,000h,000h,000h,000h  ;'L' 4c
      00                        C
C530  00 00 63 77 7F 6B 63      C       DB  000h,000h,063h,077h,07fh,06bh,063h,063h
      63                        C
C538  63 63 63 00 00 00 00      C       DB  063h,063h,063h,000h,000h,000h,000h,000h  ;'M' 4d
      00                        C
C540  00 00 63 73 7B 7F 6F      C       DB  000h,000h,063h,073h,07bh,07fh,06fh,067h
      67                        C
C548  63 63 63 00 00 00 00      C       DB  063h,063h,063h,000h,000h,000h,000h,000h  ;'N' 4e
      00                        C
C550  00 00 1C 36 63 63 63      C       DB  000h,000h,01ch,036h,063h,063h,063h,063h
      63                        C
C558  63 36 1C 00 00 00 00      C       DB  063h,036h,01ch,000h,000h,000h,000h,000h  ;'O' 4f
      00                        C
C560  00 00 7E 33 33 33 3E      C       DB  000h,000h,07eh,033h,033h,033h,03eh,030h
      30                        C
C568  30 30 78 00 00 00 00      C       DB  030h,030h,078h,000h,000h,000h,000h,000h  ;'P' 50
      00                        C
C570  00 00 1C 36 63 63 63      C       DB  000h,000h,01ch,036h,063h,063h,063h,063h
      63                        C
C578  6B 3E 1C 06 03 00 00      C       DB  06bh,03eh,01ch,006h,003h,000h,000h,000h  ;'Q' 51
      00                        C
C580  00 00 7E 33 33 33 3E      C       DB  000h,000h,07eh,033h,033h,033h,03eh,036h
      36                        C
C588  33 33 73 00 00 00 00      C       DB  033h,033h,073h,000h,000h,000h,000h,000h  ;'R' 52
      00                        C
C590  00 00 3E 63 63 30 1C      C       DB  000h,000h,03eh,063h,063h,030h,01ch,006h
      06                        C
C598  63 63 3E 00 00 00 00      C       DB  063h,063h,03eh,000h,000h,000h,000h,000h  ;'S' 53
      00                        C
C5A0  00 00 7E 5A 18 18 18      C       DB  000h,000h,07eh,05ah,018h,018h,018h,018h
      18                        C
C5A8  18 18 3C 00 00 00 00      C       DB  018h,018h,03ch,000h,000h,000h,000h,000h  ;'T' 54
      00                        C
C5B0  00 00 63 63 63 63 63      C       DB  000h,000h,063h,063h,063h,063h,063h,063h
      63                        C
C5B8  63 63 3E 00 00 00 00      C       DB  063h,063h,03eh,000h,000h,000h,000h,000h  ;'U' 55
      00                        C
C5C0  00 00 63 63 63 63 63      C       DB  000h,000h,063h,063h,063h,063h,063h,063h
      63                        C
```

```
C5C8  36 1C 08 00 00 00 00  C       DB  036h,01ch,008h,000h,000h,000h,000h,000h  ;'V' 56
      00                    C
C5D0  00 00 63 63 63 63 63  C       DB  000h,000h,063h,063h,063h,063h,063h,06bh
      6B                    C
C5D8  6B 7F 36 00 00 00 00  C       DB  06bh,07fh,036h,000h,000h,000h,000h,000h  ;'W' 57
      00                    C
C5E0  00 00 63 63 63 36 1C  C       DB  000h,000h,063h,063h,063h,036h,01ch,036h
      36                    C
C5E8  63 63 63 00 00 00 00  C       DB  063h,063h,063h,000h,000h,000h,000h,000h  ;'X' 58
      00                    C
C5F0  00 00 66 66 66 66 66  C       DB  000h,000h,066h,066h,066h,066h,066h,03ch
      3C                    C
C5F8  18 18 3C 00 00 00 00  C       DB  018h,018h,03ch,000h,000h,000h,000h,000h  ;'Y' 59
      00                    C
C600  00 00 7F 63 06 0C 18  C       DB  000h,000h,07fh,063h,006h,00ch,018h,030h
      30                    C
C608  60 63 7F 00 00 00 00  C       DB  060h,063h,07fh,000h,000h,000h,000h,000h  ;'Z' 5a
      00                    C
C610  00 00 3C 30 30 30 30  C       DB  000h,000h,03ch,030h,030h,030h,030h,030h
      30                    C
C618  30 30 3C 00 00 00 00  C       DB  030h,030h,03ch,000h,000h,000h,000h,000h  ;'[' 5b
      00                    C
C620  00 00 40 60 30 18 0C  C       DB  000h,000h,040h,060h,030h,018h,00ch,006h
      06                    C
C628  03 01 00 00 00 00 00  C       DB  003h,001h,000h,000h,000h,000h,000h,000h  ;'\' 5c
      00                    C
C630  00 00 3C 0C 0C 0C 0C  C       DB  000h,000h,03ch,00ch,00ch,00ch,00ch,00ch
      0C                    C
C638  0C 0C 3C 00 00 00 00  C       DB  00ch,00ch,03ch,000h,000h,000h,000h,000h  ;']' 5d
      00                    C
C640  08 1C 36 63 00 00 00  C       DB  008h,01ch,036h,063h,000h,000h,000h,000h
      00                    C
C648  00 00 00 00 00 00 00  C       DB  000h,000h,000h,000h,000h,000h,000h,000h  ;'^' 5e
      00                    C
C650  00 00 00 00 00 00 00  C       DB  000h,000h,000h,000h,000h,000h,000h,000h
      00                    C
C658  00 00 00 00 00 00 7F  C       DB  000h,000h,000h,000h,000h,000h,07fh,000h  ;'_' 5f
      00                    C
C660  18 18 0C 00 00 00 00  C       DB  018h,018h,00ch,000h,000h,000h,000h,000h
      00                    C
C668  00 00 00 00 00 00 00  C       DB  000h,000h,000h,000h,000h,000h,000h,000h  ;'`' 60
      00                    C
C670  00 00 00 00 00 3C 06  C       DB  000h,000h,000h,000h,000h,03ch,006h,03eh
      3E                    C
C678  66 66 3B 00 00 00 00  C       DB  066h,066h,03bh,000h,000h,000h,000h,000h  ;'a' 61
      00                    C
C680  00 00 70 30 30 3E 33  C       DB  000h,000h,070h,030h,030h,03eh,033h,033h
      33                    C
C688  33 33 6E 00 00 00 00  C       DB  033h,033h,06eh,000h,000h,000h,000h,000h  ;'b' 62
      00                    C
C690  00 00 00 00 00 3E 63  C       DB  000h,000h,000h,000h,000h,03eh,063h,060h
      60                    C
C698  60 63 3E 00 00 00 00  C       DB  060h,063h,03eh,000h,000h,000h,000h,000h  ;'c' 63
      00                    C
C6A0  00 00 0E 06 06 3E 66  C       DB  000h,000h,00eh,006h,006h,03eh,066h,066h
```

```
      66                        C
C6A8  66 66 3B 00 00 00 00      C       DB  066h,066h,03bh,000h,000h,000h,000h,000h  ;'d' 64
      00                        C
C6B0  00 00 00 00 00 3E 63      C       DB  000h,000h,000h,000h,000h,03eh,063h,07fh
      7F                        C
C6B8  60 63 3E 00 00 00 00      C       DB  060h,063h,03eh,000h,000h,000h,000h,000h  ;'e' 65
      00                        C
C6C0  00 00 1E 33 30 7C 30      C       DB  000h,000h,01eh,033h,030h,07ch,030h,030h
      30                        C
C6C8  30 30 78 00 00 00 00      C       DB  030h,030h,078h,000h,000h,000h,000h,000h  ;'f' 66
      00                        C
C6D0  00 00 00 00 00 3B 66      C       DB  000h,000h,000h,000h,000h,03bh,066h,066h
      66                        C
C6D8  66 66 3E 06 66 3C 00      C       DB  066h,066h,03eh,006h,066h,03ch,000h,000h  ;'g' 67
      00                        C
C6E0  00 00 70 30 30 36 3B      C       DB  000h,000h,070h,030h,030h,036h,03bh,033h
      33                        C
C6E8  33 33 73 00 00 00 00      C       DB  033h,033h,073h,000h,000h,000h,000h,000h  ;'h' 68
      00                        C
C6F0  00 00 0C 0C 00 1C 0C      C       DB  000h,000h,00ch,00ch,000h,01ch,00ch,00ch
      0C                        C
C6F8  0C 0C 1E 00 00 00 00      C       DB  00ch,00ch,01eh,000h,000h,000h,000h,000h  ;'i' 69
      00                        C
C700  00 00 0C 0C 00 1C 0C      C       DB  000h,000h,00ch,00ch,000h,01ch,00ch,00ch
      0C                        C
C708  0C 0C 0C 0C CC 78 00      C       DB  00ch,00ch,00ch,00ch,0cch,078h,000h,000h  ;'j' 6a
      00                        C
C710  00 00 70 30 30 33 36      C       DB  000h,000h,070h,030h,030h,033h,036h,03ch
      3C                        C
C718  36 33 73 00 00 00 00      C       DB  036h,033h,073h,000h,000h,000h,000h,000h  ;'k' 6b
      00                        C
C720  00 00 1C 0C 0C 0C 0C      C       DB  000h,000h,01ch,00ch,00ch,00ch,00ch,00ch
      0C                        C
C728  0C 0C 1E 00 00 00 00      C       DB  00ch,00ch,01eh,000h,000h,000h,000h,000h  ;'l' 6c
      00                        C
C730  00 00 00 00 00 66 7F      C       DB  000h,000h,000h,000h,000h,066h,07fh,06bh
      6B                        C
C738  6B 6B 6B 00 00 00 00      C       DB  06bh,06bh,06bh,000h,000h,000h,000h,000h  ;'m' 6d
      00                        C
C740  00 00 00 00 00 6E 33      C       DB  000h,000h,000h,000h,000h,06eh,033h,033h
      33                        C
C748  33 33 33 00 00 00 00      C       DB  033h,033h,033h,000h,000h,000h,000h,000h  ;'n' 6e
      00                        C
C750  00 00 00 00 00 3E 63      C       DB  000h,000h,000h,000h,000h,03eh,063h,063h
      63                        C
C758  63 63 3E 00 00 00 00      C       DB  063h,063h,03eh,000h,000h,000h,000h,000h  ;'o' 6f
      00                        C
C760  00 00 00 00 00 6E 33      C       DB  000h,000h,000h,000h,000h,06eh,033h,033h
      33                        C
C768  33 33 3E 30 30 78 00      C       DB  033h,033h,03eh,030h,030h,078h,000h,000h  ;'p' 70
      00                        C
C770  00 00 00 00 00 3B 66      C       DB  000h,000h,000h,000h,000h,03bh,066h,066h
      66                        C
C778  66 66 3E 06 06 0F 00      C       DB  066h,066h,03eh,006h,006h,00fh,000h,000h  ;'q' 71
      00                        C
```

```
C780  00 00 00 00 00 6E 33  C       DB  000h,000h,000h,000h,000h,06eh,033h,030h
      30                    C
C788  30 30 78 00 00 00 00  C       DB  030h,030h,078h,000h,000h,000h,000h,000h  ;'r' 72
      00                    C
C790  00 00 00 00 00 3E 63  C       DB  000h,000h,000h,000h,000h,03eh,063h,038h
      38                    C
C798  0E 63 3E 00 00 00 00  C       DB  00eh,063h,03eh,000h,000h,000h,000h,000h  ;'s' 73
      00                    C
C7A0  00 00 00 08 18 7E 18  C       DB  000h,000h,000h,008h,018h,07eh,018h,018h
      18                    C
C7A8  18 1B 0E 00 00 00 00  C       DB  018h,01bh,00eh,000h,000h,000h,000h,000h  ;'t' 74
      00                    C
C7B0  00 00 00 00 00 66 66  C       DB  000h,000h,000h,000h,000h,066h,066h,066h
      66                    C
C7B8  66 66 3B 00 00 00 00  C       DB  066h,066h,03bh,000h,000h,000h,000h,000h  ;'u' 75
      00                    C
C7C0  00 00 00 00 00 63 63  C       DB  000h,000h,000h,000h,000h,063h,063h,063h
      63                    C
C7C8  36 1C 08 00 00 00 00  C       DB  036h,01ch,008h,000h,000h,000h,000h,000h  ;'v' 76
      00                    C
C7D0  00 00 00 00 00 63 63  C       DB  000h,000h,000h,000h,000h,063h,063h,06bh
      6B                    C
C7D8  6B 7F 36 00 00 00 00  C       DB  06bh,07fh,036h,000h,000h,000h,000h,000h  ;'w' 77
      00                    C
C7E0  00 00 00 00 00 63 36  C       DB  000h,000h,000h,000h,000h,063h,036h,01ch
      1C                    C
C7E8  1C 36 63 00 00 00 00  C       DB  01ch,036h,063h,000h,000h,000h,000h,000h  ;'x' 78
      00                    C
C7F0  00 00 00 00 00 63 66  C       DB  000h,000h,000h,000h,000h,063h,066h,066h
      66                    C
C7F8  66 66 3E 06 66 3C 00  C       DB  066h,066h,03eh,006h,066h,03ch,000h,000h  ;'y' 79
      00                    C
C800  00 00 00 00 00 7F 66  C       DB  000h,000h,000h,000h,000h,07fh,066h,00ch
      0C                    C
C808  18 33 7F 00 00 00 00  C       DB  018h,033h,07fh,000h,000h,000h,000h,000h  ;'z' 7a
      00                    C
C810  00 00 0E 18 18 18 70  C       DB  000h,000h,00eh,018h,018h,018h,070h,018h
      18                    C
C818  18 18 0E 00 00 00 00  C       DB  018h,018h,00eh,000h,000h,000h,000h,000h  ;'{' 7b
      00                    C
C820  00 00 18 18 18 18 00  C       DB  000h,000h,018h,018h,018h,018h,000h,018h
      18                    C
C828  18 18 18 00 00 00 00  C       DB  018h,018h,018h,000h,000h,000h,000h,000h  ;'|' 7c
      00                    C
C830  00 00 70 18 18 18 0E  C       DB  000h,000h,070h,018h,018h,018h,00eh,018h
      18                    C
C838  18 18 70 00 00 00 00  C       DB  018h,018h,070h,000h,000h,000h,000h,000h  ;'}' 7d
      00                    C
C840  00 00 3B 6E 00 00 00  C       DB  000h,000h,03bh,06eh,000h,000h,000h,000h
      00                    C
C848  00 00 00 00 00 00 00  C       DB  000h,000h,000h,000h,000h,000h,000h,000h  ;'~' 7e
      00                    C
C850  00 00 00 00 08 1C 36  C       DB  000h,000h,000h,000h,008h,01ch,036h,063h
      63                    C
C858  63 7F 00 00 00 00 00  C       DB  063h,07fh,000h,000h,000h,000h,000h,000h  ;'' 7f
```

```
      00                    C
                            C  ;End of font matrix
                            C
                            C  fontlo16 endp


C860                        C     font_hi_8x8   label    byte                    ; 1024 bytes
                            C  include        fonthi8.asm
C860                        C  fonthi8 proc near
                            C  ;    SystemFont        ; <hi_mediumres> (m24) 8 x 8 font table for m24
                            C
C860  3C 66 60 66 3C 0C 06  C        DB  03ch,066h,060h,066h,03ch,00ch,006h,03ch  ;     80
      3C                    C
C868  00 66 00 66 66 66 3F  C        DB  000h,066h,000h,066h,066h,066h,03fh,000h  ;     81
      00                    C
C870  0E 00 3C 66 7E 60 3C  C        DB  00eh,000h,03ch,066h,07eh,060h,03ch,000h  ;     82
      00                    C
C878  7E C3 3C 06 3E 66 3F  C        DB  07eh,0c3h,03ch,006h,03eh,066h,03fh,000h  ;     83
      00                    C
C880  66 00 3C 06 3E 66 3F  C        DB  066h,000h,03ch,006h,03eh,066h,03fh,000h  ;     84
      00                    C
C888  70 00 3C 06 3E 66 3F  C        DB  070h,000h,03ch,006h,03eh,066h,03fh,000h  ;     85
      00                    C
                            C                                                        ; FONT(S) 86
                            C
C890  18 18 3C 06 3E 66 3F  C              DB  018h,018h,03ch,006h,03eh,066h,03fh,000h  ;     86
      00                    C
                            C
C898  00 00 3C 60 60 3C 06  C        DB  000h,000h,03ch,060h,060h,03ch,006h,01ch  ;     87
      1C                    C
C8A0  7E C3 3C 66 7E 60 3C  C        DB  07eh,0c3h,03ch,066h,07eh,060h,03ch,000h  ;     88
      00                    C
C8A8  66 00 3C 66 7E 60 3C  C        DB  066h,000h,03ch,066h,07eh,060h,03ch,000h  ;     89
      00                    C
C8B0  70 00 3C 66 7E 60 3C  C        DB  070h,000h,03ch,066h,07eh,060h,03ch,000h  ;     8a
      00                    C
C8B8  66 00 38 18 18 18 3C  C        DB  066h,000h,038h,018h,018h,018h,03ch,000h  ;     8b
      00                    C
C8C0  7C C6 38 18 18 18 3C  C        DB  07ch,0c6h,038h,018h,018h,018h,03ch,000h  ;     8c
      00                    C
C8C8  70 00 38 18 18 18 3C  C        DB  070h,000h,038h,018h,018h,018h,03ch,000h  ;     8d
      00                    C
C8D0  63 1C 36 63 7F 63 63  C        DB  063h,01ch,036h,063h,07fh,063h,063h,000h  ;     8e
      00                    C
                            C                                                        ; FONT(S) 8f - 92
                            C
C8D8  18 18 00 3C 66 7E 66  C              DB  018h,018h,000h,03ch,066h,07eh,066h,000h  ;     8f
      00                    C
C8E0  0E 00 7E 30 3C 30 7E  C        DB  00eh,000h,07eh,030h,03ch,030h,07eh,000h  ;     90
      00                    C
C8E8  00 00 7F 0C 7F CC 7F  C        DB  000h,000h,07fh,00ch,07fh,0cch,07fh,000h  ;     91
      00                    C
C8F0  1F 36 66 7F 66 66 67  C        DB  01fh,036h,066h,07fh,066h,066h,067h,000h  ;     92
      00                    C
                            C
```

```
C8F8  3C 66 00 3C 66 66 3C  C       DB  03ch,066h,000h,03ch,066h,066h,03ch,000h  ;     93
      00                    C
C900  00 66 00 3C 66 66 3C  C       DB  000h,066h,000h,03ch,066h,066h,03ch,000h  ;     94
      00                    C
C908  00 70 00 3C 66 66 3C  C       DB  000h,070h,000h,03ch,066h,066h,03ch,000h  ;     95
      00                    C
C910  3C 66 00 66 66 66 3F  C       DB  03ch,066h,000h,066h,066h,066h,03fh,000h  ;     96
      00                    C
C918  00 70 00 66 66 66 3F  C       DB  000h,070h,000h,066h,066h,066h,03fh,000h  ;     97
      00                    C
                            C                                                        ; FONT(S) 98
                            C
C920  00 66 00 66 66 3E 06  C               DB  000h,066h,000h,066h,066h,03eh,006h,07ch  ;     98
      7C                    C
                            C
C928  C3 18 3C 66 66 3C 18  C       DB  0c3h,018h,03ch,066h,066h,03ch,018h,000h  ;     99
      00                    C
C930  66 00 66 66 66 66 3C  C       DB  066h,000h,066h,066h,066h,066h,03ch,000h  ;     9a
      00                    C
                            C                                                        ; FONT(S) 9b - 9f
                            C
                            C
                            C
C938  18 18 7E C0 C0 7E 18  C               DB  018h,018h,07eh,0c0h,0c0h,07eh,018h,018h  ;     9b
      18                    C
C940  1C 36 32 78 30 73 7E  C       DB  01ch,036h,032h,078h,030h,073h,07eh,000h  ;     9c
      00                    C
C948  66 66 3C 7E 18 7E 18  C       DB  066h,066h,03ch,07eh,018h,07eh,018h,018h  ;     9d
      18                    C
C950  F8 CC CC FA C6 CF C6  C       DB  0f8h,0cch,0cch,0fah,0c6h,0cfh,0c6h,0c7h  ;     9e
      C7                    C
C958  0E 1B 18 3C 18 18 D8  C       DB  00eh,01bh,018h,03ch,018h,018h,0d8h,070h  ;     9f
      70                    C
                            C
                            C
                            C
C960  0E 00 3C 06 3E 66 3F  C       DB  00eh,000h,03ch,006h,03eh,066h,03fh,000h  ;     a0
      00                    C
C968  1C 00 38 18 18 18 3C  C       DB  01ch,000h,038h,018h,018h,018h,03ch,000h  ;     a1
      00                    C
C970  00 0E 00 3C 66 66 3C  C       DB  000h,00eh,000h,03ch,066h,066h,03ch,000h  ;     a2
      00                    C
C978  00 0E 00 66 66 66 3F  C       DB  000h,00eh,000h,066h,066h,066h,03fh,000h  ;     a3
      00                    C
C980  00 7C 00 7C 66 66 66  C       DB  000h,07ch,000h,07ch,066h,066h,066h,000h  ;     a4
      00                    C
C988  7E 00 66 76 7E 6E 66  C       DB  07eh,000h,066h,076h,07eh,06eh,066h,000h  ;     a5
      00                    C
                            C                                                        ; FONT(S) a6 - af
                            C
                            C
C990  3C 6C 6C 3E 00 7E 00  C               DB  03ch,06ch,06ch,03eh,000h,07eh,000h,000h  ;     a6
      00                    C
C998  38 6C 6C 38 00 7C 00  C       DB  038h,06ch,06ch,038h,000h,07ch,000h,000h  ;     a7
      00                    C
```

```
C9A0  18 00 18 30 60 66 3C  C       DB  018h,000h,018h,030h,060h,066h,03ch,000h  ;     a8
      00                    C
                            C                                                       ; FONT(S) a9 - aa
                            C
C9A8  00 00 00 7E 60 60 00  C               DB  000h,000h,000h,07eh,060h,060h,000h,000h  ;     a9
      00                    C
C9B0  00 00 00 7E 06 06 00  C       DB  000h,000h,000h,07eh,006h,006h,000h,000h  ;     aa
      00                    C
                            C
C9B8  C3 C6 CC DE 33 66 CC  C       DB  0c3h,0c6h,0cch,0deh,033h,066h,0cch,00fh  ;     ab
      0F                    C
C9C0  C3 C6 CC DB 37 6F CF  C       DB  0c3h,0c6h,0cch,0dbh,037h,06fh,0cfh,003h  ;     ac
      03                    C
C9C8  18 18 00 18 18 18 18  C       DB  018h,018h,000h,018h,018h,018h,018h,000h  ;     ad
      00                    C
C9D0  00 33 66 CC 66 33 00  C       DB  000h,033h,066h,0cch,066h,033h,000h,000h  ;     ae
      00                    C
C9D8  00 CC 66 33 66 CC 00  C       DB  000h,0cch,066h,033h,066h,0cch,000h,000h  ;     af
      00                    C
                            C
                            C
C9E0  22 88 22 88 22 88 22  C       DB  022h,088h,022h,088h,022h,088h,022h,088h  ;     b0
      88                    C
C9E8  55 AA 55 AA 55 AA 55  C       DB  055h,0aah,055h,0aah,055h,0aah,055h,0aah  ;     b1
      AA                    C
C9F0  DB 77 DB EE DB 77 DB  C       DB  0dbh,077h,0dbh,0eeh,0dbh,077h,0dbh,0eeh  ;     b2
      EE                    C
C9F8  18 18 18 18 18 18 18  C       DB  018h,018h,018h,018h,018h,018h,018h,018h  ;     b3
      18                    C
CA00  18 18 18 18 F8 18 18  C       DB  018h,018h,018h,018h,0f8h,018h,018h,018h  ;     b4
      18                    C
CA08  18 18 F8 18 F8 18 18  C       DB  018h,018h,0f8h,018h,0f8h,018h,018h,018h  ;     b5
      18                    C
CA10  36 36 36 36 F6 36 36  C       DB  036h,036h,036h,036h,0f6h,036h,036h,036h  ;     b6
      36                    C
CA18  00 00 00 00 FE 36 36  C       DB  000h,000h,000h,000h,0feh,036h,036h,036h  ;     b7
      36                    C
CA20  00 00 F8 18 F8 18 18  C       DB  000h,000h,0f8h,018h,0f8h,018h,018h,018h  ;     b8
      18                    C
CA28  36 36 F6 06 F6 36 36  C       DB  036h,036h,0f6h,006h,0f6h,036h,036h,036h  ;     b9
      36                    C
CA30  36 36 36 36 36 36 36  C       DB  036h,036h,036h,036h,036h,036h,036h,036h  ;     ba
      36                    C
CA38  00 00 FE 06 F6 36 36  C       DB  000h,000h,0feh,006h,0f6h,036h,036h,036h  ;     bb
      36                    C
CA40  36 36 F6 06 FE 00 00  C       DB  036h,036h,0f6h,006h,0feh,000h,000h,000h  ;     bc
      00                    C
CA48  36 36 36 36 FE 00 00  C       DB  036h,036h,036h,036h,0feh,000h,000h,000h  ;     bd
      00                    C
CA50  18 18 F8 18 F8 00 00  C       DB  018h,018h,0f8h,018h,0f8h,000h,000h,000h  ;     be
      00                    C
CA58  00 00 00 00 F8 18 18  C       DB  000h,000h,000h,000h,0f8h,018h,018h,018h  ;     bf
      18                    C
CA60  18 18 18 18 1F 00 00  C       DB  018h,018h,018h,018h,01fh,000h,000h,000h  ;     c0
      00                    C
```

```
CA68  18 18 18 18 FF 00 00  C       DB  018h,018h,018h,018h,0ffh,000h,000h,000h  ;   c1
      00                    C
CA70  00 00 00 00 FF 18 18  C       DB  000h,000h,000h,000h,0ffh,018h,018h,018h  ;   c2
      18                    C
CA78  18 18 18 18 1F 18 18  C       DB  018h,018h,018h,018h,01fh,018h,018h,018h  ;   c3
      18                    C
CA80  00 00 00 00 FF 00 00  C       DB  000h,000h,000h,000h,0ffh,000h,000h,000h  ;   c4
      00                    C
CA88  18 18 18 18 FF 18 18  C       DB  018h,018h,018h,018h,0ffh,018h,018h,018h  ;   c5
      18                    C
CA90  18 18 1F 18 1F 18 18  C       DB  018h,018h,01fh,018h,01fh,018h,018h,018h  ;   c6
      18                    C
CA98  36 36 36 36 37 36 36  C       DB  036h,036h,036h,036h,037h,036h,036h,036h  ;   c7
      36                    C
CAA0  36 36 37 30 3F 00 00  C       DB  036h,036h,037h,030h,03fh,000h,000h,000h  ;   c8
      00                    C
CAA8  00 00 3F 30 37 36 36  C       DB  000h,000h,03fh,030h,037h,036h,036h,036h  ;   c9
      36                    C
CAB0  36 36 F7 00 FF 00 00  C       DB  036h,036h,0f7h,000h,0ffh,000h,000h,000h  ;   ca
      00                    C
CAB8  00 00 FF 00 F7 36 36  C       DB  000h,000h,0ffh,000h,0f7h,036h,036h,036h  ;   cb
      36                    C
CAC0  36 36 37 30 37 36 36  C       DB  036h,036h,037h,030h,037h,036h,036h,036h  ;   cc
      36                    C
CAC8  00 00 FF 00 FF 00 00  C       DB  000h,000h,0ffh,000h,0ffh,000h,000h,000h  ;   cd
      00                    C
CAD0  36 36 F7 00 F7 36 36  C       DB  036h,036h,0f7h,000h,0f7h,036h,036h,036h  ;   ce
      36                    C
CAD8  18 18 FF 00 FF 00 00  C       DB  018h,018h,0ffh,000h,0ffh,000h,000h,000h  ;   cf
      00                    C
CAE0  36 36 36 36 FF 00 00  C       DB  036h,036h,036h,036h,0ffh,000h,000h,000h  ;   d0
      00                    C
CAE8  00 00 FF 00 FF 18 18  C       DB  000h,000h,0ffh,000h,0ffh,018h,018h,018h  ;   d1
      18                    C
CAF0  00 00 00 00 FF 36 36  C       DB  000h,000h,000h,000h,0ffh,036h,036h,036h  ;   d2
      36                    C
CAF8  36 36 36 36 3F 00 00  C       DB  036h,036h,036h,036h,03fh,000h,000h,000h  ;   d3
      00                    C
CB00  18 18 1F 18 1F 00 00  C       DB  018h,018h,01fh,018h,01fh,000h,000h,000h  ;   d4
      00                    C
CB08  00 00 1F 18 1F 18 18  C       DB  000h,000h,01fh,018h,01fh,018h,018h,018h  ;   d5
      18                    C
CB10  00 00 00 00 3F 36 36  C       DB  000h,000h,000h,000h,03fh,036h,036h,036h  ;   d6
      36                    C
CB18  36 36 36 36 FF 36 36  C       DB  036h,036h,036h,036h,0ffh,036h,036h,036h  ;   d7
      36                    C
CB20  18 18 FF 18 FF 18 18  C       DB  018h,018h,0ffh,018h,0ffh,018h,018h,018h  ;   d8
      18                    C
CB28  18 18 18 18 F8 00 00  C       DB  018h,018h,018h,018h,0f8h,000h,000h,000h  ;   d9
      00                    C
CB30  00 00 00 00 1F 18 18  C       DB  000h,000h,000h,000h,01fh,018h,018h,018h  ;   da
      18                    C
CB38  FF FF FF FF FF FF FF  C       DB  0ffh,0ffh,0ffh,0ffh,0ffh,0ffh,0ffh,0ffh  ;   db
      FF                    C
CB40  00 00 00 00 FF FF FF  C       DB  000h,000h,000h,000h,0ffh,0ffh,0ffh,0ffh  ;   dc
```

```
      FF                        C
CB48  F0 F0 F0 F0 F0 F0 F0      C       DB  0f0h,0f0h,0f0h,0f0h,0f0h,0f0h,0f0h,0f0h  ;   dd
      F0                        C
CB50  0F 0F 0F 0F 0F 0F 0F      C       DB  00fh,00fh,00fh,00fh,00fh,00fh,00fh,00fh  ;   de
      0F                        C
CB58  FF FF FF FF 00 00 00      C       DB  0ffh,0ffh,0ffh,0ffh,000h,000h,000h,000h  ;   df
      00                        C
CB60  00 00 3B 6E 64 6E 3B      C       DB  000h,000h,03bh,06eh,064h,06eh,03bh,000h  ;   e0
      00                        C
CB68  00 3C 66 7C 66 7C 60      C       DB  000h,03ch,066h,07ch,066h,07ch,060h,060h  ;   e1
      60                        C
CB70  00 7E 66 60 60 60 60      C       DB  000h,07eh,066h,060h,060h,060h,060h,000h  ;   e2
      00                        C
CB78  00 7F 36 36 36 36 36      C       DB  000h,07fh,036h,036h,036h,036h,036h,000h  ;   e3
      00                        C
CB80  7E 66 30 18 30 66 7E      C       DB  07eh,066h,030h,018h,030h,066h,07eh,000h  ;   e4
      00                        C
CB88  00 00 3F 6C 6C 6C 38      C       DB  000h,000h,03fh,06ch,06ch,06ch,038h,000h  ;   e5
      00                        C
CB90  00 33 33 33 33 3E 30      C       DB  000h,033h,033h,033h,033h,03eh,030h,060h  ;   e6
      60                        C
CB98  00 3B 6E 0C 0C 0C 0C      C       DB  000h,03bh,06eh,00ch,00ch,00ch,00ch,000h  ;   e7
      00                        C
CBA0  7E 18 3C 66 66 3C 18      C       DB  07eh,018h,03ch,066h,066h,03ch,018h,07eh  ;   e8
      7E                        C
CBA8  1C 36 63 7F 63 36 1C      C       DB  01ch,036h,063h,07fh,063h,036h,01ch,000h  ;   e9
      00                        C
CBB0  1C 36 63 63 36 36 77      C       DB  01ch,036h,063h,063h,036h,036h,077h,000h  ;   ea
      00                        C
CBB8  0E 18 0C 3E 66 66 3C      C       DB  00eh,018h,00ch,03eh,066h,066h,03ch,000h  ;   eb
      00                        C
CBC0  00 00 7E DB DB 7E 00      C       DB  000h,000h,07eh,0dbh,0dbh,07eh,000h,000h  ;   ec
      00                        C
CBC8  06 0C 7E DB DB 7E 60      C       DB  006h,00ch,07eh,0dbh,0dbh,07eh,060h,0c0h  ;   ed
      C0                        C
CBD0  1C 60 C0 FC C0 60 1C      C       DB  01ch,060h,0c0h,0fch,0c0h,060h,01ch,000h  ;   ee
      00                        C
CBD8  3C 66 66 66 66 66 66      C       DB  03ch,066h,066h,066h,066h,066h,066h,000h  ;   ef
      00                        C
CBE0  00 7E 00 7E 00 7E 00      C       DB  000h,07eh,000h,07eh,000h,07eh,000h,000h  ;   f0
      00                        C
CBE8  18 18 7E 18 18 00 7E      C       DB  018h,018h,07eh,018h,018h,000h,07eh,000h  ;   f1
      00                        C
CBF0  30 18 0C 18 30 00 7E      C       DB  030h,018h,00ch,018h,030h,000h,07eh,000h  ;   f2
      00                        C
CBF8  0C 18 30 18 0C 00 7E      C       DB  00ch,018h,030h,018h,00ch,000h,07eh,000h  ;   f3
      00                        C
CC00  0E 1B 1B 18 18 18 18      C       DB  00eh,01bh,01bh,018h,018h,018h,018h,018h  ;   f4
      18                        C
CC08  18 18 18 18 18 D8 D8      C       DB  018h,018h,018h,018h,018h,0d8h,0d8h,070h  ;   f5
      70                        C
CC10  18 18 00 7E 00 18 18      C       DB  018h,018h,000h,07eh,000h,018h,018h,000h  ;   f6
      00                        C
CC18  00 76 DC 00 76 DC 00      C       DB  000h,076h,0dch,000h,076h,0dch,000h,000h  ;   f7
      00                        C
```

```
CC20  38 6C 6C 38 00 00 00  C       DB  038h,06ch,06ch,038h,000h,000h,000h,000h  ;     f8
      00                    C
CC28  00 00 00 18 18 00 00  C       DB  000h,000h,000h,018h,018h,000h,000h,000h  ;     f9
      00                    C
CC30  00 00 00 00 18 00 00  C       DB  000h,000h,000h,000h,018h,000h,000h,000h  ;     fa
      00                    C
CC38  0F 0C 0C 0C EC 6C 3C  C       DB  00fh,00ch,00ch,00ch,0ech,06ch,03ch,01ch  ;     fb
      1C                    C
CC40  78 6C 6C 6C 6C 00 00  C       DB  078h,06ch,06ch,06ch,06ch,000h,000h,000h  ;     fc
      00                    C
CC48  70 18 30 60 78 00 00  C       DB  070h,018h,030h,060h,078h,000h,000h,000h  ;     fd
      00                    C
CC50  00 00 3C 3C 3C 3C 00  C       DB  000h,000h,03ch,03ch,03ch,03ch,000h,000h  ;     fe
      00                    C
CC58  00 00 00 00 00 00 00  C       DB  000h,000h,000h,000h,000h,000h,000h,000h  ;     ff
      00                    C
                            C   ;  End of font matrix
                            C
                            C   fonthi8 endp


CC60                            code ends

                            C   include kbdata.asm
                            C
                            C   ;=========================================================================
                            C   ;    Filename:kb.data: USA-ASCII
                            C   ;
                            C   ;    This module includes the keyboard scan code translation data
                            C   ;       for different keyboards.
                            C   ;
                            C   ;=========================================================================
                            C
CC60                        C   code segment  public 'ROM'
                            C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                            C
CC60                        C   kb_data1      proc
                            C
                            C   ;-------------------------------------------------------------------
                            C   ;    special_cases
                            C   ;-------------------------------------------------------------------
                            C
= 00C0                      C   kbinsequ      0C0h     ; kb_insert_lock  (min_case)
= 00C1                      C   kbcapequ      0C1h     ; kb_caps_lock
= 00C2                      C   kbnumequ      0C2h     ; kb_num_lock
= 00C3                      C   kbscrequ      0C3h     ; kb_scroll_lock
= 00C4                      C   kbaltequ      0C4h     ; kb_alt_lock
= 00C5                      C   kbctlequ      0C5h     ; kb_control_lock
= 00C6                      C   kblshequ      0C6h     ; kb_l_shift_lock
= 00C7                      C   kbrshequ      0C7h     ; kb_r_shift_lock
                            C
= 00C8                      C   kbresequ      0C8h     ; kb_reset                (mid_case)
= 00C9                      C   kbbrkequ      0C9h     ; kb_break
= 00CA                      C   pauseequ      0CAh     ; kb_pause
= 00CB                      C   kbprtequ      0CBh     ; kb_print_screen
```

```
= 00CC                  C  kbnulequ     0CCh     ; kb_null
= 00CD                  C  kNONEequ     0CDh     ; kb_none
                        C
= 00CE                  C  kdec9equ     0CEh     ; kb_alt_dec_9
= 00CF                  C  kdec8equ     0CFh     ; kb_alt_dec_8
= 00D0                  C  kdec7equ     0D0h     ; kb_alt_dec_7
= 00D1                  C  kdec6equ     0D1h     ; kb_alt_dec_6
= 00D2                  C  kdec5equ     0D2h     ; kb_alt_dec_5
= 00D3                  C  kdec4equ     0D3h     ; kb_alt_dec_4
= 00D4                  C  kdec3equ     0D4h     ; kb_alt_dec_3
= 00D5                  C  kdec2equ     0D5h     ; kb_alt_dec_2
= 00D6                  C  kdec1equ     0D6h     ; kb_alt_dec_1
= 00D7                  C  kdec0equ     0D7h     ; kb_alt_dec_0
                        C
= 00D8                  C  kdbl0equ     0D8h     ; kb_double_zero  (max_case)
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    7 CapLk  Bytes
                        C  ;--------------------------------------------------------------------
                        C
CC60                    C  kb_cap_flags  label    byte
                        C
CC60  00                C   db  00000000b; scancode 00 (00h) - 07 (07h)      ESC         to '6'
CC61  00                C   db  00000000b; scancode 08 (08h) - 15 (0Fh)      '7'         to HT
CC62  FF                C   db  11111111b; scancode 16 (10h) - 23 (17h)      'q'         to 'i'
CC63  C3                C   db  11000011b; scancode 24 (18h) - 31 (1Fh)      'o' & 'p'  to 'a' & 's'
CC64  FE                C   db  11111110b; scancode 32 (20h) - 39 (27h)      'd'         to 'l' & ';'
CC65  0F                C   db  00001111b; scancode 40 (28h) - 47 (2Fh)      ':' to '´' to 'z' to 'v'
CC66  E0                C   db  11100000b; scancode 48 (30h) - 55 (37h)      'b' to 'm' to ',' to '*'
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;       Alphabetic (Migratory)              |Olivetti|Other |
                        C  ;                                           | KB     | KBs  |
                        C  ;--------------------------------------------------------------------
                        C
CC67                    C  kb_data_table label    byte
                        C
CC67  1B1B              C   dw  (1Bh) * 100h + (1Bh)       ;  01 01hESC      ESC      (BASE)
CC69  1B1B              C   dw  (1Bh) * 100h + (1Bh)       ;                 ESC      ESC      (SHIFT)
CC6B  1B1B              C   dw  (1Bh) * 100h + (1Bh)       ;                 ESC      ESC      (CTL)
CC6D  CDCD              C   dw  kNONE * 100h + kNONE       ;                 None     None     (ALT)
CC6F  3131              C   dw  (31h) * 100h + (31h)       ;  02 02h1        1
CC71  2121              C   dw  (21h) * 100h + (21h)       ;                 !        !
CC73  CDCD              C   dw  kNONE * 100h + kNONE       ;                 None     None
CC75  7800              C   dw           7800h             ;                 X120
CC77  3232              C   dw  (32h) * 100h + (32h)       ;  03 03h2        2
CC79  2240              C   dw  (22h) * 100h + (40h)       ;                 "        @
CC7B  CDCC              C   dw  kNONE * 100h + kbnul       ;                 None     NUL=X03(^@)
CC7D  7900              C   dw           7900h             ;                 X121
CC7F  3333              C   dw  (33h) * 100h + (33h)       ;  04 04h3        3
CC81  2323              C   dw  (23h) * 100h + (23h)       ;                 #        #
CC83  CDCD              C   dw  kNONE * 100h + kNONE       ;                 None     None
CC85  7A00              C   dw           7A00h             ;                 X122
CC87  3434              C   dw  (34h) * 100h + (34h)       ;  05 05h4        4
CC89  2424              C   dw  (24h) * 100h + (24h)       ;                 $        $
```

```
CC8B CDCD          C   dw  kNONE * 100h + kNONE      ;                 None      None
CC8D 7B00          C   dw          7B00h           ;                 X123
CC8F 3535          C   dw  (35h) * 100h + (35h)     ;  06 06h5        5
CC91 2525          C   dw  (25h) * 100h + (25h)     ;                 %         %
CC93 CDCD          C   dw  kNONE * 100h + kNONE      ;                 None      None
CC95 7C00          C   dw          7C00h           ;                 X124
CC97 3636          C   dw  (36h) * 100h + (36h)     ;  07 07h6        6
CC99 265E          C   dw  (26h) * 100h + (5Eh)     ;                 &         ^
CC9B CD1E          C   dw  kNONE * 100h + (1Eh)      ;                 None      RS (^^)
CC9D 7D00          C   dw          7D00h           ;                 X125
CC9F 3737          C   dw  (37h) * 100h + (37h)     ;  08 08h7        7
CCA1 2726          C   dw  (27h) * 100h + (26h)     ;                 '         &
CCA3 CDCD          C   dw  kNONE * 100h + kNONE      ;                 None      None
CCA5 7E00          C   dw          7E00h           ;                 X126
CCA7 3838          C   dw  (38h) * 100h + (38h)     ;  09 09h8        8
CCA9 282A          C   dw  (28h) * 100h + (2Ah)     ;                 (         *
CCAB CDCD          C   dw  kNONE * 100h + kNONE      ;                 None      None
CCAD 7F00          C   dw          7F00h           ;                 X127
CCAF 3939          C   dw  (39h) * 100h + (39h)     ;  10 0Ah9        9
CCB1 2928          C   dw  (29h) * 100h + (28h)     ;                 )         (
CCB3 CDCD          C   dw  kNONE * 100h + kNONE      ;                 None      None
CCB5 8000          C   dw          8000h           ;                 X128
CCB7 3030          C   dw  (30h) * 100h + (30h)     ;  11 0Bh0        0
CCB9 5F29          C   dw  (5Fh) * 100h + (29h)     ;                 _         )
CCBB 1FCD          C   dw  (1Fh) * 100h + kNONE      ;                 US (^_)   None
CCBD 8100          C   dw          8100h           ;                 X129
CCBF 2D2D          C   dw  (2Dh) * 100h + (2Dh)     ;  12 0Ch-        -
CCC1 3D5F          C   dw  (3Dh) * 100h + (5Fh)     ;                 =         _
CCC3 CD1F          C   dw  kNONE * 100h + (1Fh)      ;                 None      US (^_)
CCC5 8200          C   dw          8200h           ;                 X130
CCC7 5E3D          C   dw  (5Eh) * 100h + (3Dh)     ;  13 0Dh^        =
CCC9 7E2B          C   dw  (7Eh) * 100h + (2Bh)     ;                 ~         +
CCCB 1ECD          C   dw  (1Eh) * 100h + kNONE      ;                 RS (^^)   None
CCCD 8300          C   dw          8300h           ;                 X131
CCCF 0808          C   dw  (08h) * 100h + (08h)     ;  14 0EhBS       BS
CCD1 0808          C   dw  (08h) * 100h + (08h)     ;                 BS        BS
CCD3 7F7F          C   dw  (7Fh) * 100h + (7Fh)     ;                 DEL       DEL
CCD5 CDCD          C   dw  kNONE * 100h + kNONE      ;                 None      None
CCD7 0909          C   dw  (09h) * 100h + (09h)     ;  15 0FhHT       HT
CCD9 0F00          C   dw          0F00h           ;                 RHT=X15
CCDB CDCD          C   dw  kNONE * 100h + kNONE      ;                 None      None
CCDD CDCD          C   dw  kNONE * 100h + kNONE      ;                 None      None
CCDF 7171          C   dw  (71h) * 100h + (71h)     ;  16 10hq        q
CCE1 5151          C   dw  (51h) * 100h + (51h)     ;                 Q         Q
CCE3 1111          C   dw  (11h) * 100h + (11h)     ;                 DC1(^Q)   DC1(^Q)
CCE5 1000          C   dw          1000h           ;                 X16
CCE7 7777          C   dw  (77h) * 100h + (77h)     ;  17 11hw        w
CCE9 5757          C   dw  (57h) * 100h + (57h)     ;                 W         W
CCEB 1717          C   dw  (17h) * 100h + (17h)     ;                 ETB(^W)   ETB(^W)
CCED 1100          C   dw          1100h           ;                 X17
CCEF 6565          C   dw  (65h) * 100h + (65h)     ;  18 12he        e
CCF1 4545          C   dw  (45h) * 100h + (45h)     ;                 E         E
CCF3 0505          C   dw  (05h) * 100h + (05h)     ;                 ENQ(^E)   ENQ(^E)
CCF5 1200          C   dw          1200h           ;                 X18
CCF7 7272          C   dw  (72h) * 100h + (72h)     ;  19 13hr        r
```

```
CCF9  5252              C   dw  (52h) * 100h + (52h)    ;                   R       R
CCFB  1212              C   dw  (12h) * 100h + (12h)    ;                   DC2(^R) DC2(^R)
CCFD  1300              C   dw          1300h          ;                   X19
CCFF  7474              C   dw  (74h) * 100h + (74h)    ;  20 14ht          t
CD01  5454              C   dw  (54h) * 100h + (54h)    ;                   T       T
CD03  1414              C   dw  (14h) * 100h + (14h)    ;                   DC4(^T) DC4(^T)
CD05  1400              C   dw          1400h          ;                   X20
CD07  7979              C   dw  (79h) * 100h + (79h)    ;  21 15hy          y
CD09  5959              C   dw  (59h) * 100h + (59h)    ;                   Y       Y
CD0B  1919              C   dw  (19h) * 100h + (19h)    ;                   EM (^Y) EM (^Y)
CD0D  1500              C   dw          1500h          ;                   X21
CD0F  7575              C   dw  (75h) * 100h + (75h)    ;  22 16hu          u
CD11  5555              C   dw  (55h) * 100h + (55h)    ;                   U       U
CD13  1515              C   dw  (15h) * 100h + (15h)    ;                   NAK(^U) NAK(^U)
CD15  1600              C   dw          1600h          ;                   X22
CD17  6969              C   dw  (69h) * 100h + (69h)    ;  23 17hi          i
CD19  4949              C   dw  (49h) * 100h + (49h)    ;                   I       I
CD1B  0909              C   dw  (09h) * 100h + (09h)    ;                   HT (^I) HT (^I)
CD1D  1700              C   dw          1700h          ;                   X23
CD1F  6F6F              C   dw  (6Fh) * 100h + (6Fh)    ;  24 18ho          o
CD21  4F4F              C   dw  (4Fh) * 100h + (4Fh)    ;                   O       O
CD23  0F0F              C   dw  (0Fh) * 100h + (0Fh)    ;                   SI (^O) SI (^O)
CD25  1800              C   dw          1800h          ;                   X24
CD27  7070              C   dw  (70h) * 100h + (70h)    ;  25 19hp          p
CD29  5050              C   dw  (50h) * 100h + (50h)    ;                   P       P
CD2B  1010              C   dw  (10h) * 100h + (10h)    ;                   DLE(^P) DLE(^P)
CD2D  1900              C   dw          1900h          ;                   X25
CD2F  405B              C   dw  (40h) * 100h + (5Bh)    ;  26 1Ah@          [
CD31  607B              C   dw  (60h) * 100h + (7Bh)    ;                   `       {
CD33  CC1B              C   dw  kbnul * 100h + (1Bh)    ;                NUL=X03(^@) ESC(^[)
CD35  CDCD              C   dw  kNONE * 100h + kNONE    ;                   None    None
CD37  5B5D              C   dw  (5Bh) * 100h + (5Dh)    ;  27 1Bh[          ]
CD39  7B7D              C   dw  (7Bh) * 100h + (7Dh)    ;                   {       }
CD3B  1B1D              C   dw  (1Bh) * 100h + (1Dh)    ;                   ESC(^[) GS (^])
CD3D  CDCD              C   dw  kNONE * 100h + kNONE    ;                   None    None
CD3F  0D0D              C   dw  (0Dh) * 100h + (0Dh)    ;  28 1ChCR         CR
CD41  0D0D              C   dw  (0Dh) * 100h + (0Dh)    ;                   CR      CR
CD43  0A0A              C   dw  (0Ah) * 100h + (0Ah)    ;                   LF      LF
CD45  CDCD              C   dw  kNONE * 100h + kNONE    ;                   None    None
CD47  C5C5              C   dw  kbctl * 100h + kbctl    ;  29 1DhCtrl       Ctrl
CD49  C5C5              C   dw  kbctl * 100h + kbctl    ;                   Ctrl    Ctrl
CD4B  C5C5              C   dw  kbctl * 100h + kbctl    ;                   Ctrl    Ctrl
CD4D  C5C5              C   dw  kbctl * 100h + kbctl    ;                   Ctrl    Ctrl
CD4F  6161              C   dw  (61h) * 100h + (61h)    ;  30 1Eha          a
CD51  4141              C   dw  (41h) * 100h + (41h)    ;                   A       A
CD53  0101              C   dw  (01h) * 100h + (01h)    ;                   SOH(^A) SOH(^A)
CD55  1E00              C   dw          1E00h          ;                   X30
CD57  7373              C   dw  (73h) * 100h + (73h)    ;  31 1Fhs          s
CD59  5353              C   dw  (53h) * 100h + (53h)    ;                   S       S
CD5B  1313              C   dw  (13h) * 100h + (13h)    ;                   DC3(^S) DC3(^S)
CD5D  1F00              C   dw          1F00h          ;                   X31
CD5F  6464              C   dw  (64h) * 100h + (64h)    ;  32 20hd          d
CD61  4444              C   dw  (44h) * 100h + (44h)    ;                   D       D
CD63  0404              C   dw  (04h) * 100h + (04h)    ;                   EOT(^D) EOT(^D)
CD65  2000              C   dw          2000h          ;                   X32
```

```
CD67  6666              C   dw  (66h) * 100h + (66h)     ;  33 21hf       f
CD69  4646              C   dw  (46h) * 100h + (46h)     ;                F         F
CD6B  0606              C   dw  (06h) * 100h + (06h)     ;                ACK(^F)   ACK(^F)
CD6D  2100              C   dw          2100h           ;                X33
CD6F  6767              C   dw  (67h) * 100h + (67h)     ;  34 22hg       g
CD71  4747              C   dw  (47h) * 100h + (47h)     ;                G         G
CD73  0707              C   dw  (07h) * 100h + (07h)     ;                BEL(^G)   BEL(^G)
CD75  2200              C   dw          2200h           ;                X34
CD77  6868              C   dw  (68h) * 100h + (68h)     ;  35 23hh       h
CD79  4848              C   dw  (48h) * 100h + (48h)     ;                H         H
CD7B  0808              C   dw  (08h) * 100h + (08h)     ;                BS (^H)   BS (^H)
CD7D  2300              C   dw          2300h           ;                X35
CD7F  6A6A              C   dw  (6Ah) * 100h + (6Ah)     ;  36 24hj       j
CD81  4A4A              C   dw  (4Ah) * 100h + (4Ah)     ;                J         J
CD83  0A0A              C   dw  (0Ah) * 100h + (0Ah)     ;                LF (^J)   LF (^J)
CD85  2400              C   dw          2400h           ;                X36
CD87  6B6B              C   dw  (6Bh) * 100h + (6Bh)     ;  37 25hk       k
CD89  4B4B              C   dw  (4Bh) * 100h + (4Bh)     ;                K         K
CD8B  0B0B              C   dw  (0Bh) * 100h + (0Bh)     ;                VT (^K)   VT (^K)
CD8D  2500              C   dw          2500h           ;                X37
CD8F  6C6C              C   dw  (6Ch) * 100h + (6Ch)     ;  38 26hl       l
CD91  4C4C              C   dw  (4Ch) * 100h + (4Ch)     ;                L         L
CD93  0C0C              C   dw  (0Ch) * 100h + (0Ch)     ;                FF (^L)   FF (^L)
CD95  2600              C   dw          2600h           ;                X38
CD97  3B3B              C   dw  (3Bh) * 100h + (3Bh)     ;  39 27h;       ;
CD99  2B3A              C   dw  (2Bh) * 100h + (3Ah)     ;                +         :
CD9B  CDCD              C   dw  kNONE * 100h + kNONE     ;                None      None
CD9D  CDCD              C   dw  kNONE * 100h + kNONE     ;                None      None
CD9F  3A27              C   dw  (3Ah) * 100h + (27h)     ;  40 28h'       '
CDA1  2A22              C   dw  (2Ah) * 100h + (22h)     ;                *         "
CDA3  CDCD              C   dw  kNONE * 100h + kNONE     ;                None      None
CDA5  CDCD              C   dw  kNONE * 100h + kNONE     ;                None      None
CDA7  5D60              C   dw  (5Dh) * 100h + (60h)     ;  41 29h]       `
CDA9  7D7E              C   dw  (7Dh) * 100h + (7Eh)     ;                }         ~
CDAB  1DCD              C   dw  (1Dh) * 100h + kNONE     ;                GS (^])   None
CDAD  CDCD              C   dw  kNONE * 100h + kNONE     ;                None      None
CDAF  C6C6              C   dw  kblsh * 100h + kblsh     ;  42 2AhLShft   LShft
CDB1  C6C6              C   dw  kblsh * 100h + kblsh     ;                LShft     LShft
CDB3  C6C6              C   dw  kblsh * 100h + kblsh     ;                LShft     LShft
CDB5  C6C6              C   dw  kblsh * 100h + kblsh     ;                LShft     LShft
CDB7  5C5C              C   dw  (5Ch) * 100h + (5Ch)     ;  43 2Bh
CDB9  7C7C              C   dw  (7Ch) * 100h + (7Ch)     ;                |         |
CDBB  1C1C              C   dw  (1Ch) * 100h + (1Ch)     ;                FS (^)    FS (^)
CDBD  CDCD              C   dw  kNONE * 100h + kNONE     ;                None      None
CDBF  7A7A              C   dw  (7Ah) * 100h + (7Ah)     ;  44 2Chz       z
CDC1  5A5A              C   dw  (5Ah) * 100h + (5Ah)     ;                Z         Z
CDC3  1A1A              C   dw  (1Ah) * 100h + (1Ah)     ;                SUB(^Z)   SUB(^Z)
CDC5  2C00              C   dw          2C00h           ;                X44
CDC7  7878              C   dw  (78h) * 100h + (78h)     ;  45 2Dhx       x
CDC9  5858              C   dw  (58h) * 100h + (58h)     ;                X         X
CDCB  1818              C   dw  (18h) * 100h + (18h)     ;                CAN(^X)   CAN(^X)
CDCD  2D00              C   dw          2D00h           ;                X45
CDCF  6363              C   dw  (63h) * 100h + (63h)     ;  46 2Ehc       c
CDD1  4343              C   dw  (43h) * 100h + (43h)     ;                C         C
CDD3  0303              C   dw  (03h) * 100h + (03h)     ;                ETX(^C)   ETX(^C)
```

```
CDD5  2E00            C   dw          2E00h           ;                 X46
CDD7  7676            C   dw  (76h) * 100h + (76h)   ;  47 2Fhv      v
CDD9  5656            C   dw  (56h) * 100h + (56h)   ;                 V         V
CDDB  1616            C   dw  (16h) * 100h + (16h)   ;                 SYN(^V)   SYN(^V)
CDDD  2F00            C   dw          2F00h           ;                 X47
CDDF  6262            C   dw  (62h) * 100h + (62h)   ;  48 30hb      b
CDE1  4242            C   dw  (42h) * 100h + (42h)   ;                 B         B
CDE3  0202            C   dw  (02h) * 100h + (02h)   ;                 STX(^B)   STX(^B)
CDE5  3000            C   dw          3000h           ;                 X48
CDE7  6E6E            C   dw  (6Eh) * 100h + (6Eh)   ;  49 31hn      n
CDE9  4E4E            C   dw  (4Eh) * 100h + (4Eh)   ;                 N         N
CDEB  0E0E            C   dw  (0Eh) * 100h + (0Eh)   ;                 SO (^N)   SO (^N)
CDED  3100            C   dw          3100h           ;                 X49
CDEF  6D6D            C   dw  (6Dh) * 100h + (6Dh)   ;  50 32hm      m
CDF1  4D4D            C   dw  (4Dh) * 100h + (4Dh)   ;                 M         M
CDF3  0D0D            C   dw  (0Dh) * 100h + (0Dh)   ;                 CR (^M)   CR (^M)
CDF5  3200            C   dw          3200h           ;                 X50
CDF7  2C2C            C   dw  (2Ch) * 100h + (2Ch)   ;  51 33h,      ,
CDF9  3C3C            C   dw  (3Ch) * 100h + (3Ch)   ;                 <         <
CDFB  CDCD            C   dw  kNONE * 100h + kNONE    ;                 None      None
CDFD  CDCD            C   dw  kNONE * 100h + kNONE    ;                 None      None
CDFF  2E2E            C   dw  (2Eh) * 100h + (2Eh)   ;  52 34h.      .
CE01  3E3E            C   dw  (3Eh) * 100h + (3Eh)   ;                 >         >
CE03  CDCD            C   dw  kNONE * 100h + kNONE    ;                 None      None
CE05  CDCD            C   dw  kNONE * 100h + kNONE    ;                 None      None
CE07  2F2F            C   dw  (2Fh) * 100h + (2Fh)   ;  53 35h/      /
CE09  3F3F            C   dw  (3Fh) * 100h + (3Fh)   ;                 ?         ?
CE0B  CDCD            C   dw  kNONE * 100h + kNONE    ;                 None      None
CE0D  CDCD            C   dw  kNONE * 100h + kNONE    ;                 None      None
                      C
                      C ;-------------------------------------------------------------------
                      C ;       Alphabetic (Non-Migratory)          |Olivetti|Other |
                      C ;                                           |  KB    | KBs  |
                      C ;-----------------------------------------------------
                      C
CE0F  C7C7            C   dw  kbrsh * 100h + kbrsh    ;  54 36hRShft    RShft
CE11  C7C7            C   dw  kbrsh * 100h + kbrsh    ;                 RShft     RShft
CE13  C7C7            C   dw  kbrsh * 100h + kbrsh    ;                 RShft     RShft
CE15  C7C7            C   dw  kbrsh * 100h + kbrsh    ;                 RShft     RShft
CE17  2A2A            C   dw  (2Ah) * 100h + (2Ah)   ;  55 37h*      *
CE19  CBCB            C   dw  kbprt * 100h + kbprt    ;                 PrtSc     PrtSc
CE1B  7200            C   dw          7200h           ;                 X114
CE1D  CDCD            C   dw  kNONE * 100h + kNONE    ;                 None      None
CE1F  C4C4            C   dw  kbalt * 100h + kbalt    ;  56 38hALT      ALT
CE21  C4C4            C   dw  kbalt * 100h + kbalt    ;                 ALT       ALT
CE23  C4C4            C   dw  kbalt * 100h + kbalt    ;                 ALT       ALT
CE25  C4C4            C   dw  kbalt * 100h + kbalt    ;                 ALT       ALT
CE27  2020            C   dw  (20h) * 100h + (20h)   ;  57 39hSP      SP
CE29  2020            C   dw  (20h) * 100h + (20h)   ;                 SP        SP
CE2B  2020            C   dw  (20h) * 100h + (20h)   ;                 SP        SP
CE2D  2020            C   dw  (20h) * 100h + (20h)   ;                 SP        SP
CE2F  C1C1            C   dw  kbcap * 100h + kbcap    ;  58 3AhCapLk    CapLk
CE31  C1C1            C   dw  kbcap * 100h + kbcap    ;                 CapLk     CapLk
CE33  C1C1            C   dw  kbcap * 100h + kbcap    ;                 CapLk     CapLk
CE35  C1C1            C   dw  kbcap * 100h + kbcap    ;                 CapLk     CapLk
```

```
CE37  3B00            C   dw          3B00h           ;  59 3BhF01=X59
CE39  5400            C   dw          5400h           ;                F11=X84
CE3B  5E00            C   dw          5E00h           ;                F21=X94
CE3D  6800            C   dw          6800h           ;                F31=X104
CE3F  3C00            C   dw          3C00h           ;  60 3ChF02=X60
CE41  5500            C   dw          5500h           ;                F12=X85
CE43  5F00            C   dw          5F00h           ;                F22=X95
CE45  6900            C   dw          6900h           ;                F32=X105
CE47  3D00            C   dw          3D00h           ;  61 3DhF03=X61
CE49  5600            C   dw          5600h           ;                F13=X86
CE4B  6000            C   dw          6000h           ;                F23=X96
CE4D  6A00            C   dw          6A00h           ;                F33=X106
CE4F  3E00            C   dw          3E00h           ;  62 3EhF04=X62
CE51  5700            C   dw          5700h           ;                F14=X87
CE53  6100            C   dw          6100h           ;                F24=X97
CE55  6B00            C   dw          6B00h           ;                F34=X107
CE57  3F00            C   dw          3F00h           ;  63 3FhF05=X63
CE59  5800            C   dw          5800h           ;                F15=X88
CE5B  6200            C   dw          6200h           ;                F25=X98
CE5D  6C00            C   dw          6C00h           ;                F35=X108
CE5F  4000            C   dw          4000h           ;  64 40hF06=X64
CE61  5900            C   dw          5900h           ;                F16=X89
CE63  6300            C   dw          6300h           ;                F26=X99
CE65  6D00            C   dw          6D00h           ;                F36=X109
CE67  4100            C   dw          4100h           ;  65 41hF07=X65
CE69  5A00            C   dw          5A00h           ;                F17=X90
CE6B  6400            C   dw          6400h           ;                F27=X100
CE6D  6E00            C   dw          6E00h           ;                F37=X110
CE6F  4200            C   dw          4200h           ;  66 42hF08=X66
CE71  5B00            C   dw          5B00h           ;                F18=X91
CE73  6500            C   dw          6500h           ;                F28=X101
CE75  6F00            C   dw          6F00h           ;                F38=X111
CE77  4300            C   dw          4300h           ;  67 43hF09=X67
CE79  5C00            C   dw          5C00h           ;                F19=X92
CE7B  6600            C   dw          6600h           ;                F29=X102
CE7D  7000            C   dw          7000h           ;                F39=X112
CE7F  4400            C   dw          4400h           ;  68 44hF10=X68
CE81  5D00            C   dw          5D00h           ;                F20=X93
CE83  6700            C   dw          6700h           ;                F30=X103
CE85  7100            C   dw          7100h           ;                F40=X113
CE87  C2C2            C   dw  kbnum * 100h + kbnum    ;  69 45hNumLk   NumLk
CE89  C2C2            C   dw  kbnum * 100h + kbnum    ;                NumLk    NumLk
CE8B  CACA            C   dw  pause * 100h + pause    ;                Pause    Pause
CE8D  C2C2            C   dw  kbnum * 100h + kbnum    ;                NumLk    NumLk
CE8F  C3C3            C   dw  kbscr * 100h + kbscr    ;  70 46hScrLk   ScrLk
CE91  C3C3            C   dw  kbscr * 100h + kbscr    ;                ScrLk    ScrLk
CE93  C9C9            C   dw  kbbrk * 100h + kbbrk    ;                Break    Break
CE95  C3C3            C   dw  kbscr * 100h + kbscr    ;                ScrLk    ScrLk
                      C
                      C ;-------------------------------------------------------
                      C ;       Numeric Keypad                      |Olivetti|Other |
                      C ;                                           | KB     | KBs  |
                      C ;-------------------------------------------------------
                      C
CE97  4700            C   dw          4700h           ;  71 47hHome=X71
```

```
CE99  3737              C   dw  (37h) * 100h + (37h)      ;                 7       7
CE9B  7700              C   dw           7700h           ;                 X119
CE9D  D0D0              C   dw  kdec7 * 100h + kdec7      ;                 XDec7   XDec7
CE9F  4800              C   dw           4800h           ; 72 48hUp  =X72
CEA1  3838              C   dw  (38h) * 100h + (38h)      ;                 8       8
CEA3  CDCD              C   dw  kNONE * 100h + kNONE      ;                 None    None
CEA5  CFCF              C   dw  kdec8 * 100h + kdec8      ;                 XDec8   XDec8
CEA7  4900              C   dw           4900h           ; 73 49hPgUp=X73
CEA9  3939              C   dw  (39h) * 100h + (39h)      ;                 9       9
CEAB  8400              C   dw           8400h           ;                 X132
CEAD  CECE              C   dw  kdec9 * 100h + kdec9      ;                 XDec9   XDec9
CEAF  2D2D              C   dw  (2Dh) * 100h + (2Dh)      ; 74 4Ah-         -
CEB1  2D2D              C   dw  (2Dh) * 100h + (2Dh)      ;                 -       -
CEB3  CDCD              C   dw  kNONE * 100h + kNONE      ;                 None    None
CEB5  CDCD              C   dw  kNONE * 100h + kNONE      ;                 None    None
CEB7  4B00              C   dw           4B00h           ; 75 4BhLeft=X75
CEB9  3434              C   dw  (34h) * 100h + (34h)      ;                 4       4
CEBB  7300              C   dw           7300h           ;                 X115
CEBD  D3D3              C   dw  kdec4 * 100h + kdec4      ;                 XDec4   XDec4
CEBF  CDCD              C   dw  kNONE * 100h + kNONE      ; 76 4ChNone      None
CEC1  3535              C   dw  (35h) * 100h + (35h)      ;                 5       5
CEC3  CDCD              C   dw  kNONE * 100h + kNONE      ;                 None    None
CEC5  D2D2              C   dw  kdec5 * 100h + kdec5      ;                 XDec5   XDec5
CEC7  4D00              C   dw           4D00h           ; 77 4DhRght=X77
CEC9  3636              C   dw  (36h) * 100h + (36h)      ;                 6       6
CECB  7400              C   dw           7400h           ;                 X116
CECD  D1D1              C   dw  kdec6 * 100h + kdec6      ;                 XDec6   XDec6
CECF  2B2B              C   dw  (2Bh) * 100h + (2Bh)      ; 78 4Eh+         +
CED1  2B2B              C   dw  (2Bh) * 100h + (2Bh)      ;                 +       +
CED3  CDCD              C   dw  kNONE * 100h + kNONE      ;                 None    None
CED5  CDCD              C   dw  kNONE * 100h + kNONE      ;                 None    None
CED7  4F00              C   dw           4F00h           ; 79 4FhEnd =X79
CED9  3131              C   dw  (31h) * 100h + (31h)      ;                 1       1
CEDB  7500              C   dw           7500h           ;                 X117
CEDD  D6D6              C   dw  kdec1 * 100h + kdec1      ;                 XDec1   XDec1
CEDF  5000              C   dw           5000h           ; 80 50hDown=X80
CEE1  3232              C   dw  (32h) * 100h + (32h)      ;                 2       2
CEE3  CDCD              C   dw  kNONE * 100h + kNONE      ;                 None    None
CEE5  D5D5              C   dw  kdec2 * 100h + kdec2      ;                 XDec2   XDec2
CEE7  5100              C   dw           5100h           ; 81 51hPgDn=X81
CEE9  3333              C   dw  (33h) * 100h + (33h)      ;                 3       3
CEEB  7600              C   dw           7600h           ;                 X118
CEED  D4D4              C   dw  kdec3 * 100h + kdec3      ;                 XDec3   XDec3
CEEF  C0C0              C   dw  kbins * 100h + kbins      ; 82 52hINS=X82   INS=X82
CEF1  3030              C   dw  (30h) * 100h + (30h)      ;                 0       0
CEF3  CDCD              C   dw  kNONE * 100h + kNONE      ;                 None    None
CEF5  D7D7              C   dw  kdec0 * 100h + kdec0      ;                 XDec0   XDec0
CEF7  5300              C   dw           5300h           ; 83 53hDEL =X83
CEF9  2E2E              C   dw  (2Eh) * 100h + (2Eh)      ;                 .       .
CEFB  C8C8              C   dw  kbres * 100h + kbres      ;                 Reset   Reset
CEFD  C8C8              C   dw  kbres * 100h + kbres      ;                 Reset   Reset
                        C
                        C   ;-------------------------------------------------------
                        C   ;       Function Keypad                     |Olivetti|Other|
                        C   ;                                           |  KB    | KBs |
```

```
                        C ;--------------------------------------------------------
                        C
CEFF  D8CD              C   dw  kdbl0 * 100h + kNONE      ;  84 54h00       None
CF01  D8CD              C   dw  kdbl0 * 100h + kNONE      ;                 00        None
CF03  CDCD              C   dw  kNONE * 100h + kNONE      ;                 None      None
CF05  CDCD              C   dw  kNONE * 100h + kNONE      ;                 None      None
CF07  CBCD              C   dw  kbprt * 100h + kNONE      ;  85 55hPrtSc    None
CF09  CBCD              C   dw  kbprt * 100h + kNONE      ;                 PrtSc     None
CF0B  7200              C   dw          7200h             ;                 X114
CF0D  CDCD              C   dw  kNONE * 100h + kNONE      ;                 None      None
CF0F  CACD              C   dw  pause * 100h + kNONE      ;  86 56hPause    None
CF11  CACD              C   dw  pause * 100h + kNONE      ;                 Pause     None
CF13  CACD              C   dw  pause * 100h + kNONE      ;                 Pause     None
CF15  CACD              C   dw  pause * 100h + kNONE      ;                 Pause     None
CF17  0D0D              C   dw  (0Dh) * 100h + (0Dh)      ;  87 57hCR       CR
CF19  0D0D              C   dw  (0Dh) * 100h + (0Dh)      ;                 CR        CR
CF1B  0A0A              C   dw  (0Ah) * 100h + (0Ah)      ;                 LF        LF
CF1D  CDCD              C   dw  kNONE * 100h + kNONE      ;                 None      None
CF1F  4B00              C   dw          4B00h             ;  88 58hLeft=X75
CF21  7300              C   dw          7300h             ;                 Rev Word = X115
CF23  7300              C   dw          7300h             ;                 Rev Word = X115
CF25  7300              C   dw          7300h             ;                 Rev Word = X115
CF27  5000              C   dw          5000h             ;  89 59hDown=X80
CF29  5000              C   dw          5000h             ;                 Down=X80
CF2B  5000              C   dw          5000h             ;                 Down=X80
CF2D  5000              C   dw          5000h             ;                 Down=X80
CF2F  4D00              C   dw          4D00h             ;  90 5AhRght=X77
CF31  7400              C   dw          7400h             ;                 Adv Word = X116
CF33  7400              C   dw          7400h             ;                 Adv Word = X116
CF35  7400              C   dw          7400h             ;                 Adv Word = X116
CF37  4800              C   dw          4800h             ;  91 5BhUp  =X72
CF39  4700              C   dw          4700h             ;                 Home=X71
CF3B  4700              C   dw          4700h             ;                 Home=X71
CF3D  4700              C   dw          4700h             ;                 Home=X71
CF3F  C9CD              C   dw  kbbrk * 100h + kNONE      ;  92 5ChBreak    Break
CF41  C9CD              C   dw  kbbrk * 100h + kNONE      ;                 Break     Break
CF43  C9CD              C   dw  kbbrk * 100h + kNONE      ;                 Break     Break
CF45  C9CD              C   dw  kbbrk * 100h + kNONE      ;                 Break     Break
CF47  C9CD              C   dw  kbbrk * 100h + kNONE      ;  93 5DhBreak    Break
CF49  C9CD              C   dw  kbbrk * 100h + kNONE      ;                 Break     Break
CF4B  C9CD              C   dw  kbbrk * 100h + kNONE      ;                 Break     Break
CF4D  C9CD              C   dw  kbbrk * 100h + kNONE      ;                 Break     Break
CF4F  C2C2              C   dw  kbnum * 100h + kbnum      ;  94 5EhNumLk    NumLk
CF51  C2C2              C   dw  kbnum * 100h + kbnum      ;                 NumLk     NumLk
CF53  CACA              C   dw  pause * 100h + pause      ;                 Pause     Pause
CF55  C2C2              C   dw  kbnum * 100h + kbnum      ;                 NumLk     NumLk
CF57  2F2F              C   dw  (2Fh) * 100h + (2Fh)      ;  95 5Fh/        /
CF59  2F2F              C   dw  (2Fh) * 100h + (2Fh)      ;                 /         /
CF5B  CDCD              C   dw  kNONE * 100h + kNONE      ;                 None      None
CF5D  CDCD              C   dw  kNONE * 100h + kNONE      ;                 None      None
CF5F  5400              C   dw          5400h             ;  96 60hF11=X84
CF61  5400              C   dw          5400h             ;                 F11=X84
CF63  5400              C   dw          5400h             ;                 F11=X84
CF65  5400              C   dw          5400h             ;                 F11=X84
CF67  5500              C   dw          5500h             ;  97 61hF12=X85
```

```
CF69  5500                C   dw          5500h           ;               F12=X85
CF6B  5500                C   dw          5500h           ;               F12=X85
CF6D  5500                C   dw          5500h           ;               F12=X85
CF6F  5600                C   dw          5600h           ;  98 62hF13=X86
CF71  5600                C   dw          5600h           ;               F13=X86
CF73  5600                C   dw          5600h           ;               F13=X86
CF75  5600                C   dw          5600h           ;               F13=X86
CF77  5700                C   dw          5700h           ;  99 63hF14=X87
CF79  5700                C   dw          5700h           ;               F14=X87
CF7B  5700                C   dw          5700h           ;               F14=X87
CF7D  5700                C   dw          5700h           ;               F14=X87
CF7F  5800                C   dw          5800h           ; 100 64hF15=X88
CF81  5800                C   dw          5800h           ;               F15=X88
CF83  5800                C   dw          5800h           ;               F15=X88
CF85  5800                C   dw          5800h           ;               F15=X88
CF87  5900                C   dw          5900h           ; 101 65hF16=X89
CF89  5900                C   dw          5900h           ;               F16=X89
CF8B  5900                C   dw          5900h           ;               F16=X89
CF8D  5900                C   dw          5900h           ;               F16=X89
CF8F  5A00                C   dw          5A00h           ; 102 66hF17=X90
CF91  5A00                C   dw          5A00h           ;               F17=X90
CF93  5A00                C   dw          5A00h           ;               F17=X90
CF95  5A00                C   dw          5A00h           ;               F17=X90
CF97  5B00                C   dw          5B00h           ; 103 67hF18=X91
CF99  5B00                C   dw          5B00h           ;               F18=X91
CF9B  5B00                C   dw          5B00h           ;               F18=X91
CF9D  5B00                C   dw          5B00h           ;               F18=X91
                          C
                          C kb_data1      endp
                          C
CF9F                      C code ends

                          C include hdu.asm
                          C
                          C ;===============================================================
                          C ;
                          C ;    Basic Input/Ouput System code for the ST506 SASI
                          C ;    hard disk contollers.
                          C ;
                          C ;    This code provides access to fixed disks compatible
                          C ;    with IBM XT/PC fixed disks.
                          C ;
                          C ;    Please note that the following Bios routines must
                          C ;    be entered solely through software interrupts.
                          C ;    Other references to this code violate the structure
                          C ;    and design thus jeopardizing correct functioning.
                          C ;
                          C ;---------------------------------------------------------------
                          C
CF9F                      C code segment  public   'ROM'
                          C       assume   cs:code,ds:nothing,es:nothing,ss:nothing
                          C
                          C ;Error returned by the bios:
= 00FF                    C h_status      EQU      0FFh            ;Read status failed
= 00BB                    C h_undefd      EQU      0BBh            ;Undefined error
```

```
= 0011                  C  h_ecc_cor     EQU     011h            ;ECC error in data corrected
= 000B                  C  h_bad_trk     EQU     00Bh            ;bad track
= 0007                  C  h_initz               EQU     007h            ;drive initialization failed
= 0005                  C  h_reset               EQU     005h            ;reset failed
= 0000                  C  h_no_err      EQU     000             ;no error found!
                        C
                        C  ;Error codes returned by the disk controllers:
= 0018                  C  h_erc         EQU     18h             ;ECC Corrected.
                        C
                        C  ;INT 13h Command Codes:
= 0000                  C  h_rstEQU      0                       ;controller reset
= 0001                  C  h_cc EQU      1                       ;last completion code return
= 0002                  C  h_rd EQU      2                       ;sector(s) read
= 0003                  C  h_wr EQU      3                       ;sector(s) write
= 0004                  C  h_vr EQU      4                       ;sector(s) verify
= 0005                  C  h_ft EQU      5                       ;track format
= 0006                  C  h_ftbEQU      6                       ;bad track format
= 0007                  C  h_fd EQU      7                       ;drive format
= 0008                  C  h_p_rEQU      8                       ;fetch drive parameters
= 0009                  C  h_p_sEQU      9                       ;assign drive parameters
= 000A                  C  h_rdlEQU      10                      ;read long
= 000B                  C  h_wrlEQU      11                      ;write long
= 000C                  C  h_sk EQU      12                      ;seek cylinder
= 000D                  C  h_rst_1       EQU     13                      ;controller reset (as " 0" )
= 000E                  C  h_bf_r        equ     14                      ;read content of sector buffer
= 000F                  C  h_bf_w        EQU     15                      ;write content of sector buffer to disk.
= 0010                  C  h_rdyEQU      16                      ;test hard disk drive readiness.
= 0011                  C  h_rclEQU      17                      ;recalibrate hard disk (single step to
                        C                                        ;cylinder.
= 0012                  C  h_dg_r        EQU     18                ;run internal ram diagnostics
= 0013                  C  h_dg_d        EQU     19                ;run disk drive diagnostics
= 0014                  C  h_dg_c        EQU     20                ;run controller diagnostics (disk is NOT accessed)
                        C  ;The following are the command codes as used by ST506 disk controllers.
                        C  ;They are translated from INT 13h command codes and sent to the controller
                        C  ;thru the Command Description Block by this BIOS.
= 0000                  C  hc_rdy        EQU     0                       ;ready test
= 0001                  C  hc_rcl        EQU     1                       ;recalibrate
= 0003                  C  hc_stat       EQU     3                       ;request sense (status)
= 0004                  C  hc_fdEQU      4                       ;format drive (from a given address)
= 0005                  C  hc_vrEQU      5                       ;read and verify sector(s)
= 0006                  C  hc_ftEQU      6               ;format track (no flags set in ID fields; of sectors on track(s).
= 0007                  C  hc_fbt        EQU     7               ;format bad track
= 0008                  C  hc_rdEQU      8                       ;read sector(s)
= 000A                  C  hc_wrEQU      0AH                     ;write sector(s) to disk
= 000B                  C  hc_skEQU      0Bh                     ;seek to specified cylinder
= 000C                  C  hc_p_s        EQU     0Ch                     ;assign parameters to controller (disk
                        C                                        ;is not accessed)
= 000D                  C  hc_ecc        EQU     0DH                     ;read ECC burst error length
= 000E                  C  hc_bf_r       EQU     0EH                     ;read controller buffer
= 000F                  C  hc_bf_w       EQU     0FH                     ;write controller buffer to disk
= 00E0                  C  hc_dg_r       EQU     0E0H                    ;run internal ram diagnostics
= 00E3                  C  hc_dg_d       EQU     0E3H                    ;run disk drive diagnostics
= 00E4                  C  hc_dg_c       EQU     0E4H                    ;run controller diagnostics (no disk
                        C                                        ;access!)
= 00E5                  C  hc_rdl        EQU     0E5H                    ;read long (512data+4ECC)bytes
```

```
= 00E6                  C  hc_wrl       EQU     0E6H                    ;write long (as above)
                        C
                        C  ;Command Description Block (CDB) offsets:
= 0000                  C  ccb_cmd              EQU     0               ;command byte offset
= 0004                  C  ccb_blks     EQU     4               ;block number/interlv factor offset.
= 0005                  C  ccb_opt              EQU     5               ;option byte offset.
                        C
                        C  ;Logical Unit Number (LUN) 1 (Other is 0) is set in CDB 2nd byte by
= 0020                  C  ccb_drv_b    equ     20h
                        C
                        C  ;Command Completion byte:
= 0002                  C  cc_erequ     2                       ;error flag bit (if 0 then no error)
                        C
                        C  ;Offsets in disk parameter table:
= 0000                  C  wst_cyl              EQU     0               ;word, number of cylinder
= 0002                  C  wst_heads    EQU     2               ;byte, number of heads.
= 0003                  C  wst_re_wr    EQU     3               ;word, reduced wr. current cyl. start
= 0005                  C  wst_wr_pre   EQU     5               ;word, start wr. precomp. cylinder
= 0007                  C  wst_er_bur   EQU     7               ;byte, error burst lenght
= 0008                  C  wst_opt              EQU     8               ;byte, option (step rate, error retry)
                        C
                        C  ;Activation Record Description:
= 000A                  C  a1_dx        EQU     10              ;
= 000C                  C  a1_cx        EQU     12
= 000E                  C  a1_bx        EQU     14
                        C
                        C  ;Timer values:
                        C  ;ti_0_1              EQU     ti_fin-570d     ;wait time after drive 0 is found
                        C                                       ;usable (apprx. 15 seconds).
= 0196                  C  ti_k_rst     EQU     ti_fin-40d      ;wait time after keyboard reset
                        C                                       ;(approx. 2 seconds).
= 01BE                  C  ti_fin               EQU     446d            ;max. wait at cold pwr-up during
                        C                                       ;reset (approx.25 seconds).
                        C
                        C  ;Miscellaneous values:
= 0200                  C  sec_size     EQU     512d            ;number of bytes per sector.
= 0320                  C  p_wx2        EQU     0320H           ;first controller I/O offset.
                        C
                        C  ;Offset from controller BASE port:
                        C  ;            READ operations:
= 0001                  C  hd_st        EQU     1               ;status port.
= 0002                  C  hd_cfg               EQU     2               ;configuration switches.
                        C
                        C  ;            WRITE operations:
= 0001                  C  hd_rs        EQU     1               ;reset controller.
= 0002                  C  hd_sl        EQU     2               ;select controller port.
= 0003                  C  hd_msk               EQU     3               ;DMA & INTRQ mask register port.
                        C
                        C  ;Bits in status register (offset 1):
= 0008                  C  h_busy               EQU     00001000B;Busy (Selected).
= 0004                  C  h_cd         EQU     00000100B;Command/Data.
= 0002                  C  h_io?        EQU     00000010B;Input/Output
= 0001                  C  h_rqs        EQU     00000001B;Request (Ready) (in WX2 controller this
                        C                                       ;bit starts state machine!).
= 0020                  C  h_intp               equ     00100000B;Interrupt requested by controller.
```

```
= 0010                  C  h_drq          EQU     00010000B;DMA Requested by controller.
                        C
= 0001                  C  h_mkdma                EQU     01B               ;DMA mask bit (0 disabled).
= 0002                  C  h_mkint                EQU     10B               ;Intreq mask bit (0 disabled)
                        C
                        C  ;Commands for DMA chip 8237:
                        C  ;              Mode register:
= 0047                  C  h_dma_w                EQU     01000111B;write, single mode select
                        C                                           ;channel (chn) 3 selected.
= 004B                  C  h_dma_r                EQU     01001011B;read, single mode select
                        C                                           ;chn 3 selected.
                        C  ;
                        C  ;              Mask register:
= 0003                  C  h_dma_3                EQU     11B               ;Chn 3 mask bits
= 0004                  C  h_dma_s                EQU     100B              ;Set mask bit
                        C
                        C  ;Bits in Programmable Interrupt Controller (PIC):
= 0001                  C  h_ocw1_m0      EQU     00000001B;Chn 0 mask (Chn 0 used by clock!).
= 0020                  C  h_ocw1_m5      EQU     00100000B;Chn 5 mask (used by the hd controller).
                        C
                        C
                        C  ; Winchester Parameter Tables.
                        C  ;
                        C  ;  Offset
                        C  ;   0  - 15   subtable 0        CDC Wren  1 30 MB full height (hgt)
                        C  ;   16 - 31   subtable 1        Seagate ST225 20MB
                        C  ;                               CMI 4426 20 MB Full hgt.
                        C  ;   32 - 47   subtable 2        Tandon TM262 10 MB
                        C  ;   48 - 63   subtable 3        Standard 10 MB
                        C  ;   64 - 79   subtable 4        Miniscribe 3425 20MB
                        C  ;                               OPE XM5220-1 20 MB
                        C  ;   80 - 95   subtable 5        CDC Wren 1 30 MB
                        C  ;   96 -111   subtable 6        Seagate ST225 20 MB
                        C  ;                               CMI 4426 20 MB
                        C  ;  112 -127   subtable 7        Miniscribe 3425 20 MB
                        C  ;                               OPE XM5220-1 20 MB
                        C  ;  128 -143   subtable 8        CMI6426 40 MB
                        C  ;  144 -159   subtable 9        Miniscribe 6053 45 MB
                        C  ;  160 - 175  subtable 10       Tandon TM755 42 MB
                        C  ;  176 -191   subtable 11       CDC Wren II Slim 51 MB
                        C  ;  192 -297   subtable 12       CDC Wren II 67 MB
                        C  ;  208 -223   subtable 13       Micropolis 1325 68 MB
                        C  ;                               Miniscribe 6085
                        C  ;                               Seagate 4085
                        C  ;  224 -239   subtable 14       Seagate ST251 40 MB
                        C  ;  240 -255   subtable 15       Used only to indicate no drive present for
                        C  ;                               second drive.
                        C  ;
                        C  ;   The configuration switches accessible via port wx2_r_config specify
                        C  ; the subtable to be used for each drive.  Each drive has two switches
                        C  ; which determine a 2-bit field in port wx2_r_config which specifies
                        C  ; the subtable number for the drive.
                        C  ;
                        C  ; Winchester Parameter Subtable
                        C  ;
```

```
                        C  ; Offset     Byte Length
                        C  ;  0          2          Number of cylinders
                        C  ;  2          1          Number of heads
                        C  ;  3          2          Starting reduced write current cylinder
                        C  ;  5          2          Starting write precompensation cylinder
                        C  ;  7          1          Max. correctable error burst length
                        C  ;  8          1          CCB option byte
                        C  ;  9          7          Reserved for future use
                        C  ;
                        C  ;   CCB option byte
                        C  ;  Bit
                        C  ;   0, 1, 2   Step option
                        C  ;             Minimum Step Rates allowed:
                        C  ;             DTC 5150BX : 13usec.
                        C  ;             WD10002-SWX2 with 1015-14 : 70usec.
                        C  ;             WD10002-SWX2 with 1015-24 : 35usec.
                        C  ;
                        C  ;   3, 4, 5   Reserved for future use
                        C  ;   6        0 --> Stable ECC required before correction
                        C  ;          1 --> Immediate ECC correction
                        C  ;   7        0 --> Retries allowed
                        C  ;          1 --> No retries allowed
                        C  ;
                        C
                        C  parm_tbl      struc
0000  0132              C  p_cyl         dw       306d     ;maximum cylinder number.
0002  04                C  p_heads               db       4d       ;maximum number of heads.
0003  0132              C  p_write_cur   dw       306d     ;Starting reduced write current cyl.
0005  0080              C  p_precomp     dw       128d     ;Starting Write precompensation cyl.
0007  0B                C  p_ecc_len     db       11d      ;ECC correctable error burst length.
0008  05                C  p_control_byte db      5d       ;CDB option byte.
0009  0C                C  p_timeout     db       12d      ;Standard operation timeout (at 10MHz) (base).
000A  2D                C  p_fmt_timeout db       45d      ;Drive formatting timeout (10MHz) (base).
000B  28                C  p_drvdiag_timeout      db       40d ;Diagnostic timeout (base).
000C  00 00 00 00       C  p_zzjdb       0,0,0,0           ;not used.
0010                    C  parm_tbl      ends
                        C
CF9F                    C  h_params      proc     near
                        C
CF9F                    C  hdu_parm_tbl:
CF9F  02B9              C       parm_tbl <697d,5d,128d,0,...,135>    ;CDC Wren1 30 MB
CFA1  05                C
CFA2  0080              C
CFA4  0000              C
CFA6  0B                C
CFA7  05                C
CFA8  0C                C
CFA9  87                C
CFAA  28                C
CFAB  0001[             C
          00            C
          00            C
          00            C
          00            C
                    ]   C
```

```
                                C
CFAF  0264                      C       parm_tbl <612d,4d,256d,256d,...,5ah> ;Seagate ST225 20 MB
CFB1  04                        C
CFB2  0100                      C
CFB4  0100                      C
CFB6  0B                        C
CFB7  05                        C
CFB8  0C                        C
CFB9  5A                        C
CFBA  28                        C
CFBB  0001[                     C
            00                  C
            00                  C
            00                  C
            00                  C
                        ]       C
                                C
                                C                                                       ;CMI 4426 20 MB
CFBF  0264                      C       parm_tbl <612d,4d,612d,612d,...,5ah> ;Tandon TM262
CFC1  04                        C
CFC2  0264                      C
CFC4  0264                      C
CFC6  0B                        C
CFC7  05                        C
CFC8  0C                        C
CFC9  5A                        C
CFCA  28                        C
CFCB  0001[                     C
            00                  C
            00                  C
            00                  C
            00                  C
                        ]       C
                                C
CFCF  0132                      C       parm_tbl <>                                     ;Standard 10 MB
CFD1  04                        C
CFD2  0132                      C
CFD4  0080                      C
CFD6  0B                        C
CFD7  05                        C
CFD8  0C                        C
CFD9  2D                        C
CFDA  28                        C
CFDB  0001[                     C
            00                  C
            00                  C
            00                  C
            00                  C
                        ]       C
                                C
CFDF  0264                      C       parm_tbl <612,4,128,128,...,90>                 ;Miniscribe  3425 20 MB
CFE1  04                        C
CFE2  0080                      C
CFE4  0080                      C
CFE6  0B                        C
```

```
CFE7  05                C
CFE8  0C                C
CFE9  5A                C
CFEA  28                C
CFEB  0001[             C
          00            C
          00            C
          00            C
          00            C
                  ]     C
                        C
                        C                                                   ;OPE XM5220-1 20 MB
CFEF  02B9              C       parm_tbl <697d,5d,128d,0,,,,135>    ;CDC Wren1 30 MB
CFF1  05                C
CFF2  0080              C
CFF4  0000              C
CFF6  0B                C
CFF7  05                C
CFF8  0C                C
CFF9  87                C
CFFA  28                C
CFFB  0001[             C
          00            C
          00            C
          00            C
          00            C
                  ]     C
                        C
CFFF  0264              C       parm_tbl <612d,4d,256d,256d,,,,5ah> ;Seagate ST225 20MB
D001  04                C
D002  0100              C
D004  0100              C
D006  0B                C
D007  05                C
D008  0C                C
D009  5A                C
D00A  28                C
D00B  0001[             C
          00            C
          00            C
          00            C
          00            C
                  ]     C
                        C
                        C                                                   ;CMI 4426 20 MB
D00F  0264              C       parm_tbl <612,4,128,128,,,,90>              ;Miniscribe 3425 20 MB
D011  04                C
D012  0080              C
D014  0080              C
D016  0B                C
D017  05                C
D018  0C                C
D019  5A                C
D01A  28                C
D01B  0001[             C
```

```
                00              C
                00              C
                00              C
                00              C
                        ]       C
                                C
                                C                                       ;OPE XM5220-1 20 MB
D01F  0280              C       parm_tbl <640d,4d,256d,256d,...,5ah> ;CMI 6426 40 MB
D021  04                C
D022  0100              C
D024  0100              C
D026  0B                C
D027  05                C
D028  0C                C
D029  5A                C
D02A  28                C
D02B  0001[             C
                00      C
                00      C
                00      C
                00      C
                        ]       C
                                C
D02F  0400              C       parm_tbl <1024,5,512,512,...,203>       ;Miniscribe 6053 45 MB
D031  05                C
D032  0200              C
D034  0200              C
D036  0B                C
D037  05                C
D038  0C                C
D039  CB                C
D03A  28                C
D03B  0001[             C
                00      C
                00      C
                00      C
                00      C
                        ]       C
                                C
D03F  03D5              C       parm_tbl <981,5,981,981,...,189>        ;Tandon TM755 42MB
D041  05                C
D042  03D5              C
D044  03D5              C
D046  0B                C
D047  05                C
D048  0C                C
D049  BD                C
D04A  28                C
D04B  0001[             C
                00      C
                00      C
                00      C
                00      C
                        ]       C
                                C
```

```
D04F  03D5                C       parm_tbl <981,5,981,128,,,,255>          ;CDC Wren II Slim 51 MB
D051  05                  C
D052  03D5                C
D054  0080                C
D056  0B                  C
D057  05                  C
D058  0C                  C
D059  FF                  C
D05A  28                  C
D05B  0001[               C
            00            C
            00            C
            00            C
            00            C
                      ]   C
                          C
D05F  039D                C       parm_tbl <925,7,925,0,,,,255>            ;CDC Wren II 67 MB
D061  07                  C
D062  039D                C
D064  0000                C
D066  0B                  C
D067  05                  C
D068  0C                  C
D069  FF                  C
D06A  28                  C
D06B  0001[               C
            00            C
            00            C
            00            C
            00            C
                      ]   C
                          C
D06F  0400                C       parm_tbl <1024,8,1024,1024,,,,255>  ;Micropolis 1325 68 MB
D071  08                  C
D072  0400                C
D074  0400                C
D076  0B                  C
D077  05                  C
D078  0C                  C
D079  FF                  C
D07A  28                  C
D07B  0001[               C
            00            C
            00            C
            00            C
            00            C
                      ]   C
                          C
                          C                                                  ;Miniscribe 6085
                          C                                                  ;Seagate 4085
D07F  0334                C       parm_tbl <820,6,820,256,,,,200>          ;Seagate ST251 40 MB
D081  06                  C
D082  0334                C
D084  0100                C
D086  0B                  C
```

```
D087  05                  C
D088  0C                  C
D089  C8                  C
D08A  28                  C
D08B  0001[               C
            00            C
            00            C
            00            C
            00            C
                      ]   C
                          C
D08F  0132                C       parm_tbl <>                                 ;Indicates no drive
D091  04                  C
D092  0132                C
D094  0080                C
D096  0B                  C
D097  05                  C
D098  0C                  C
D099  2D                  C
D09A  28                  C
D09B  0001[               C
            00            C
            00            C
            00            C
            00            C
                      ]   C
                          C
                          C                                                     ; present for 2nd. drive
                          C h_params      endp
                          C
                          C ;************** HDU INITIALIZATION *****************
                          C
                          C       assume  ds:abs0 ,es:abs0
D09F                      C h_init        proc    near
D09F  33 C0               C       xor     ax,ax               ;clear ax
D0A1  8E D8               C       mov     ds,ax               ;initialize data segment register
D0A3  8E C0               C       mov     es,ax               ;and Extra segment.
                          C
                          C ;Set interrupt Vectors to install BIOS into Operating System.
D0A5  FA                  C       cli                         ;disable INT's while assigning vectors.
                          C
                          C ;Install hardware hdu interrupt service routine:
D0A6  C7 06 0034 R D23B R C       mov     word ptr ds:[int0Dlocn+0000h],cs:(offset h_int)
D0AC  8C 0E 0036 R        C       mov     word ptr ds:[int0Dlocn+0002h],cs
                          C
                          C ;Transfer OLD FDU INT 13h vector to the new INT 40h location:
D0B0  A1 004C R           C       mov     ax,word ptr ds:[int13locn+0000h]
D0B3  A3 0100             C       mov     word ptr ds:[(4*40h)+0000h],ax
                          C
D0B6  A1 004E R           C       mov     ax,word ptr ds:[int13locn+0002h]
D0B9  A3 0102             C       mov     word ptr ds:[(4*40h)+0002h],ax
                          C
                          C ;Install new HDU request INT 13h vector:
D0BC  C7 06 004C R D24C R C       mov     word ptr ds:[int13locn+0000h],cs:(offset h_io)
D0C2  8C 0E 004E R        C       mov     word ptr ds:[int13locn+0002h],cs
```

```
                                C
                                C  ;Install new HDU BOOT-STRAP INT 19h vector:
0DC6  C7 06 0064 R D1B3 R       C       mov     word ptr ds:[int19locn+0000h],cs:(offset h_boot)
0DCC  8C 0E 0066 R              C       mov     word ptr ds:[int19locn+0002h],cs
                                C
                                C  ;Install new INT 41h HDU Parameter Table Vector:
0DD0  C7 06 0104 CF9F R         C       mov     word ptr ds:[(4*41h)+0000h],cs:(offset hdu_parm_tbl)
0DD6  8C 0E 0106                C       mov     word ptr ds:[(4*41h)+0002h],cs
                                C
0DDA  FB                        C       sti                     ;exit critical code.
                                C
                                C       assume  ds:data
0DDB  B8 0040                   C       mov     ax,data_seg
0DDE  8E D8                     C       mov     ds,ax           ;load " data_seg" address.
                                C
0DE0  C6 06 0074 R 00           C       mov     disk_status,0   ;clear previous disk status
0DE5  C6 06 0075 R 00           C       mov     hf_num,00;clear number of hd drives
0DEA  B8 0196                   C       mov     ax,ti_k_rst     ;initial timer value for keyboard reset
0DED  81 3E 0072 R 1234         C       cmp     reset_flag,1234h ;keyboard reset?
0DF3  74 0A                     C       je      k_rst           ;jump if so
0DF5  81 3E 0072 R 4321         C       cmp     reset_flag,4321h ;pushbutton reset?
0DFB  74 02                     C       je      k_rst           ;jump if so
                                C  ;nop
                                C  ;nop
0DFD  33 C0                     C       xor     ax,ax           ;otherwise clear RAX.
0DFF                            C  k_rst:
0DFF  A3 006C R                 C       mov     t_low_order,ax  ;initialize time counter
0D102  FA                       C       cli                     ;disable interrupts while accessing the PIC.
0D103  E4 21                    C       in      al,pic_1 ;read INT mask register
0D105  24 FE                    C       and     al,not h_ocw1_m0  ;enable INT's on chn 0
0D107  E6 21                    C       out     pic_1,al ;Set INT mask register & start the timer
0D109  FB                       C       sti                     ;exit critical code.
                                C
                                C     ; Count the usable winchester drives.  To speed this counting, if a
                                C     ; controller is unusable, then testing of its second drive is skipped.
                                C     ; A drive is usable if and only if its controller resets, runs its
                                C     ; controller diagnostic, recalibrates the drive, and executes test
                                C     ; drive ready all without error.
                                C
0D10A  B2 80                    C       mov     dl,80h          ;base Winchester drive number
0D10C                           C  ctlr_init:
0D10C  B4 00                    C       mov     ah,h_rst ;RESET function code
0D10E  CD 13                    C       int     13h             ;call disk BIOS.
                                C
0D110  72 16                    C       jc      nxt_ctlr ;jump if error
0D112  B4 14                    C       mov     ah,h_dg_c;BIOS call code
0D114  CD 13                    C       int     13h             ;call disk Bios
                                C
0D116  72 10                    C       jc      nxt_ctlr ;jump if error
                                C
0D118                           C  tst_drv_rdy:
0D118  B4 10                    C       mov     ah,h_rdy ;TEST DRIVE READY function code
0D11A  CD 13                    C       int     13h             ;call BIOS code
                                C
0D11C  73 2B                    C       jnc     drv_rdy         ;jump if drive is ready
```

```
D11E  81 3E 006C R 01BE     C       cmp     t_low_order,ti_fin;time-out?
D124  72 F2                 C       jb      tst_drv_rdy       ;jump if no time-out yet
                            C
D126  EB 2B                 C       jmp     short nxt_drv     ;go to next drive
                            C
D128                        C  nxt_ctlr:
D128  FE C2                 C       inc     dl
D12A  EB 27                 C       jmp     short nxt_drv     ;skip drive 1 on unusable controller
                            C
D12C  4E 6F 74 20 52 65 61  C  mes_1701:    db      'Not Ready',0Dh,0Ah,NUL
      64 79 0D 0A 00        C
D138  20 44 69 73 6B 28 73  C  mes_rdy:     db      ' Disk(s) Ready',0dh,0Ah,NUL
      29 20 52 65 61 64 79  C
      0D 0A 00              C
                            C
D149                        C  drv_rdy:
D149  B4 11                 C       mov     ah,h_rcl ;BIOS call code    recallibrate
D14B  CD 13                 C       int     13h               ;call disk Bios
D14D  72 04                 C       jc      nxt_drv           ;jump if error
                            C
D14F  FE 06 0075 R          C       inc     hf_num            ; update drive count
                            C
                            C  ;cmp dl,80h           ;first drive?
                            C  ;jne nxt_drv          ;jump if not
                            C  ;cmp reset_flag,4321h ;is it a pushbutton reset?
                            C  ;je  nxt_drv          ;jump if so
                            C  ;nop
                            C  ;nop
                            C  ;cmp reset_flag,1234h ;is it a keyboard reset?
                            C  ;je  nxt_drv          ;jump if so
                            C
                            C  ;;movt_low_order,ti_0_1;1st drive usable, thus wait only
                            C                                   ;up to 15 sec for other drives
D153                        C  nxt_drv:
D153  FE C2                 C       inc     dl                ;increment drive number
D155  F6 C2 01              C       test    dl,1              ;test for second drive (drv #1) on controller.
D158  74 28                 C       jz      tst_nxt_cntrlr    ;if not 2nd drive, test for next controller
                            C
                            C  ; Check for second drive defined as drive type 15: skip spinup if it is
                            C
D15A  52                    C       push    dx                ;save drive designator
D15B  50                    C       push    ax                ;save AX, just in case
D15C  80 EA 80              C       sub     dl,80h            ;origin number to zero
D15F  81 E2 00FE            C       and     dx,0FEh           ;isolate drive designator
D163  D0 E2                 C       shl     dl,1              ;create index for port address
D165  8B F2                 C       mov     si,dx             ;set up si as port offset
D167  8D 94 0322            C       lea     dx,p_wx2+hd_cfg[si] ;get port address
D16B  EC                    C       in      al,dx             ;read switch setting from controller
D16C  24 33                 C       and     al,00110011b      ;isolate switch bits for drive 2
D16E  8A E0                 C       mov     ah,al             ;copy switch-bits, to be shifted
D170  D0 EC                 C       shr     ah,1              ;shift the switch-bits, to align the
D172  D0 EC                 C       shr     ah,1              ; bits to create a drive-type
D174  0A C4                 C       or      al,ah             ;now al = drive type in low nibble
D176  24 0F                 C       and     al,0Fh            ;isolate drive type
D178  3C 0F                 C       cmp     al,15             ;drive type 15 ? (means no drive present)
```

```
D17A  58                C       pop     ax              ;restore AX
D17B  5A                C       pop     dx              ;restore drive and head number
D17C  75 9A             C       jne     tst_drv_rdy     ;if not drive type 15, go test for drive ready
D17E  FE C2             C       inc     dl              ;if type 15, address drive 0 of next controller
D180  EB 00             C       jmp short tst_nxt_cntrlr ;and skip spin-up of drive 1 n this controller
                        C
D182                    C  tst_nxt_cntrlr:
D182  80 FA 88          C       cmp     dl,80h+8 ;are there more hdu controllers to test?
D185  73 02             C       jae     chk_any_disks   ;jump if no morecontrollers to be tested
D187  EB 83             C       jmp     ctlr_init;jump if more controllers to be tested
D189                    C  chk_any_disks:
D189  80 3E 0075 R 00   C       cmp     hf_num,0 ;Check how many disks passed above boot tests.
D18E  75 0C             C       jne     wins_usable     ;jump if some (or all) did.
                        C
                        C  ;Since no disks passed the above boot tests, the " Not ready"  message is
                        C  ;displayed and the error flag is set on exit.
                        C
D190  BE D12C R         C       mov     si,offset mes_1701 ;point to messsage string " Not Ready" .
D193  E8 DFFA R         C       call    DRomString      ;call routine to print out string ended by NUL.
D196  BD 000F           C       mov     bp,15d          ;set error flag on exit.
D199  EB 17 90          C       jmp     d_c_2           ;exit (return).
                        C
D19C                    C  wins_usable:
D19C  50                C       push    ax              ;save RAX and RCX:
D19D  51                C       push    cx
D19E  A0 0075 R         C       mov     al,hf_num;get disk drive count.
D1A1  0C 30             C       or      al,30h          ;convert it to ASCII character.
D1A3  B9 0001           C       mov     cx,1            ;one character to be printed out.
D1A6  B4 0E             C       mov     ah,14           ;bios video call code.
D1A8  CD 10             C       int     10h             ;call video bios.
D1AA  59                C       pop     cx              ;restore RAX and RCX:
D1AB  58                C       pop     ax
D1AC  BE D138 R         C       mov     si,offset mes_rdy  ;point to message string "  Disk(s) Ready"
D1AF  E8 DFFA R         C       call    DRomString      ;print string.
D1B2  C3                C  d_c_2:       ret                     ;exit.
                        C  h_init       endp
                        C
                        C  ;=====================================
                        C  ;
                        C  ; Call:     int   19h
                        C  ;
                        C  ;  Boot the operating system from diskette A or from the first usable
                        C  ;  winchester drive.
                        C  ;
                        C  ; Entry:   No entry parameters.
                        C  ;
                        C  ; Normal Exit:Jump to the start of the bootstrap sector read from disk.
                        C  ;
                        C  ; Error Exit: int  18h  -- ROM resident BASIC entered.
                        C  ;    The ROM resident BASIC is entered if the system can't be booted.
                        C  ;
                        C  ;    Attempt to reset drive 0 (diskette " A" ) and then to read sector 1,
                        C  ; cylinder 0, head 0.  If this read is successful, then jump to the start
                        C  ; of this sector in memory.  Retry the preceding three times.
                        C  ;
```

```
                        C  ;    Beginning with drive 80h (the first winchester) and ending with
                        C  ; drive 87h (the last possible winchester) do the following:  Attempt to
                        C  ; reset the drive and then to read sector 1, cylinder 0, head 0.  If this
                        C  ; read is successful and the last word of this sector contains AA55h, then
                        C  ; jump to the start of this sector in memory.
                        C  ;
                        C  ;    Otherwise, enter the ROM resident BASIC by executing int 18h.
                        C
D1B3                    C  h_boot       proc    far             ;INT 19h could be called by other segments!
                        C       assume  ds:abs0,es:abs0
D1B3  33 C0             C       xor     ax,ax           ;clear RAX.
D1B5  8E D8             C       mov     ds,ax           ;load RDS and RES with proper register.
D1B7  8E C0             C       mov     es,ax
                        C
                        C  ; Install pointers to the default diskette parameter table and the
                        C  ; default winchester parameter table.
                        C
D1B9  FA                C       cli                     ;disable interrrupts during critical code.
                        C
                        C  ;install vector to hard disk parameter table (F000:CF9F hex):
D1BA  C7 06 0104 CF9F R C       mov     word ptr ds:(4*41h),cs:(offset hdu_parm_tbl)
D1C0  8C 0E 0106        C       mov     word ptr ds:[(4*41h)+2],cs
                        C
                        C  ;install vector to diskette parameter table (F000:EFC7 hex):
D1C4  C7 06 0078 R EFC7 R C     mov     word ptr ds:[int1Elocn],cs:(offset fd_parms)
D1CA  8C 0E 007A R      C       mov     word ptr ds:[int1Elocn+2],cs
                        C
D1CE  FB                C       sti                     ;enable interrupts after critical code.
                        C
                        C       ; Do floppy setup.  Use "call" rather than Int13, since
                        C       ; this is a non-standard command, and should not be used in
                        C       ; cases where Int40 has been intercepted
                        C
                        C  ;mov ax,1800h ; "Setup" command
                        C  ;callnear ptr fd_io    ;call diskette bios.
                        C
                        C  ;Attempt to boot from diskette 0 up to 3 times.  A time-out error
                        C  ; immediately ends the attempt to boot from diskette.
                        C
D1CF  B9 0003           C       mov     cx,3            ;number of retry for diskette.
D1D2  33 D2             C       xor     dx,dx           ;set head zero and drive 0.
D1D4                    C  dett_boot:
D1D4  B8 0000           C       mov     ax,h_rst*256    ;RAH <= bios call no., RAL <= block count.
                        C  ;pushf                       ;push flags and ...
                        C  ;pushcs              ;... code segment to emulate an INT call.
                        C  ;    call    near ptr fd_io  ;call diskette bios.
D1D7  CD 40             C       int     40h             ; Use vector rather than hard-coded address
D1D9  72 0F             C       jc      dett_boot_nxt   ;jump if error
                        C
D1DB  BB 7C00           C       mov     bx,07c00h;Otherwise, read 1 sector & buffer it at 7C00.
D1DE  B8 0201           C       mov     ax,h_rd*256+1
D1E1  51                C       push    cx              ;save retry count.
D1E2  B9 0001           C       mov     cx,1            ;read from sector number 1.
                        C  ;pushf                       ;save flags and RCS for INT emulation:
                        C  ;pushcs
```

```
                        C  ;    call    near ptr fd_io    ;call diskette bios.
D1E5  CD 40             C       int     40h               ; Use vector rather than hard-coded address
D1E7  59                C       pop     cx                ;restore retry count.
                        C
D1E8  73 48             C       jnc     boot_succ;if no error then go and execute loaded code!
D1EA                    C  dett_boot_nxt:
D1EA  80 FC 80          C       cmp     ah,time_out       ;Did we time out?
D1ED  74 02             C       je      dett_boot_end     ;if so jump ...
D1EF  E2 E3             C       loop    dett_boot;otherwise try again.
                        C
D1F1                    C  dett_boot_end:
D1F1  B8 0000           C       mov     ax,h_rst*256      ;... and reset diskette:
                        C  ;pushf                         ;(flags and RCS are pushed for INT emulation).
                        C  ;pushcs
                        C  ;    call    near ptr fd_io    ;call diskette bios.
D1F4  CD 40             C       int     40h               ; Use vector rather than hard-coded address
                        C
                        C  ; Boot from the first possible winchester drive.  Only one attempt is made
                        C  ; to boot from each drive.  A valid boot sector must have AA55 in its last
                        C  ; word.
                        C
D1F6  B2 80             C       mov     dl,80h            ;start with first hard disk.
D1F8  1E                C       push    ds                ;save current RDS and RAX:
D1F9  50                C       push    ax
                        C       assume  ds:data
D1FA  B8 0040           C       mov     ax,data_seg       ;load " data_seg"  to point to hdu number.
D1FD  8E D8             C       mov     ds,ax
D1FF  80 3E 0075 R 00   C       cmp     hf_num,0 ;check for any functioning hdu's.
D204  74 2A             C       je      boot_basic        ;if none is available then boot BASIC.
                        C
D206  32 ED             C       xor     ch,ch             ;clear RCH.
D208  8A 0E 0075 R      C       mov     cl,hf_num;load RCL with total number of hdu's present.
                        C       assume  ds:nothing
D20C  58                C       pop     ax                ;restore previous RDS and RAX.
D20D  1F                C       pop     ds
                        C
D20E                    C  win_boot:
D20E  B4 00             C       mov     ah,h_rst ;" reset"  code.
D210  CD 13             C       int     13h               ; call disk BIOS to reset hdu.
D212  72 18             C       jc      win_boot_nxt      ;jump if error.
                        C
D214  BB 7C00           C       mov     bx,07c00h;Save into RAM 7C00hex ...
D217  B8 0201           C       mov     ax,h_rd*256+1     ;one sector read ...
D21A  51                C       push    cx                ;(but save drive count!)
D21B  B9 0001           C       mov     cx,1              ;starting from sector #1 ...
D21E  CD 13             C       int     13h               ;using a call to the disk BIOS.
D220  59                C       pop     cx                ;Restore drive count.
D221  72 09             C       jc      win_boot_nxt      ;jump if error.
                        C
D223  26: 81 3E 7DFE AA55 C     cmp     word ptr es:[07c00h+510d],0aa55h  ;check for valid boot sector.
D22A  74 06             C       je      boot_succ;jump if so.
                        C
D22C                    C  win_boot_nxt:
D22C  FE C2             C       inc     dl                ;otherwise increment drive number
D22E  E2 DE             C       loop    win_boot ;and retry as long as there are more hdu's.
```

```
                        C
D230                    C  boot_basic:
D230  CD 18             C       int     18h             ;Boot BASIC if we cannot boot from any disk or
                        C                               ;diskette.
                        C
D232                    C  boot_succ:
D232  2E: FF 2E D237 R  C       jmp     dword ptr cs:[boot_indirect]  ;jump to boot code.
D237                    C  boot_indirect:
D237  7C00              C       dw      07c00h          ;jump indirect to boot code at 0:7C00 hex.
D239  0000              C       dw      0000
                        C
                        C  h_boot       endp
                        C
                        C  ;=======================================
                        C  ;
                        C  ; Call:        int    0dh
                        C  ;             return
                        C  ;
                        C  ; Winchester interrupt handler
                        C  ;
                        C  ; Entry:   No entry parameters
                        C  ;
                        C  ; Exit:   No registers changed.
                        C
D23B                    C  h_intproc
D23B  50                C       push    ax              ;save RAX
D23C  B0 20             C       mov     al,pic_neoi     ;end of interrupt bit.
D23E  E6 20             C       out     pic_0,al ;end of interrupt to 8259.
D240  B0 07             C       mov     al,h_dma_s or h_dma_3     ;set chn 3 mask bit.
D242  E6 0A             C       out     dma_addr_0+dma_mask_bit,al ;set bit to disable chn 3.
                        C  ;
                        C  ;Note that the next three operations are necessary for the disk BIOS to work
                        C  ;correctly. In fact, command completion is checked on the IRQ5 mask bit set in
                        C  ;the PIC Mask Register:
D244  E4 21             C       in      al,pic_1 ;read interrupt mask.
D246  0C 20             C       or      al,h_ocw1_m5      ;set mask register bit to disable ...
D248  E6 21             C       out     pic_1,al ;... chn 5 now used by the hdu controller.
                        C
D24A  58                C       pop     ax              ;restore RAX.
D24B  CF                C       iret                    ;reurn from INT (long ret.) & restore flags.
                        C
                        C  h_intendp
                        C
                        C  ;=======================================
                        C  ;
                        C  ; Call:       int      13h
                        C  ;             return
                        C  ;
                        C  ; Function call to disk BIOS.
                        C  ;
                        C  ; Entry:        (AH) = Command code
                        C  ;    (AL) = Sector count or interleave factor
                        C  ;    (ES:BX) = Address of buffer
                        C  ;    (CL) bits 0 - 5 = Sector number: 1, 2, 3, . . .
                        C  ;    (CL) bits 6, 7 = Bits 8, 9 of cylinder number
```

```
                        C  ;    (CH) = Bits 0 - 7 of cylinder number
                        C  ;    (DL) = Drive number.  0 - 7f are diskettes. 80h - 87h up are winchesters
                        C  ;    (DH) = Head number
                        C  ;
                        C  ; Exit:  (CF) = 0 --> no error
                        C  ;            1 --> error
                        C  ;     (AH) = Completion code.  0 --> no error
                        C  ;    (AL) = Error burst length if (AH) = 11h.  Else, changed.
                        C  ;
                        C  ; Read parameters:
                        C  ;    (DL) = Number of functioning drives at installation time
                        C  ;    (DH) = Maximum head number
                        C  ;    (CL) Bits 0 - 5 = 17d --- Maximum sector number
                        C  ;    (CL) bits 6, 7 = Bits 8, 9 of Maximum cylinder number
                        C  ;    (CH) = Bits 0 - 7 of maximum cylinder number
                        C
D24C                    C  h_io proc     far
                        C       assume   ds:nothing,es:nothing
                        C
D24C  FB                C       sti                        ;enable interrupts.
D24D  80 FA 80          C       cmp      dl,80h            ;check for hdu calling.
D250  73 05             C       jae      win_bc            ;jump if so.
                        C
                        C  ;    jmp      near ptr fd_io    ;otherwise pass control to the diskette BIOS.
D252  CD 40             C       int      40h               ; Use vector to call floppy driver
D254  CA 0002           C       ret      2                 ; All finished, ret to caller
                        C
D257                    C  win_bc:
D257  80 FC 00          C       cmp      ah,h_rst ;is it a reset command?
D25A  75 0E             C       jne      no_reset ;jump if not.
                        C
D25C  52                C       push     dx                ;save hdu number and ...
                        C  ;pushf                          ;call diskette BIOS to reset FDU's as well.
                        C  ;pushcs
                        C  ;    call     near ptr fd_io
D25D  CD 40             C       int      40h               ; Use vector to issue reset to floppy
D25F  5A                C       pop      dx                ;restore hdu number.
D260  80 FA 87          C       cmp      dl,87h            ;check for invalid hdu number.
D263  76 03             C       jbe      win_bc_1 ;jump if valid.
D265  CA 0002           C       ret      2                 ;otherwise return without flags.
956                        C
D268                    C  win_bc_1:
D268  B4 00             C       mov      ah,h_rst ;restore command code.
                        C
D26A                    C  no_reset:
D26A  53                C       push     bx                ;save all registers to be used by "H_IO".
D26B  51                C       push     cx
D26C  52                C       push     dx
D26D  55                C       push     bp
D26E  57                C       push     di
D26F  56                C       push     si
D270  1E                C       push     ds
D271  06                C       push     es
D272  8B EC             C       mov      bp,sp             ;initialize activation record pointer.
                        C
```

```
D274  50                C       push    ax              ;temporarily save RAX for next operation.
                        C
                        C       assume  ds:data
D275  B8 0040           C       mov     ax,data_seg
D278  8E D8             C       mov     ds,ax           ;load RDS with " data_seg"  and restore RAX:
D27A  58                C       pop     ax
                        C
D27B  80 FC 14          C       cmp     ah,h_dg_c;command code in range?
D27E  77 05             C       ja      bc_bad          ;jump if out of range.
D280  80 FA 88          C       cmp     dl,80h+8 ;hdu number in range?
D283  72 04             C       jb      drvn_ok         ;jump if in range.
                        C
D285                    C bc_bad:
D285  B4 01             C       mov     ah,cmd_error    ;command error completion code.
D287  EB 03             C       jmp     short cmd_done  ;Exit.
                        C
D289                    C drvn_ok:
D289  E8 D2B3 R         C       call    command_br      ;send out command following proper format and
                        C                               ;protocol.
                        C
D28C                    C cmd_done:
D28C  50                C       push    ax              ;save output parameter and ...
D28D  88 26 0074 R      C       mov     disk_status,ah  ;completion code.
                        C
D291  8D 94 0323        C       lea     dx,p_wx2+hd_msk[si] ;calculate port address to ...
D295  B0 FC             C       mov     al,not (h_mkdma or h_mkint) ;disable DMAReq and INTreq by
                        C                                   ;controller.
D297  EE                C       out     dx,al           ;disable them.
D298  B0 07             C       mov     al,h_dma_3 or h_dma_s
D29A  E6 0A             C       out     dma_addr_0+dma_mask_bit,al ;set bit 3 to disable chn 3.
D29C  FA                C       cli                     ;disable interrupts.
                        C
D29D  E4 21             C       in      al,pic_1 ;read mask register.
D29F  0C 20             C       or      al,h_ocw1_m5    ;set bit 5 to disable chn 3.
D2A1  E6 21             C       out     pic_1,al ;set mask register.
D2A3  FB                C       sti                     ;enable interrupts.
                        C
D2A4  80 C4 FF          C       add     ah,0ffh         ;set Carry flag to indicate (no) error.
D2A7  58                C       pop     ax              ;restore all registers previously saved.
D2A8  07                C       pop     es
D2A9  1F                C       pop     ds
D2AA  5E                C       pop     si
D2AB  5F                C       pop     di
D2AC  5D                C       pop     bp
D2AD  5A                C       pop     dx
D2AE  59                C       pop     cx
D2AF  5B                C       pop     bx
D2B0  CA 0002           C       ret     2               ;exit without flags.
                        C h_io endp
                        C
                        C ;=======================================
                        C ;
                        C ; Call:         CALL   command_br
                        C ;             return
                        C ;
```

```
                        C  ; Branch through the bios command table
                        C  ;
                        C  ; Entry:  Same as h_io.
                        C  ;    (BP) = Pointer to activation record described by symbols a1_XXXX
                        C  ;
                        C  ; Jump to command specific routine:
                        C  ;    (DS) = data_seg
                        C  ;    (BP) = Pointer to activation record described by symbols a1_XXXX
                        C  ;    (si) = offset to port base for target controller
                        C  ;
                        C  ; Exit:   (AH) = Completion code
                        C  ;    (DS) = data_seg
                        C
D2B3                    C  command_br    proc     near
                        C
                        C  ;Set up the CDB:
                        C
D2B3  A2 0046 R         C       mov      cmd_block+ccb_blks,al      ;sector count or interleave factor.
D2B6  FE C9             C       dec      cl                ;controller expects sectors numbered 0 thru
                        C                                  ;16!
D2B8  89 0E 0044 R      C       mov      word ptr cmd_block+2,cx    ;store cyl. and sect. # into CDB.
D2BC  8A EA             C       mov      ch,dl             ;drive number
D2BE  80 E5 01          C       and      ch,01             ;mask to bit 0.
D2C1  B1 05             C       mov      cl,5              ;shift count for proper ...
D2C3  D2 E5             C       shl      ch,cl             ;CDB drive bit alignement.
D2C5  80 E6 0F          C       and      dh,0fh            ;mask off garbage if any (from commands not
                        C                                  ;required to specify head #).
D2C8  0A EE             C       or       ch,dh             ;combine with head number.
D2CA  88 2E 0043 R      C       mov      cmd_block+1,ch    ;Store it into CDB (as 2nd byte).
                        C
                        C                        ;Calculate the offset from p_wx2 for drive:
                        C                        ; Drive #      Offset
                        C                        ; 80, 81  0
                        C                        ; 82, 83  4
                        C                        ; 84, 85  8
                        C                        ; 86, 87  12d
D2CE  80 EA 80          C       sub      dl,80h            ;origin to zero.
D2D1  81 E2 00FE        C       and      dx,0FEh
D2D5  D0 E2             C       shl      dl,1
D2D7  8B F2             C       mov      si,dx             ;save controller port offset into [SI] reg.
D2D9  88 16 0077 R      C       mov      port_off,dl       ;and loc. "port_off" (40:77)hex.
D2DD  50                C       push     ax                ;save registers ...
D2DE  06                C       push     es
                        C
D2DF  E8 D5B2 R         C       call     subtable ;determine subtable to use (type #).
D2E2  26: 8A 47 08      C       mov      al,es:wst_opt[bx] ;fetch option byte for proper step rate
D2E6  A2 0047 R         C       mov      cmd_block+ccb_opt,al ; and store it into CDB and into
D2E9  A2 0076 R         C       mov      control_byte,al ;proper RAM location (40:76)hex.
                        C
D2EC  07                C       pop      es                ;restore buffer address and
D2ED  58                C       pop      ax                ;command code.
                        C
D2EE  8A C4             C       mov      al,ah             ;BIOS command code.
D2F0  BB D32D R         C       mov      bx,offset dc_tbl  ;pointer to controller command code
                        C      ;table.
```

```
D2F3  2E: D7              C       xlat    cs:dc_tbl;translate bios command code into ST506 code.
D2F5  A2 0042 R           C       mov     cmd_block+ccb_cmd,al        ;Command code into CDB.
                          C
D2F8  8A DC               C       mov     bl,ah               ;BIOS command code into RBL
D2FA  32 FF               C       xor     bh,bh               ;zero high byte
D2FC  D0 E3               C       sal     bl,1                ;multiply by two and ...
D2FE  2E: FF A7 D303 R    C       jmp     cs:br_tbl[bx]       ;... branch to command specific routine.
                          C
D303                      C br_tbl        label   word
D303  D342 R              C       dw      i13_reset
D305  D3F2 R              C       dw      i13_cc
D307  D412 R              C       dw      i13_rd
D309  D3F7 R              C       dw      i13_wr
D30B  D469 R              C       dw      dma_no              ;4 --- Verify.
D30D  D469 R              C       dw      dma_no              ;5 --- Format track
D30F  D469 R              C       dw      dma_no              ;6 --- Format Bad Track
D311  D469 R              C       dw      dma_no              ;7 --- Format Drive
D313  D3CB R              C       dw      i13_par_rd
D315  D365 R              C       dw      i13_par_wr
D317  D3FB R              C       dw      i13_rdl
D319  D404 R              C       dw      i13_wrl
D31B  D469 R              C       dw      dma_no              ;12 --- Seek
D31D  D342 R              C       dw      i13_reset;13 --- Same as 0
D31F  D408 R              C       dw      i13_buff_rd
D321  D40E R              C       dw      i13_buff_wr
D323  D469 R              C       dw      dma_no              ;16 --- Test Drive Ready
D325  D469 R              C       dw      dma_no              ;17 --- Recalibrate
D327  D469 R              C       dw      dma_no              ;18 --- Internal RAM diagnostic
D329  D469 R              C       dw      dma_no              ;19 --- Drive Diagnostic
D32B  D469 R              C       dw      dma_no              ;20 --- Controller Diagnostic
                          C
D32D                      C dc_tbl        label   byte
D32D  0C                  C       db      hc_p_s              ;all resets also set parameters
D32E  00                  C       db      0                   ;return completion code
D32F  08                  C       db      hc_rd
D330  0A                  C       db      hc_wr
D331  05                  C       db      hc_vr
D332  06                  C       db      hc_ft
D333  07                  C       db      hc_fbt
D334  04                  C       db      hc_fd
D335  00                  C       db      0
D336  0C                  C       db      hc_p_s              ;read parameters
D337  E5                  C       db      hc_rdl
D338  E6                  C       db      hc_wrl
D339  0B                  C       db      hc_sk
D33A  0C                  C       db      hc_p_s              ;all resets also set parameters
D33B  0E                  C       db      hc_bf_r
D33C  0F                  C       db      hc_bf_w
D33D  00                  C       db      hc_rdy
D33E  01                  C       db      hc_rcl
D33F  E0                  C       db      hc_dg_r
D340  E3                  C       db      hc_dg_d
D341  E4                  C       db      hc_dg_c
                          C
                          C ;--------
```

```
                        C  ; Call:         call   i13_reset
                        C  ;              return
                        C  ;
                        C  ; Purpose:  Reset the target winchester controller.
                        C  ;
                        C  ; Entry:    (si) = Offset to port base for target controller
                        C  ;
                        C  ; Exit:     (ah) = Completion code
                        C
D342                    C  i13_reset:
D342  8D 94 0321        C       lea     dx,p_wx2+hd_rs[si]
D346  EE                C       out     dx,al
                        C  ;This is a 110msec delay loop to accomodate completion of controller diagnostic
                        C  ; ( first release WD1002 - 0X2) issued upon reset command.
D347  B9 FFFF           C       mov     cx,0ffffh;17 cycles @ 100nsec each x 64K = 110msec.
D34A                    C  loop_wait:
D34A  E2 FE             C       loop    loop_wait;wait 110msec.
                        C
D34C  B4 19             C       mov     ah,25           ;msb of selection loop count.
D34E                    C  re_w:
D34E  42                C       inc     dx              ;control port
D34F  EE                C       out     dx,al           ;select controller
D350  4A                C       dec     dx              ;back to the status port
D351  EC                C  re_w_1:      in      al,dx           ;read status
D352  A8 30             C       test    al,30h          ;if Intrq or DMAreq pending then controller
                        C                       ;is missing (lines are being pulled up by the bus)
D354  75 0C             C       jnz     ctlr_missing
D356  24 0F             C       and     al,0fh          ;mask off most significant (ms) nibble.
D358  3C 0D             C       cmp     al,0dh          ;check for select status (0Dh)
D35A  74 09             C       jz      i13_par_wr      ;is selected continue
D35C  E2 F3             C       loop    re_w_1          ;otherwise test again ...
D35E  FE CC             C       dec     ah
D360  75 EF             C       jnz     re_w_1          ; ... until time-out.
D362                    C  ctlr_missing:
D362  B4 05             C       mov     ah,h_reset
D364                    C  ret_near:
D364  C3                C       ret                     ;exit.
                        C
                        C  ;------
                        C  ; Call:         call   i13_par_wr
                        C  ;              return
                        C  ;
                        C  ;
                        C  ; Entry:    (si) = Offset to port base for target controller
                        C  ;
                        C  ; Exit:     (ah) = Completion code
                        C  ;
                        C
D365                    C  i13_par_wr:
D365  C6 06 0043 R 00   C       mov     cmd_block+1,0   ;drive 0 is first target
D36A  E8 D37C R         C       call    par_wr          ;set parameters for first drive
D36D  72 F5             C       jc      ret_near ;jump if error
                        C
D36F  C6 06 0043 R 20   C       mov     cmd_block+1,ccb_drv_b   ;update ctlr drive count (LUN)
D374  E8 D37C R         C       call    par_wr          ;set paramters for second drive
```

```
D377  72 EB              C       jc      ret_near ;jump if error
                         C
D379  EB 42 90           C       jmp     ret_near_1      ;othrewise exit with no error on command.
                         C
D37C                     C  par_wr:
D37C  B0 FC              C       mov     al,not(h_mkdma or h_mkint) ;disable ctlr's Intrq and DMAreq.
D37E  E8 D54A R          C       call    ccb_send ;send CDB to target controller
D381  72 E1              C       jc      ret_near ;jump if error
                         C
D383  E8 D5B2 R          C       call    subtable ;determine hdu type (es:bx <= type table offset)
D386  BF 0001            C       mov     di,wst_cyl+1    ;offset to byte to send:
D389  E8 D3BE R          C       call    send_byte;send msb of cylinders
                         C
D38C  BF 0000            C       mov     di,wst_cyl+0    ;LSB of number of Cylinders
D38F  E8 D3BE R          C       call    send_byte
                         C
D392  BF 0002            C       mov     di,wst_heads    ;number of heads
D395  E8 D3BE R          C       call    send_byte
                         C
D398  BF 0004            C       mov     di,wst_re_wr+1  ;MSB of reduced write current cylinder
D39B  E8 D3BE R          C       call    send_byte
                         C
D39E  BF 0003            C       mov     di,wst_re_wr+0  ;LSB of reduced write current cylinder
D3A1  E8 D3BE R          C       call    send_byte
                         C
D3A4  BF 0006            C       mov     di,wst_wr_pre+1 ;MSB of write precomp. cylinder
D3A7  E8 D3BE R          C       call    send_byte
                         C
D3AA  BF 0005            C       mov     di,wst_wr_pre+0 ;LSB of write precomp. cylinder
D3AD  E8 D3BE R          C       call    send_byte
                         C
D3B0  BF 0007            C       mov     di,wst_er_bur   ;maximum burst length
D3B3  E8 D3BE R          C       call    send_byte
                         C
D3B6  E8 D583 R          C       call    wx2_cc          ;fetch command completion byte
D3B9  73 02              C       jnc     ret_near_1      ;jump if no error
                         C
D3BB                     C  par_wr_erx:
D3BB  B4 07              C       mov     ah,h_initz      ;if error prevented reception then set error
                         C                               ;code and ...
D3BD                     C  ret_near_1:
D3BD  C3                 C       ret                     ;... exit.
                         C
D3BE                     C  send_byte:
D3BE  E8 D5A0 R          C       call    wx2_req         ;check for ready status
D3C1  72 05              C       jc      send_err ;jump if error
D3C3  26: 8A 01          C       mov     al,es:[bx+di]   ;byte to send from drive table
D3C6  EE                 C       out     dx,al           ;... to controller.
D3C7  C3                 C       ret                     ;exit.
                         C
D3C8                     C  send_err:
D3C8  58                 C       pop     ax              ;throw away near return and ...
D3C9  EB F0              C       jmp     par_wr_erx      ;error exit from assign parameters.
                         C
                         C  ;=======================================
```

```
                        C  ;
                        C  ; Call:         call   i13_par_rd
                        C  ;             return
                        C  ;
                        C  ; Read parameters
                        C  ;
                        C  ; Entry:        (si) = offset to port base for target controller
                        C  ;
                        C  ; Exit: (DL) = Number of usable drives at install time
                        C  ;    (DH) = Maximum head number
                        C  ;    (CL) Bits 0 - 5 = 17d --- Maximum sector number
                        C  ;    (CL) bits 6, 7 = Bits 8, 9 of Maximum cylinder number
                        C  ;    (CH) = Bits 0 - 7 of maximum cylinder number
                        C
D3CB                    C  i13_par_rd:
D3CB  E8 D5B2 R         C       call    subtable ;determine drive type
D3CE  26: 8B 07         C       mov     ax,es:wst_cyl[bx] ;number of cylinders
D3D1  2D 0002           C       sub     ax,2              ;maximum cylinder number (last one reserved).
D3D4  8A E8             C       mov     ch,al             ;temp. save
D3D6  D1 E8             C       shr     ax,1              ;align bits 8 & 9 of cylinder #
D3D8  D1 E8             C       shr     ax,1
D3DA  24 C0             C       and     al,11000000b      ;mask to bits 8 & 9 of cylinder #
D3DC  0C 11             C       or      al,17             ;maximum sector number
D3DE  8A E5             C       mov     ah,ch             ;bits 0 thru 7 of cylinder # ...
D3E0  89 46 0C          C       mov     a1_cx[bp],ax      ;set into activation record
D3E3  26: 8A 67 02      C       mov     ah,es:wst_heads[bx] ;number of heads
D3E7  FE CC             C       dec     ah                ;maximum head number
D3E9  A0 0075 R         C       mov     al,hf_num;number of operational drives
D3EC  89 46 0A          C       mov     a1_dx[bp],ax      ;set into activation record
D3EF                    C  ret_no_err:
D3EF  B4 00             C       mov     ah,h_no_err       ;no error
D3F1  C3                C       ret                       ;exit.
                        C
                        C  ;=====================================
                        C  ;
                        C  ; Call:         call   i13_cc
                        C  ;             return
                        C  ;
                        C  ; Return the completion code for the previous command to any winchester drive.
                        C  ;
                        C  ; Entry:  (ram_cc) = Completion code for last command to any winchester
                        C  ;
                        C  ; Exit:   (al) = completion code for last command
                        C
D3F2                    C  i13_cc:
D3F2  A0 0074 R         C       mov     al,disk_status    ;last command completion code
D3F5  EB F8             C       jmp     ret_no_err        ;no error exit
                        C
                        C  ;=====================================
                        C
                        C  ; Call:         call   i13_wr
                        C  ;             return
                        C  ;
                        C  ; Purpose:  Write sectors
                        C  ;
```

```
                        C  ; Entry:   (si) = Offset to port base for target controller
                        C  ;
                        C  ; Exit:    (ah) = Completion code
                        C
D3F7                    C  i13_wr:
D3F7  B0 4B             C       mov     al,h_dma_r        ; byte to set DMA chip to read from host.
D3F9  EB 19             C       jmp     short io_norm
                        C
                        C  ;=======================================
                        C
                        C  ; Call:        call   i13_rdl
                        C  ;            return
                        C  ;
                        C  ; Purpose:  Read sector and ECC bytes
                        C  ;
                        C  ; Entry:   (si) = Offset to port base for target controller
                        C  ;
                        C  ; Exit:    (ah) = Completion code
                        C
D3FB                    C  i13_rdl:
D3FB  B0 47             C       mov     al,h_dma_w        ;byte to set DMA chip to write to host.
D3FD                    C  io_long:
D3FD  B2 01             C       mov     dl,1              ;DTC controller can only rd/wr long
                        C                                 ;1 sector!
D3FF  BF 0204           C       mov     di,sec_size+4     ;number of bytes/sector plus 4 ECC bytes.
D402  EB 17             C       jmp     short dma_start
                        C
                        C  ;=======================================
                        C  ; Call:        call   i13_wrl
                        C  ;            return
                        C  ;
                        C  ; Purpose:  Write sector and ECC bytes
                        C  ;
                        C  ; Entry:   (si) = Offset to port base for target controller
                        C  ;
                        C  ; Exit:    (ah) = Completion code
                        C
D404                    C  i13_wrl:
D404  B0 4B             C       mov     al,h_dma_r        ;byte to set DMA chip to read to host
D406  EB F5             C       jmp     short io_long
                        C
                        C  ;=======================================
                        C  ; Call:        call   i13_buff_rd
                        C  ;            return
                        C  ;
                        C  ; Purpose:  Read sector buffer
                        C  ;
                        C  ; Entry:   (si) = Offset to port base for target controller
                        C  ;
                        C  ; Exit:    (ah) = Completion code
                        C
D408                    C  i13_buff_rd:
D408  B0 47             C       mov     al,h_dma_w        ;set DMA chip to write to host.
                        C
D40A                    C  buff_io:
```

```
D40A  B2 01              C       mov     dl,1
D40C  EB 0A              C       jmp     short sec_size_norm
                         C
                         C  ;=====================================
D40E                     C  i13_buff_wr:
D40E  B0 4B              C       mov     al,h_dma_r
D410  EB F8              C       jmp     short buff_io
                         C
                         C  ;=====================================
                         C  ; Call:        call   i13_rd
                         C  ;             return
                         C  ;
                         C  ; Purpose:  Read sectors
                         C  ;
                         C  ; Entry:   (si) = Offset to port base for target controller
                         C  ;
                         C  ; Exit:    (ah) = Completion code
                         C
D412                     C  i13_rd:
D412  B0 47              C       mov     al,h_dma_w          ;set DMA chip to write to host.
D414                     C  io_norm:
D414  8A 16 0046 R       C       mov     dl,cmd_block+ccb_blks ;number of blocks to transfer
D418                     C  sec_size_norm:
D418  BF 0200            C       mov     di,sec_size         ;number of bytes/sector.
                         C
                         C  ;Prepare the DMA chip:
                         C
D41B                     C  dma_start:
D41B  FA                 C       cli                         ;disable interrupts
D41C  E6 0B              C       out     dma_addr_0+dma_mode,al      ;set read or write to host
D41E  E6 0C              C       out     dma_addr_0+dma_ff_clr,al ;clear pointer to register byte.
                         C
                         C  ;Calculate the 20 bit address of the DMA base address:
                         C
D420  8C C0              C       mov     ax,es               ;segm. reg. for I/O buffer
D422  B1 04              C       mov     cl,4                ;count for shift
D424  D3 C0              C       rol     ax,cl               ;multiply by 16
D426  8A E8              C       mov     ch,al               ;bit 16 -- 19 in bits 0 - 3
D428  24 F0              C       and     al,0f0h             ;clear bits 0 - 3 of address
D42A  03 46 0E           C       add     ax,a1_bx[bp]        ;combine 16-bit offset with seg. base
D42D  80 D5 00           C       adc     ch,0                ;add carry into MS bits
D430  E6 06              C       out     dma_addr_0+dma_addr_3,al ;send low address
D432  86 E0              C       xchg    ah,al               ;bit 8 - 15 into RAL
D434  E6 06              C       out     dma_addr_0+dma_addr_3,al ;send high address
D436  86 C5              C       xchg    al,ch               ;bits 16 - 19 of DMA address into RAL
D438  8A CC              C       mov     cl,ah               ;bits 0 - 15 od=f DMA address now in RCX
D43A  24 0F              C       and     al,0fh              ;mask to 4 bits
D43C  E6 82              C       out     dma_segm_1,al       ;set 4 bit latch for highest address bits
                         C
                         C  ;Compute number of bytes to be transferred minus 1:
D43E  8B C7              C       mov     ax,di               ;block size in bytes
D440  32 F6              C       xor     dh,dh               ;clear upper byte
D442  F7 E2              C       mul     dx                  ;number of bytes to DMA: (RAX) LSW
                         C                                   ;(RDX) MSW
D444  2D 0001            C       sub     ax,1                ;byte offset to last byte to be transferred
```

```
D447  80 DA 00            C       sbb     dl,0                    ;bits 16 - 23 of byte offset
                          C                                       ;and set ZF.
D44A  E6 07               C       out     dma_addr_0+dma_count_3,al ;set bits 0 - 7 of byte count
D44C  86 C4               C       xchg    al,ah                   ;bits 8 - -15 of count to RAL
D44E  E6 07               C       out     dma_addr_0+dma_count_3,al ;set bits 8 - 15 of byte count
                          C
D450  FB                  C       sti                             ;enable interrupts
                          C
                          C     ; Check for DMA spanning 64k absolute address boundary
                          C                                               ;Are bits 16 - 23 of byte offset 0?
                          C                                               ;  note:  If the block count is 0,
                          C                                               ;       then (dl) = ff.
D451  75 13               C       jnz     dma_64k                 ;jump if DMA more than 64kbytes.
D453  86 E0               C       xchg    ah,al                   ;byte offset to last byte transferred
D455  03 C1               C       add     ax,cx                   ;lsw of address of last byte transferred
D457  72 0D               C       jc      dma_64k                 ;jump if DMA crosses 64k boundary
                          C
                          C  ;Start hard disk controller and DMA controller:
D459  B0 03               C       mov     al,h_mkdma or h_mkint       ;DMAreq & INTreq enabled
D45B  E8 D54A R           C       call    ccb_send ;send CDB to controller
D45E  72 08               C       jc      ret_near_3              ;jump if error
                          C
D460  B0 03               C       mov     al,h_dma_3              ;clear bit of DMA mask register
D462  E6 0A               C       out     dma_addr_0+dma_mask_bit,al ;enable chn 3
D464  EB 0A               C       jmp     short wx2_wait
                          C
D466                      C  dma_64k:
D466  B4 09               C       mov     ah,dma_seg_error ;completion error code
D468                      C  ret_near_3:
D468  C3                  C       ret                             ;exit
                          C
                          C  ;=======================================
                          C  ; Call:         call   i13_dma_no
                          C  ;             return
                          C  ;
                          C  ; Purpose:  Execute a command involving the winchester controller
                          C  ;        but not involving DMA.
                          C  ;
                          C  ; Entry:    (si) = Offset to port base for target controller
                          C  ;
                          C  ; Exit:     (ah) = Completion code
                          C
D469                      C  dma_no:
D469  B0 02               C       mov     al,h_mkint              ;enable interrupt by controller
D46B  E8 D54A R           C       call    ccb_send ;send CDB to controller
D46E  72 F8               C       jc      ret_near_3              ;jump if error
                          C
                          C  ;Wait for the controller interrupt in a polling loop:
D470                      C  wx2_wait:
D470  FA                  C       cli                             ;disable interrupts
D471  E4 21               C       in      al,pic_1 ;read interrupt mask
D473  24 DF               C       and     al,not h_ocw1_m5 ;enable chn 5 interrupts
D475  E6 21               C       out     pic_1,al ;set interrupt mask
D477  FB                  C       sti                             ;enable interrupts
                          C
```

```
D478  56                C       push    si              ;save controller port offset
D479  33 C0             C       xor     ax,ax           ;zero RAX
D47B  E8 D5B2 R         C       call    subtable ;compute type of requesting drive
D47E  32 E4             C       xor     ah,ah           ;zero RAH
D480  26: 8A 47 09      C       mov     al,es:[bx].p_timeout ;fetch regular time-out
D484  80 3E 0042 R 04   C       cmp     byte ptr ds:[cmd_block+0],hc_fd ;Is it a format drive command?
D489  75 0C             C       jne     w1              ; jump if not
D48B  26: 8A 47 0A      C       mov     al,es:[bx].p_fmt_timeout ; otherwise fetch format timeout
D48F  3C FF             C       cmp     al,255          ; Format timeout value at maximum ?
D491  72 0F             C       jb      w2              ; if < max it's OK; otherwise...
D493  D1 E0             C       shl     ax,1            ;   double format timeout parameter value
D495  EB 0B             C       jmp short w2            ;    and then go multiply the value by 16
D497                    C  w1:
D497  80 3E 0042 R E3   C       cmp     byte ptr ds:[cmd_block+0],hc_dg_d ;Is it a diagnostic command?
D49C  75 04             C       jne     w2              ;jump if not
D49E  26: 8A 47 0B      C       mov     al,es:[bx].p_drvdiag_timeout ;if so fetch diagnostic time-out.
D4A2                    C  w2:
D4A2  B1 04             C       mov     cl,4
D4A4  D3 E0             C       shl     ax,cl           ;multiply time-out value by 16:
                        C
                        C  ;Time out computation:
                        C  ;    base value x 2 due to increased processing speed (4.7 vs 10)Mhz
                        C  ;          "        x 8 due to slower formatting routine of some controller
                        C  ;
D4A6  8B D8             C       mov     bx,ax           ;time out value ==> RBX
D4A8  33 C0             C       xor     ax,ax           ;zero RAX
D4AA  B9 0000           C       mov     cx,0            ;prepare RCX for LOOP count
D4AD                    C  wait_for_irq5:
D4AD  5E                C       pop     si              ;restore port offset
D4AE  8D 94 0321        C       lea     dx,p_wx2+hd_st[si] ;set RDX ==> controller status port
                        C
                        C  ;In order to minimize bus usage while data is been transferred thru DMA,
                        C  ; command completion is checked  by sensing the PIC interrupt line. If an
                        C  ; interrupt is sensed then the controller Intrq line is sensed as well to
                        C  ; ensure that the interrupt indeed was originated by it and not other devices.
                        C  ; If the interrupt was not originated by the controller then the PIC line
                        C  ; is sensed again until originated by the controller.
                        C  ;    Sensing the PIC instead of the controller port largely diminishes CPU
                        C  ; bus usage (code execution while waiting for command completion) thus allowing
                        C  ; the DMA to gain more control of the bus and increase data transfer rate.
                        C
D4B2                    C  int_wait:
D4B2  E4 21             C       in      al,pic_1 ;get PIC status port and ...
D4B4  A8 20             C       test    al,h_intp; ... sense for int. mask set by interrupt
                        C                               ;service routine (h_int).
D4B6  75 08             C       jnz     wx2_int         ;if int. mask 1 (i.e. hdu interrupt
                        C                               ;requested and serviced).
D4B8  E2 F8             C       loop    int_wait ;otherwise keep on reading the PIC
D4BA  4B                C       dec     bx              ; until time-out occurs:
D4BB  75 F5             C       jnz     int_wait
D4BD                    C  ret_time_j:
D4BD  E9 D569 R         C       jmp     ret_time ;if timed-out exit with error flag.
                        C
                        C  ;wx2_int_0:
                        C  ;    in      al,dx           ;get controller status.
```

```
                        C  ;    test    al,h_intp;check for interrupt pending
                        C  ;    jnz     wx2_int         ;if so then exit loop.
                        C  ;    jmp     int_wait ;if not a controller interrupts, test PIC again.
                        C
D4C0                    C  wx2_int:
D4C0  B9 0000           C       mov     cx,0            ;set RCX for LOOP
D4C3                    C  wx2_int_2:
                        C  ;    in      al,dx           ;get controller status.
                        C  ;and al,0fh          ;mask off the high nibble and ...
                        C  ;cmp al,0fh          ; ... check command completion byte (0Fh).
                        C  ;jz  wx2_int_1;If 0Fh then exit;
                        C  ;loopwx2_int_2;otherwise test again til status reported is 0Fh
                        C  ;jmp ret_time ;if NEVER 0Fh then the controller is bad.
                        C
D4C3                    C  wx2_int_1:
D4C3  8D 94 0323        C       lea     dx,p_wx2+hd_msk[si] ;disable ctlr's DMAreq and Intreq.
D4C7  B0 FC             C       mov     al,not (h_mkdma or h_mkint)
D4C9  EE                C       out     dx,al
D4CA  E8 D583 R         C       call    wx2_cc          ;fetch command completion byte
D4CD  72 EE             C       jc      ret_time_j      ;jump if error prevented cc reception
D4CF  74 97             C       jz      ret_near_3      ;jump if no error
                        C
                        C    ; The controller reported an error.  Read status to obtain more error
                        C    ; information.
                        C
D4D1  C6 06 0042 R 03   C       mov     cmd_block+ccb_cmd.hc_stat ;read status command
D4D6  B0 FC             C       mov     al,not (h_mkdma or h_mkint) ;no DMA or Int's
D4D8  E8 D54A R         C       call    ccb_send ;send command to controller
D4DB  72 6A             C       jc      stat_err ;jump if error
                        C
                        C  ; Read the 4 bytes of error status from the controller:
D4DD  BF 0042 R         C       mov     di,offset hd_error;offset to 4 byte area for status
D4E0  8C D8             C       mov     ax,ds                   ;data_seg
D4E2  8E C0             C       mov     es,ax                   ;set RES for "stosb" instruction
D4E4  B9 0004           C       mov     cx,4                    ;number of status bytes
D4E7  FC                C       cld                     ;clear direction ---> inc di
                        C
D4E8                    C  stat_loop:
D4E8  E8 D5A0 R         C       call    wx2_req         ;wait for request
D4EB  72 5A             C       jc      stat_err ;jump if error
D4ED  EC                C       in      al,dx           ;read statsu byte
D4EE  AA                C       stosb                   ;save status byte
D4EF  E2 F7             C       loop    stat_loop;jump if more status to read
                        C
D4F1  E8 D583 R         C       call    wx2_cc          ;receive command completion byte
D4F4  72 51             C       jc      stat_err ;jump if error prevented reception
D4F6  75 4F             C       jnz     stat_err ;jump if error on command
                        C
                        C  ; Translate the controller error code into a BIOS error code
D4F8  8A 2E 0042 R      C       mov     ch,hd_error     ;error code
D4FC  8A DD             C       mov     bl,ch
D4FE  81 E3 0030        C       and     bx,30h          ;mask to error type field
D502  B1 03             C       mov     cl,3            ;shift count
D504  D2 EB             C       shr     bl,cl           ;type times 2
D506  8A E5             C       mov     ah,ch           ;error code
```

```
D508  80 E4 0F            C       and     ah,0fh              ;mask to error in type field
D50B  2E: 3A A7 D5DF R    C       cmp     ah,cs:[bx]+offset er_master_tbl ;number of error code in type
D510  73 32               C       jnc     undef               ;jump if undefined
D512  43                  C       inc     bx                  ;update table address
D513  2E: 8A 9F D5DF R    C       mov     bl,cs:[bx]+offset er_master_tbl ;offset to subtable
D518  02 DC               C       add     bl,ah               ;error code subfield
D51A  2E: 8A A7 D5DF R    C       mov     ah,cs:[bx]+offset er_master_tbl ;completion code
D51F  80 FD 18            C       cmp     ch,h_erc ;correctable ECC error?
D522  75 22               C       jne     ret_near_5          ;jump if not.
                          C
                          C ; ECC error corrected ---> read the error burst length
                          C
D524  8A FC               C       mov     bh,ah               ;save completion code
D526  C6 06 0042 R 0D     C       mov     cmd_block+ccb_cmd,hc_ecc ;read error burst length
D52B  B0 FC               C       mov     al,not (h_mkdma or h_mkint) ;no DMA or Intreq from controller
D52D  E8 D54A R           C       call    ccb_send ;send command to controller.
D530  72 15               C       jc      stat_err ;jump if error
D532  E8 D5A0 R           C       call    wx2_req             ;wait for request
D535  72 10               C       jc      stat_err ;jump if error
D537  EC                  C       in      al,dx               ;read error burst length
D538  8A D8               C       mov     bl,al               ;save
D53A  E8 D583 R           C       call    wx2_cc              ;receive completion byte
D53D  72 08               C       jc      stat_err ;jump if error prevented reception
D53F  75 06               C       jnz     stat_err ;jump if error on command
D541  8B C3               C       mov     ax,bx               ;completion code and error burst length
D543  C3                  C       ret                         ;exit
                          C
D544                      C undef:
D544  B4 BB               C       mov     ah,h_undefd         ;undefined error code
D546                      C ret_near_5:
D546  C3                  C       ret                         ;exit
                          C
D547                      C stat_err:
D547  B4 FF               C       mov     ah,h_status         ;read status failed
D549  C3                  C       ret                         ;exit
                          C
                          C ;=======================================
                          C ; Call:         call   ccb_send
                          C ;              return
                          C ;
                          C ; Send the Command Control Block to the controller
                          C ;
                          C ; Entry:  (al) = byte to set into the controller mask port
                          C ;    (ram_ccb + 0, 1, 2, 3, 4, 5) = ccb to send to controller
                          C ;    (si) = offset to port base for target controller
                          C ;
                          C ; Exit:  (CF) = 0 --> no error.  1 --> error
                          C ;    (AH) = Error completion code, if error
                          C ;  Changed:  ax, cx, dx, di
                          C ;  Unchanged:  bx, bp, si
                          C
D54A                      C ccb_send:
D54A  8D 94 0323          C       lea     dx,p_wx2+hd_msk[si]       ;load mask port address
D54E  EE                  C       out     dx,al               ;set mask port
                          C
```

```
D54F  8D 94 0321          C       lea     dx,p_wx2+hd_st[si];load status port address
D553                      C       in      al,dx             ;fetch status from controller.
D554  24 0F               C       and     al,0fh            ;mask off high nibble.
D556  3C 0D               C       cmp     al,0dh            ;check for select status.
D558  74 13               C       jz      busy              ;skip selection phase if board selected!
                          C
                          C  ;Otherwise ...:
D55A  42                  C       inc     dx                ;... get select port address and ...
D55B  EE                  C       out     dx,al             ; ... select controller!
D55C  4A                  C       dec     dx                ;load status port address
D55D  B9 0000             C       mov     cx,0              ;loop counter
D560                      C  busy_wait:
D560  EC                  C       in      al,dx             ;read status
D561  24 0F               C       and     al,0fh            ;mask off high nibble
D563  3C 0D               C       cmp     al,0dh            ;and check for select status
D565  74 06               C       jz      busy              ;is so then exit loop
D567  E2 F7               C       loop    busy_wait;test status until ...
D569                      C  ret_time:
D569  B4 80               C       mov     ah,time_out       ;... time out occurs, and ...
D56B  F9                  C       stc                       ; ... set carry as an error flag
D56C                      C  ret_near_2:
D56C  C3                  C       ret                       ;exit
                          C
D56D                      C  busy:
                          C
                          C  ;if call   wx2_req and following are used then comment out next instruction:
                          C
D56D  8D 94 0320          C       lea     dx,p_wx2[si]      ;load pointer to Data Register.
                          C
D571  BF 0042 R           C       mov     di,offset cmd_block ;point to CDB
D574  B9 0006             C       mov     cx,6              ;six bytes to send to controller
D577  FC                  C       cld                       ;clear direction --> inc di
D578                      C  ccb_byte:
                          C
                          C  ; ******* To decrease overhead eliminate CDB handshake.
                          C  ;    call    wx2_req           ;wait for controller " Ready"
                          C  ;    jc      ret_near_2        ;jump if error.
                          C  ;    and     al,0eh
                          C  ;    xor     al,0ch            ;command byte requested?
                          C  ;    jnz     ret_time ;jump if not.
                          C  ; *******
D578  87 FE               C       xchg    di,si             ;CDB pointer to RSI and save port offset.
D57A  AC                  C       lodsb                     ;CDB pointer to RAL
D57B  87 FE               C       xchg    di,si             ;restore port offset to RSI and save CDB.
D57D  EE                  C       out     dx,al             ;send to controller.
D57E  E2 F8               C       loop    ccb_byte ;loop if more CDB bytes.
D580  E9 D3EF R           C       jmp     ret_no_err
                          C
                          C  ;=======================================
                          C  ; Call:        call   wx2_cc
                          C  ;             return
                          C  ;
                          C  ; Read command completion byte
                          C  ;
                          C  ; Entry:  (si) = offset to port base for target controller
```

```
                        C  ;
                        C  ; Exit:  (cf) = 0 --> no error in reading completion byte
                        C  ;             1 --> error, completion byte not read
                        C  ;    (zf) =  Valid only if (cf) = 0
                        C  ;             1 --> error bit in completion byte not set
                        C  ;             0 --> error bit in completion byte set
                        C  ;    (ah) = 0
                        C  ;   changed:  ax, dx
                        C  ;   unchanged:  bx, cx, bp, si, di
                        C
D583                    C  ww2_cc:
D583  E8 D5A0 R         C       call    wx2_req             ;wait for request.
D586  B4 00             C       mov     ah,h_no_err         ;no error completion code
D588  72 15             C       jc      ret_near_4          ;jump if error
D58A  24 0E             C       and     al,h_busy or h_cd or h_io?
D58C  3C 0E             C       cmp     al,h_busy or h_cd or h_io?
D58E  75 0E             C       jnz     ret_stc             ;jump if not a completion code byte
D590  EC                C       in      al,dx               ;read command completion byte
D591  8A E0             C       mov     ah,al               ;and save it into RAH
D593  42                C       inc     dx                  ;Address of status port.
                        C
D594                    C  cc_busy:
D594  EC                C       in      al,dx               ;read status.
D595  24 08             C       and     al,h_busy;busy?
D597  75 FB             C       jnz     cc_busy             ;jump if busy.
D599  86 C4             C       xchg    al,ah               ;clear RAH and fetch completion byte
D59B  A8 02             C       test    al,cc_er ;error?
D59D  C3                C       ret                         ;exit
                        C
D59E                    C  ret_stc:
D59E  F9                C       stc                         ;set carry
D59F                    C  ret_near_4:
D59F  C3                C       ret                         ;exit
                        C
                        C  ;=======================================
                        C  ; Call:         call   wx2_req
                        C  ;              return
                        C  ;
                        C  ; Wait for request signal from target controller
                        C  ;
                        C  ; Entry:   (SI) = Offset to port base for target controller
                        C  ;
                        C  ; Exit:   (CF) = 0 --> no error.  1 --> error
                        C  ;      (al) = Last status read
                        C  ;      (ah) = completion code if error
                        C  ;     (DX) = Address of port wx2_r/w_data for target controller
                        C  ;  unchanged:  bx, cx, bp, si, di
                        C
D5A0                    C  wx2_req:
D5A0  8D 94 0321        C       lea     dx,p_wx2+hd_st[si];status port of target controller.
D5A4                    C  req_1:
D5A4  EC                C       in      al,dx               ;read status.
D5A5  A8 01             C       test    al,h_rqs ;data request (& clear CF)?
D5A7  75 07             C       jnz     req_succ ;jump if yes.
D5A9  A8 08             C       test    al,h_busy;busy?
```

```
D5AB  75 F7              C       jnz     req_1           ;jump if busy.
                         C
D5AD  F9                 C       stc                     ;set carry to indicate error.
D5AE  B4 80              C       mov     ah,time_out     ;error code.
                         C
D5B0                     C  req_succ:
D5B0  4A                 C       dec     dx              ;port address for data I/O port.
                         C                               ;decrement dos not affect CF!
D5B1  C3                 C       ret                     ;exit.
                         C
                         C  ;====================================
                         C  ; Call:        call    subtable
                         C  ;            return
                         C  ;
                         C  ; Calculate the address of the subtable to be used for the target drive
                         C  ;
                         C  ; Entry: (si) = Offset to port for target controller
                         C  ;    (ram_ccb+1) and ccb_drv_b = Drive bit
                         C  ;
                         C  ; Exit:  (ES:BX) = Address of parameter subtable for target drive
                         C
D5B2                     C  subtable:
                         C       assume  es:abs0
                         C
D5B2  33 C0              C       xor     ax,ax           ;zero RAX.
D5B4  8E C0              C       mov     es,ax           ;satisfy above " assume"
                         C
D5B6  26: C4 1E 0104     C       les     bx,dword ptr es:[4*41h] ;20bit ptr to HDU parameter table.
                         C
D5BB  8D 94 0322         C       lea     dx,p_wx2+hd_cfg[si]     ;controller switch address.
D5BF  EC                 C       in      al,dx           ;read controller switches (4 -- 8).
D5C0  F6 06 0043 R 20    C       test    cmd_block+1,ccb_drv_b   ;Is it drive one (i.e. second HDU)?
D5C5  75 04              C       jnz     drv_1           ;jump if drive 1.
                         C
D5C7  D0 E8              C       shr     al,1
D5C9  D0 E8              C       shr     al,1            ;align swtch# 2,3 and 7,8 to rightest position.
                         C
D5CB                     C  drv_1:
D5CB  24 33              C       and     al,00110011b    ;get mask switch bits for this drive type.
D5CD  8A E0              C       mov     ah,al           ;save switch reading
D5CF  D0 EC              C       shr     ah,1
D5D1  D0 EC              C       shr     ah,1            ;align bits 4 and 5 into position 3 and 4
D5D3  0A C4              C       or      al,ah           ;combine nibbles.
D5D5  B1 04              C       mov     cl,4
D5D7  D2 E0              C       shl     al,cl           ;multiply by 16 to get proper table offset.
                         C                               ;and also get rid of high nibble garbage!
D5D9  B4 00              C       mov     ah,0
D5DB  03 D8              C       add     bx,ax           ;add offset to base address.
D5DD  C3                 C       ret                     ;Exit.
                         C
                         C  command_br   endp
                         C
D5DE                     C  h_fmtproc    near
D5DE  C3                 C       ret
                         C  h_fmtendp
```

```
                        C
D5DF                    C  h_errors      proc     near
D5DF                    C  er_master_tbl label    byte
D5DF  09                C       db       t0l              ;Number of errors in type class.
D5E0  08                C       db       t0_tbl-er_master_tbl       ;byte offset to subtable.
                        C
D5E1  0A                C       db       t1l              ;Number of errors in type class.
D5E2  11                C       db       t1_tbl-er_master_tbl       ;byte offset to subtable
                        C
D5E3  02                C       db       t2l              ;Number of errors in type class.
D5E4  1B                C       db       t2_tbl-er_master_tbl       ;byte offset to subtable
                        C
D5E5  03                C       db       t3l              ;Number of errors in type class.
D5E6  1D                C       db       t3_tbl-er_master_tbl       ;byte offset to subtable
                        C
D5E7                    C  t0_tbl:
D5E7  00                C       db       h_no_err ;no error
D5E8  20                C       db       fdc_error;no index pulse.
D5E9  40                C       db       seek_error       ;no seek complete.
D5EA  20                C       db       fdc_error;write fault.
D5EB  80                C       db       time_out ;drive not ready.
D5EC  00                C       db       h_no_err ;not used
D5ED  20                C       db       fdc_error;track 0 not found
D5EE  00                C       db       h_no_err ;not used
D5EF  40                C       db       seek_error       ;buffered seek in progress.
= 0009                  C  t0l   equ      $-t0_tbl
                        C
D5F0                    C  t1_tbl:
D5F0  10                C       db       crc_error;ECC error in ID field.
D5F1  10                C       db       crc_error;uncorrectable ECC error.
D5F2  02                C       db       addr_mark_error ;address mark not found.
D5F3  00                C       db       h_no_err ;not used.
D5F4  04                C       db       sect_not_found   ;sector not found.
D5F5  40                C       db       seek_error       ;seek error.
D5F6  00                C       db       h_no_err ;not used
D5F7  00                C       db       h_no_err ;not used
D5F8  11                C       db       h_ecc_cor;ECC error corrected.
D5F9  0B                C       db       h_bad_trk;bad track.
= 000A                  C  t1l   equ      $-t1_tbl
                        C
D5FA                    C  t2_tbl:
D5FA  01                C       db       cmd_error;invalid command.
D5FB  02                C       db       addr_mark_error ;illegal disk address.
= 0002                  C  t2l   equ      $-t2_tbl
                        C
D5FC                    C  t3_tbl:
D5FC  20                C       db       fdc_error;RAM failure.
D5FD  20                C       db       fdc_error;ROM checksum error.
D5FE  10                C       db       crc_error;ECC subsystem failure.
= 0003                  C  t3l   equ      $-t3_tbl
                        C
                        C  h_errors      endp
D5FF                    C  code ends

                        C  include graph.asm
```

```
                        C
                        C  ;===========================================================================
                        C  ;    Filename:graph.src
                        C  ;
                        C  ;    This module includes the four graphics functions for INT 10h.
                        C  ;
                        C  ;
                        C  ;===========================================================================
                        C
                        C  page
                        C  ;---------------------------------------------------------------------------
                        C  ;       INT 10h Graphics Support
                        C  ;
                        C  ;    Must preserve bx, cx, dx and return values in ax
                        C  ;
                        C  ;    All other registers are saved and restored by video dispatcher.
                        C  ;
                        C  ;    Entered with the following in registers:
                        C  ;             al, bx, cx, dx intact (set by INT 10 invoker)
                        C  ;             ah = v_mode
                        C  ;---------------------------------------------------------------------------
                        C
D5FF                    C  code segment  public 'ROM'
                        C       assume   cs:code, ds:data, es:v_ram, ss:nothing
                        C
                        C  page
                        C  ;===========================================================================
                        C  ;
                        C  ;    Read Dot                    function code = 0Dh
                        C  ;
                        C  ;    reads a pixel from the indicated location returning its value.
                        C  ;    valid in graphics modes only.
                        C  ;
                        C  ;    Input:
                        C  ;             AH = CRT Mode
                        C  ;             DX = row (0..199 in Med. & Hi-Res., 0..399 in Ultra Res Mode)
                        C  ;             CX = column (0..319 in Med, 0..639 in Hi & Ultra Res. Modes)
                        C  ;
                        C  ;    Output:
                        C  ;             AL = dot value read
                        C  ;             AH = mask of pixel in byte (from g_addr)
                        C  ;
                        C  ;    Saved:   BX, CX, DX (Video dispatcher saves the rest)
                        C  ;
                        C  ;---------------------------------------------------------------------------
                        C
D5FF                    C  grf_read_dot  proc     near
                        C
D5FF  52                C       push     dx                      ; save registers
D600  51                C       push     cx
                        C
D601  50                C       push     ax                      ; save crt mode (high byte)
D602  E8 D644 R         C       call     g_addr                  ; compute address & mask
D605  8A C4             C       mov      al,ah                   ; copy mask
D607  26: 22 05         C       and      al,byte ptr es:[di]     ; AND with byte containing pixel
```

```
D60A  5A              C       pop     dx                      ; dh = crt mode
D60B  FE C1           C       inc     cl                      ; prep for wraparound left (B & W)
D60D  80 FE 05        C       cmp     dh,5                    ; color?
D610  7F 02           C       jg      g_jsfy_dot              ; jump if not
D612  FE C1           C       inc     cl                      ; prep for wraparound left (color)
D614                  C  g_jsfy_dot:
D614  D2 C0           C       rol     al,cl                   ; right justify pixel
                      C
D616  59              C       pop     cx                      ; restore registers
D617  5A              C       pop     dx
D618  C3              C       ret
                      C
                      C  grf_read_dot  endp
                      C
                      C  page
                      C  ;===========================================================================
                      C  ;
                      C  ;    Write Dot                  function code = 0Ch
                      C  ;
                      C  ;    writes a pixel, with the indicated value, at the indicated location.
                      C  ;    valid in graphics modes only.
                      C  ;
                      C  ;    Input:
                      C  ;            AL = dot value to write (1 or 2 bits depending on mode,
                      C  ;                 right justified).  Bit 7 = 1 specifies XOR the value
                      C  ;                 with the pixel at DX, CX
                      C  ;            AH = CRT Mode
                      C  ;            DX = row (0-399)     (the actual value depends on the mode)
                      C  ;            CX = column (0-639)  (the values are not range checked)
                      C  ;
                      C  ;    Saved:  AX, BX, CX, DX (Video dispatcher saves the rest)
                      C  ;
                      C  ;---------------------------------------------------------------------------
                      C
D619                  C  grf_write_dot proc     near
                      C
D619  52              C       push    dx              ; preserve working registers
D61A  51              C       push    cx
D61B  50              C       push    ax
                      C
D61C  E8 D644 R       C       call    g_addr          ; compute address & mask
D61F  26: 8A 05       C       mov     al,byte ptr es:[di] ; fetch byte from video memory
D622  5A              C       pop     dx              ; get v_mode & dot value to write
D623  52              C       push    dx              ; resave
D624  0A D2           C       or      dl,dl           ; is the XOR bit set?
D626  78 06           C       js      g_xorbit ; jump if yes
D628  F6 D4           C       not     ah              ; invert mask
D62A  22 C4           C       and     al,ah           ; clear bits for new pixel
D62C  F6 D4           C       not     ah              ; restore mask for later
D62E                  C  g_xorbit:
D62E  FE C1           C       inc     cl              ; prep for wraparound right (b&w)
D630  80 FE 05        C       cmp     dh,5            ; color ?
D633  7F 02           C       jg      g_align_dot     ; jump if no
D635  FE C1           C       inc     cl              ; prep for wraparound right (color)
D637                  C  g_align_dot:
```

```
D637  D2 CA              C       ror     dl,cl               ; align new pixel
D639  22 D4              C       and     dl,ah               ; strip off non-pixel bits (xor bit, etc)
D63B  32 C2              C       xor     al,dl               ; OR or XOR in new pixel depending on xor bit
D63D  26: 88 05          C       mov     byte ptr es:[di],al ; update video memory
                         C
D640  58                 C       pop     ax                  ; restore registers
D641  59                 C       pop     cx
D642  5A                 C       pop     dx
D643  C3                 C       ret
                         C
                         C grf_write_dot endp
                         C
                         C page
                         C ;===========================================================================
                         C ;
                         C ;    This subroutine determines the video RAM byte location
                         C ;    of the indicated row column value.  The current graphics mode
                         C ;    is taken into account.
                         C ;
                         C ;    INPUT:
                         C ;             AH = current graphics mode
                         C ;             DX = row value (0-399)
                         C ;             CX = column value (0-639)
                         C ;
                         C ;    OUTPUT:
                         C ;             DI = byte address of pixel location in video ram
                         C ;             AH = mask of pixel in byte
                         C ;            CL = # of bits from left end of byte to leftmost bit in mask
                         C ;
                         C ;    DESTROYED:
                         C ;             AL, CX, DX, BP
                         C ;
                         C ;---------------------------------------------------------------------------
                         C
D644                     C g_addr                proc    near
                         C
                         C ; convert row-count to a "mod 4" value:
                         C ;   multiply it by 2000H for offset to start of v_ram subarea
                         C
D644  51                 C       push    cx                      ; save column count
D645  BD 0001            C       mov     bp,1                    ; shift count: assume not 640x400
D648  8B CD              C       mov     cx,bp                   ; multiplier: assume not 640x400
D64A  80 FC 40           C       cmp     ah,64                   ; video mode 64 or 72?
D64D  72 06              C       jb      g_skp_1                 ;       -no: jmp
D64F  BD 0002            C       mov     bp,2                    ;       -yes: change shift count
D652  B9 0003            C       mov     cx,3                    ;       -yes: change multiplier
                         C
D655                     C g_skp_1:
D655  33 FF              C       xor     di,di                   ; init offset amount
D657  23 CA              C       and     cx,dx                   ; get least sig. bit(s) from row count
D659  74 06              C       jz      g_skp_3                 ;     jmp if nothing to add
D65B                     C g_skp_2:                              ; mini-multiply loop
D65B  81 C7 2000         C       add     di,2000H        ;     add v_ram sub-area size
D65F  E2 FA              C       loop    g_skp_2                 ;     do it again (maybe)
                         C
```

```
                        C  ; add bytes for the row coordinate to the offset into the v_ram subarea
                        C
D661                    C  g_skp_3:
D661  57                C       push    di                      ; temp. store offset
D662  8B FA             C       mov     di,dx                   ; DX<--row count
D664  8B CD             C       mov     cx,bp                   ; CX<--mode-related shift val
D666  D3 EF             C       shr     di,cl                   ; get #rows into a v_ram subarea
                        C
D668  8B D7             C       mov     dx,di                   ; save a copy for a moment
D66A  B9 0406           C       mov     cx,0406H        ; mult. di by 80 [rows-->byte offset]
D66D  D3 E7             C       sal     di,cl                   ;       [fast multiply-
D66F  8A CD             C       mov     cl,ch                   ;           by-80]
D671  D3 E2             C       sal     dx,cl
D673  03 FA             C       add     di,dx                   ; #subarea byte offset for this row
                        C
D675  59                C       pop     cx                      ; retrieve subarea beginning's offset
D676  03 F9             C       add     di,cx                   ; offset from start of v_ram
D678  5A                C       pop     dx                      ; restore column count
                        C
                        C  ; find column offset
                        C
D679  B9 0703           C       mov     cx,703H                 ; initialize for black & white
D67C  80 FC 05          C       cmp     ah,5                    ; color ?
D67F  7F 03             C       jg      g_skp_4                 ; CX = 703H (black & white)
D681  B9 0302           C       mov     cx,302H                 ; CX = 302H (color)
D684                    C  g_skp_4:
                        C
                        C  ; CL = shift count to divide column by pixels per byte ...
                        C  ;      3 for b&w (divide by 8). 2 for color (divide by 4)
                        C  ; CH = remainder's mask during division (7 for b&w . 3 for color)
                        C  ; DX = column
                        C  ; DI = address to column 0 of requested row
                        C
D684  22 EA             C       and     ch,dl                   ; get division's remainder in CH
D686  D3 EA             C       shr     dx,cl                   ; perform division
                        C
                        C  ; DX = quotient = column's byte offset from start of row
                        C  ; CH = remainder (= pixel's offset into byte)
                        C
D688  03 FA             C       add     di,dx                   ; DI = pixel's byte address
                        C
                        C  ;    NOW:
                        C  ;            DI = address of byte containing the pixel
                        C  ;            CH = pixel offset into byte (remainder)
                        C  ;            CL = 3 (black & white) or 2 (color)
                        C
D68A  80 F9 03          C       cmp     cl,3                    ; black & white?
D68D  74 02             C       je      g_bitmask       ; jump if yes
D68F  D0 E5             C       shl     ch,1                    ; color: convert to bit offset
                        C
D691                    C  g_bitmask:
D691  86 CD             C       xchg    cl,ch                   ; load CL, preserve "mode"
D693  B4 80             C       mov     ah,80H                  ; set high bit in byte
D695  D2 EC             C       shr     ah,cl                   ; mask pixel's leftmost bit
D697  80 FD 03          C       cmp     ch,3                    ; black & white?
```

```
D69A  74 06              C       je      g_skp_5                 ; jump if yes
D69C  8A C4              C       mov     al,ah                   ; color: create 2-bit mask
D69E  D0 E8              C       shr     al,1
D6A0  0A E0              C       or      ah,al
                         C
D6A2                     C  g_skp_5:                             ; DI = byte address, AH = pixel mask,
D6A2  C3                 C       ret                             ; CL = bit offset: pixel's leftmost bit
                         C
                         C  g_addr       endp
                         C
                         C  page
                         C  ;===============================================================================
                         C  ;
                         C  ; Scroll Up In Graphics Mode
                         C  ;
                         C  ;    Scroll up the number of lines specified within the specified screen
                         C  ;    area (window).
                         C  ;
                         C  ;    Input:
                         C  ;            AL =    number of lines to be scrolled up ( zero
                         C  ;                    means clear the window)
                         C  ;            BH =    fill pattern to be used
                         C  ;            CH,CL = upper left corner of window in which to scroll
                         C  ;            DH,DL = lower right corner of window in which to scroll
                         C  ;            DS =    data segment
                         C  ;            ES =   graphics refresh ram segment
                         C  ;
                         C  ;    Saved:  BX, CX, DX (Video dispatcher saves the rest)
                         C  ;
                         C  ;-------------------------------------------------------------------------------
                         C
D6A3                     C  grf_graphics_up      proc    near
                         C
D6A3  53                 C       push    bx                      ; save
D6A4  51                 C       push    cx                      ;       the
D6A5  52                 C       push    dx                      ;            registers
                         C
                         C  ;    cld                             ;dir. flag = increment (from v_scrl_up)
D6A6  BD 0050            C       mov     bp,80                   ;offset to next scanline (CLD => +80)
                         C
D6A9  50                 C       push    ax                      ;save mode, # rows to scroll
D6AA  8B C1              C       mov     ax,cx                   ;compute address of window's
D6AC  E8 D99D R          C       call    g_curs_off              ;   upper left corner
D6AF  8B F8              C       mov     di,ax                   ;save in DI for string instructions
                         C
D6B1  06                 C       push    es                      ;set DS to video ram for string inst.
D6B2  1F                 C       pop     ds
                         C
D6B3  2B D1              C       sub     dx,cx                   ;compute window's dimensions
D6B5  81 C2 0101         C       add     dx,101H                 ;   DH = height,  DL = width
                         C
D6B9  58                 C       pop     ax                      ;AH = mode,  AL = # rows to scroll
                         C
D6BA  B3 02              C       mov     bl,2                    ;# interlace areas = 2 for modes 4,5,6
D6BC  8A CB              C       mov     cl,bl                   ;# scanlines per i.a. = 4 for modes ~72
```

```
D6BE  80 FC 40          C       cmp     ah,64
D6C1  72 06             C       jb      g_tst_mod       ;jump if mode = 4,5,6
D6C3  B3 04             C       mov     bl,4                    ;# interlace areas = 4 for modes 64 & 72
D6C5  74 02             C       je      g_tst_mod       ;jump if mode = 64
D6C7  D0 F9             C       sar     cl,1                    ;# scanlines per i.a. = 2 for mode 72
                        C
D6C9                    C g_tst_mod:
D6C9  D2 E0             C       sal     al,cl                   ;convert number of rows to number of
D6CB  D2 E6             C       sal     dh,cl                   ;   scanlines per interlace area
D6CD  8B C8             C       mov     cx,ax                   ;CH = mode, CL = # scanlines to scroll
D6CF  80 FD 06          C       cmp     ch,6                    ;are we in a medium resolution mode ?
D6D2  7D 04             C       jge     g_set_up        ;jump if no
D6D4  D1 E7             C       sal     di,1                    ;double number of bytes per character
D6D6  D0 E2             C       sal     dl,1
                        C
D6D8                    C g_set_up:     ;get address of lines to scroll in refresh ram memory
D6D8  81 E1 00FF        C       and     cx,00FFH        ;CX = # of scanlines to scroll per i.a.
D6DC  74 3C             C       jz      g_filler        ;if zero, go fill all of window
                        C
D6DE  8B C1             C       mov     ax,cx                   ;make DH = # scanlines to fill per i.a.
D6E0  8A E9             C       mov     ch,cl                   ;       AX = # scanlines to scroll "   "
D6E2  86 F5             C       xchg    dh,ch                   ;       CH = # scanlines in window "   "
                        C
D6E4  8B F7             C       mov     si,di                   ;compute address of scanline to be
D6E6  B1 04             C       mov     cl,4                    ;   scrolled to top of window:
D6E8  D3 E0             C       sal     ax,cl                   ;   <window's address> +
D6EA  03 F0             C       add     si,ax                   ;   (<# scanlines to scroll per i.a.> *
D6EC  D1 E0             C       sal     ax,1                    ;    <# bytes per scanline = 80>)
D6EE  D1 E0             C       sal     ax,1
D6F0  03 F0             C       add     si,ax
                        C
D6F2  8A E5             C       mov     ah,ch                   ;compute # of scanlines per i.a.
D6F4  2A E6             C       sub     ah,dh                   ;   to be moved
                        C
D6F6  33 C9             C       xor     cx,cx                   ;CH = 0 for REP counter
                        C
                        C ;     jmp short g_scroller            ;go scroll up and fill
                        C
                        C grf_graphics_up       endp
                        C
                        C page
                        C ;===========================================================================
                        C ;
                        C ;     Scroll rows in graphics refresh memory
                        C ;
                        C ;     Input:
                        C ;             AH =    Number of scanlines per interlace area to be moved
                        C ;             BL =    Number of interlace areas
                        C ;             CH =    0
                        C ;             DI =    Destination scanline address of first byte to fill
                        C ;             SI =    Source scanline address of first byte to fill
                        C ;             DL =    Number of bytes to fill in each scanline
                        C ;             BP =    offset to next scanline (+/- 80)
                        C ;
                        C ;     Output:
```

```
                        C  ;            DI =    address of first byte to be filled
                        C  ;            AH =    0
                        C  ;
                        C  ;    BL,BH,CH,DL,DH,BP preserved
                        C  ;
                        C  ;--------------------------------------------------------------------------
                        C
D6F8                    C  g_scroller   proc    near
                        C
D6F8  56                C       push    si                      ;save original source
D6F9  57                C       push    di                      ;  and destination addresses
D6FA  53                C       push    bx                      ;save interlace areas count (BL)
                        C
D6FB                    C  g_m_area:    ;Move Interlace Areas Loop
D6FB  57                C       push    di                      ;save interlace area
D6FC  56                C       push    si                      ;   addresses
D6FD  8A CA             C       mov     cl,dl                   ;count of bytes to be moved
D6FF  F3/ A4            C       rep     movsb                   ;move the scanline
D701  5E                C       pop     si                      ;restore interlace area
D702  5F                C       pop     di                      ;   addresses
D703  81 C7 2000        C       add     di,2000H        ;next interlace area
D707  81 C6 2000        C       add     si,2000H        ;   addresses
D70B  FE CB             C       dec     bl                      ;loop to move one scanline in
D70D  75 EC             C       jnz     g_m_area        ;   each interlace area
                        C
D70F  5B                C       pop     bx                      ;restore interlace areas count (BL)
D710  5F                C       pop     di                      ;restore original source
D711  5E                C       pop     si                      ;   and destination addresses
D712  03 FD             C       add     di,bp                   ;address next scanline in
D714  03 F5             C       add     si,bp                   ;   each interlace area
D716  FE CC             C       dec     ah                      ;loop to move "all" scanlines in
D718  75 DE             C       jnz     g_scroller              ;   each interlace area
                        C
                        C  ;    jmp short g_filler      ;now fill in the gap
                        C
                        C  g_scroller   endp
                        C
                        C  page
                        C  ;==========================================================================
                        C  ;
                        C  ; Fill rows in graphics refresh memory with the fill pattern
                        C  ;
                        C  ;
                        C  ;    Input:
                        C  ;            BL =    Number of interlace areas
                        C  ;            BH =    Fill pattern to be used
                        C  ;            CH =    0
                        C  ;            DL =    Number of bytes to fill in each scanline
                        C  ;            DH =    Number of scanlines to fill in each interlace area
                        C  ;            DI =    Destination scanline address of first byte to fill
                        C  ;            BP =    offset to next scanline (+/- 80)
                        C  ;
                        C  ;    Output:
                        C  ;            AL =    fill pattern
                        C  ;            AH =    0
```

```
                        C  ;
                        C  ;--------------------------------------------------------------------------
                        C
D71A                    C  g_filler     proc    near
                        C
D71A  8A C7             C       mov     al,bh                   ;AL = fill pattern for STOSB instruction
                        C
D71C                    C  g_f_i_lp:    ;Fill Interlace Areas Loop
D71C  57                C       push    di                      ;save interlace area address
D71D  52                C       push    dx                      ;save scanlines count (DH)
                        C
D71E                    C  g_f_s_lp:    ;Fill Scanlines Loop
D71E  57                C       push    di                      ;save scanline address
D71F  8A CA             C       mov     cl,dl                   ;count of bytes to fill in scanline
D721  F3/ AA            C       rep     stosb                   ;fill scanline
D723  5F                C       pop     di                      ;restore scanline address
D724  03 FD             C       add     di,bp                   ;next scanline in interlace area
D726  FE CE             C       dec     dh                      ;fill an interlace area
D728  75 F4             C       jnz     g_f_s_lp
                        C
D72A  5A                C       pop     dx                      ;restore scanlines count (DH)
D72B  5F                C       pop     di                      ;restore interlace area address
D72C  81 C7 2000        C       add     di,2000H        ;next interlace area address
D730  FE CB             C       dec     bl                      ;fill next interlace area
D732  75 E8             C       jnz     g_f_i_lp
                        C
D734  5A                C       pop     dx                      ;restore
D735  59                C       pop     cx                      ;       the
D736  5B                C       pop     bx                      ;            registers
D737  C3                C       ret                             ;exit scroll routine
                        C
                        C  g_filler     endp
                        C
                        C  page
                        C  ;==========================================================================
                        C  ;
                        C  ; Scroll Down In Graphics Mode
                        C  ;
                        C  ;    Scroll down the number of lines specified within the specified screen
                        C  ;    area (window).
                        C  ;
                        C  ;    Input:
                        C  ;            AL    = number of lines to be scrolled up ( zero
                        C  ;                    means clear the window)
                        C  ;            BH    = fill pattern to be used
                        C  ;            CH,CL =  upper left corner of window in which to scroll
                        C  ;            DH,DL =  lower right corner of window in which to scroll
                        C  ;            DS    = data segment
                        C  ;            ES    = graphics refresh ram segment
                        C  ;
                        C  ;    Saved:  BX, CX, DX (Video dispatcher saves the rest)
                        C  ;
                        C  ;--------------------------------------------------------------------------
                        C
D738                    C  grf_graphics_down    proc near
```

```
                        C
D738  53                C       push    bx                      ; save
D739  51                C       push    cx                      ;       the
D73A  52                C       push    dx                      ;            registers
                        C
                        C ;     std                             ;dir. flag = decrement (from v_scrl_dn)
D73B  BD FFB0           C       mov     bp,-80                  ;offset to next scanline (STD => -80)
                        C
D73E  50                C       push    ax                      ;save mode, # rows to scroll
D73F  8B C2             C       mov     ax,dx                   ;compute address of window's
D741  E8 D99D R         C       call    g_curs_off              ;   lower right corner
D744  8B F8             C       mov     di,ax                   ;save in DI for string instructions
                        C
D746  06                C       push    es                      ;set DS to video ram for string inst.
D747  1F                C       pop     ds
                        C
D748  2B D1             C       sub     dx,cx                   ;compute window's dimensions
D74A  81 C2 0101        C       add     dx,101H                 ;   DH = height,  DL = width
                        C
D74E  58                C       pop     ax                      ;AH = mode,  AL = # rows to scroll
                        C
D74F  B3 02             C       mov     bl,2                    ;# interlace areas = 2 for modes 4,5,6
D751  8A CB             C       mov     cl,bl                   ;# scanlines per i.a. = 4 for modes ~72
D753  80 FC 40          C       cmp     ah,64
D756  72 06             C       jb      g_cmp_mod       ;jump if mode = 4,5,6
D758  B3 04             C       mov     bl,4                    ;# interlace areas = 4 for modes 64 & 72
D75A  74 02             C       je      g_cmp_mod       ;jump if mode = 64
D75C  D0 F9             C       sar     cl,1                    ;# scanlines per i.a. = 2 for mode 72
                        C
D75E                    C g_cmp_mod:
D75E  D2 E0             C       sal     al,cl                   ;convert number of rows to number of
D760  D2 E6             C       sal     dh,cl                   ;   scanlines per interlace area
D762  80 FC 06          C       cmp     ah,6                    ;are we in a medium resolution mode ?
D765  7D 05             C       jge     g_setdown       ;jump if no
D767  D1 E7             C       sal     di,1                    ;double number of bytes per character
D769  D0 E2             C       sal     dl,1
D76B  47                C       inc     di                      ;address last byte in bottom row
                        C
D76C                    C g_setdown:    ;get address of lines to scroll in refresh RAM memory
D76C  BE 0050           C       mov     si,80                   ;address bottom scanline in i.a.
D76F  D3 E6             C       sal     si,cl
D771  83 EE 50          C       sub     si,80
D774  03 FE             C       add     di,si
D776  8B C8             C       mov     cx,ax                   ;CH = mode, CL = # scanlines to scroll
D778  81 E1 00FF        C       and     cx,00FFH        ;CL = # of scanlines to scroll per i.a.
D77C  74 9C             C       jz      g_filler        ;if zero, go fill all of window
                        C
D77E  8B C1             C       mov     ax,cx                   ;make DH = # scanlines to fill per i.a.
D780  8A E9             C       mov     ch,cl                   ;     AX = # scanlines to scroll "  "
D782  86 F5             C       xchg    dh,ch                   ;     CH = # scanlines in window "  "
                        C
D784  8B F7             C       mov     si,di                   ;compute address of scanline to be
D786  B1 04             C       mov     cl,4                    ;   scrolled to top of window:
D788  D3 E0             C       sal     ax,cl                   ;   <window's address> -
D78A  2B F0             C       sub     si,ax                   ;   (<# scanlines to scroll per i.a.> *
```

```
D78C  D1 E0              C       sal     ax,1                    ;    <# bytes per scanline = 80>)
D78E  D1 E0              C       sal     ax,1
D790  2B F0              C       sub     si,ax
                         C
D792  8A E5              C       mov     ah,ch                   ;compute # of scanlines per i.a.
D794  2A E6              C       sub     ah,dh                   ;   to be moved
                         C
D796  33 C9              C       xor     cx,cx                   ;CH = 0 for REP counter
                         C
D798  E9 D6F8 R          C       jmp     g_scroller              ;go scroll down and fill
                         C
                         C  grf_graphics_down     endp
                         C
                         C  page
                         C  ;===========================================================================
                         C  ;
                         C  ;    graphics_read - read the character at the current cursor
                         C  ;                    position on the screen, or zero.
                         C  ;    Input:  none
                         C  ;
                         C  ;    Output: AL =    character at the current cursor position or zero
                         C  ;
                         C  ;    Saved:  BX, CX, DX (video dispatcher saves the rest)
                         C  ;
                         C  ;---------------------------------------------------------------------------
                         C
D79B                     C  grf_graphics_read     proc    near
                         C
D79B  53                 C       push    bx                      ; save
D79C  51                 C       push    cx                      ;       the
D79D  52                 C       push    dx                      ;           registers
                         C
D79E  8A DC              C       mov     bl,ah                   ;BL saves video_mode
D7A0  A1 0050 R          C       mov     ax,word ptr ds:[v_curpos]
D7A3  E8 D99D R          C       call    g_curs_off              ;get address of cursor position
D7A6  8B F0              C       mov     si,ax                   ;save as a pointer for later
                         C
D7A8  8A C3              C       mov     al,bl                   ;AL saves video_mode
D7AA  B9 0004            C       mov     cx,4                    ;# scanlines per i.a. = 4 for modes 72
D7AD  33 DB              C       xor     bx,bx                   ;make BX=0 for modes 4,5,6
D7AF  3C 40              C       cmp     al,64
D7B1  72 08              C       jb      g_lds_r         ;jump if mode = 4,5,6
D7B3  B3 04              C       mov     bl,4                    ;make BX=4 for mode 64
D7B5  74 04              C       je      g_lds_r         ;jump if mode = 64
D7B7  D1 F9              C       sar     cx,1                    ;# scanlines per i.a. = 2 for mode 72
D7B9  33 DB              C       xor     bx,bx                   ;make BX=0 for mode 72
                         C
D7BB                     C  g_lds_r:              ;(BX= font pointer offset in master table)
D7BB  C5 3E 0084 R       C       lds     di,dword ptr ds:[master_tbl_ptr] ;get pointer to master table
D7BF  C5 79 06           C       lds     di,dword ptr ds:[di+6][bx] ;get pointer to font's 1st 128 chars
D7C2  1E                 C       push    ds                      ;XCHG DS,ES
D7C3  06                 C       push    es
D7C4  1F                 C       pop     ds                      ;DS:SI -> grafix ram read location  ???
D7C5  07                 C       pop     es                      ;ES:DI -> font's 1st 128 characters
                         C
```

```
D7C6  D1 E3          C       sal     bx,1                    ;make BX=8 for modes 4,5,6,72
D7C8  83 C3 08       C       add     bx,8                    ;     BX=16 for mode 64
                     C               ;(BX= number of font bytes per character)
                     C
D7CB  2B E3          C       sub     sp,bx                   ;get stack space for font bytes
D7CD  8B EC          C       mov     bp,sp                   ;save pointer to stack space
D7CF  BA 0002        C       mov     dx,2                    ;number of interlace areas (modes 4,5,6)
D7D2  3C 06          C       cmp     al,6                    ;check graphics mode
D7D4  7C 29          C       jl      g_rd_med        ;jump if in medium resolution mode (4,5)
D7D6  74 03          C       je      g_rdloop        ;jump if in 640x200 resolution mode (6)
                     C
                     C  ; we're in super resolution (640x400) mode (64 & 72)
D7D8  BA 0004        C       mov     dx,4                    ;number of interlace areas
                     C
D7DB                 C  g_rdloop:            ;Read Scanlines Loop for both 640x200 and 640x400
D7DB  51             C       push    cx                      ;save # of scanlines per interlace area
D7DC  56             C       push    si                      ;save current addr. in interlace area #1
D7DD  8B CA          C       mov     cx,dx                   ;init. counter: # of interlace areas
                     C
D7DF                 C  g_rd_ia:             ;Read Interlace Area Loop
D7DF  8A 04          C       mov     al,[si]         ;get byte from grafix ram
D7E1  88 46 00       C       mov     [bp],al         ;save on reserved stack
D7E4  45             C       inc     bp                      ;bump reserved stack address
D7E5  81 C6 2000     C       add     si,2000H        ;address next interlace area
D7E9  E2 F4          C       loop    g_rd_ia
                     C
D7EB  5E             C       pop     si                      ;restore interlace area #1 address
D7EC  83 C6 50       C       add     si,80                   ;address next scanline in each i.a.
D7EF  59             C       pop     cx                      ;restore # of scanlines counter
D7F0  E2 E9          C       loop    g_rdloop
                     C
D7F2  83 FA 04       C       cmp     dx,4                    ;are we in mode 64 or 72 (640x400)?
D7F5  BA 0000        C       mov     dx,0                    ;clear dx without disturbing flags.
D7F8  75 3C          C       jne     g_matchb        ;jump if no (reverse video not allowed)
D7FA  BA 0001        C       mov     dx,1                    ;if we don't match the charcter on
                     C                                       ;normal video pass, we will reverse
                     C                                       ;video the character and try again.
                     C
D7FD  EB 37          C       jmp     short g_matchb
                     C
                     C
D7FF                 C  g_rd_med:                    ;Read Medium Resolution
D7FF  D1 E6          C       sal     si,1                    ;double graf ram pointer (2 bytes/char)
                     C
D801                 C  g_medget:            ;Get font bytes in medium resolution (320 X 200) mode
D801  51             C       push    cx                      ;save # scanlines per i.a. counter
D802  56             C       push    si                      ;save current scanline address
D803  B9 0002        C       mov     cx,2                    ;init. # interlace areas counter
D806                 C  g_med_ia:
D806  51             C       push    cx              ;save i.a. counter
D807  8B 04          C       mov     ax,[si]         ;get 2 bytes of 1 char from video memory
D809  86 E0          C       xchg    ah,al                   ;order them logically
                     C
D80B  F7 D0          C       not     ax                      ;map background pixels to 0,
D80D  8B D0          C       mov     dx,ax                   ;     foreground pixels to 1
```

```
D80F  D1 E2           C       shl     dx,1
D811  23 D0           C       and     dx,ax
D813  F7 D2           C       not     dx
                      C
D815  32 C0           C       xor     al,al                   ;clear result accumulator
D817  B9 0008         C       mov     cx,8                    ;prepare to process 8 bits
D81A                  C  g_med_bit:
D81A  D1 EA           C       shr     dx,1                    ;ignore unused bit
D81C  D1 EA           C       shr     dx,1                    ;load carry with mapped pixel value
D81E  D0 D8           C       rcr     al,1                    ;rotate it into result accumulator
D820  E2 F8           C       loop    g_med_bit       ;process next bit of character
                      C
D822  88 46 00        C       mov     [bp],al         ;save font byte on reserved stack
D825  45              C       inc     bp                      ;bump reserved stack pointer
D826  81 C6 2000      C       add     si,2000H        ;address next interlace area
D82A  59              C       pop     cx                      ;restore i.a. counter
D82B  E2 D9           C       loop    g_med_ia        ;process next i.a. of character
                      C
D82D  5E              C       pop     si                      ;restore scanline address
D82E  83 C6 50        C       add     si,80                   ;address next scanline
D831  59              C       pop     cx                      ;restore scanlines counter
D832  E2 CD           C       loop    g_medget        ;process next scanline of character
D834  33 D2           C       xor     dx,dx                   ;set dx to zero, so that we
                      C                                       ;don't attempt a reverse video
                      C                                       ;match
D836                  C  g_matchb:                    ;Match Font Byte: find character
D836  06              C       push    es
D837  57              C       push    di              ;save di, in case we need to make another
                      C                               ;pass.
D838  2B EB           C       sub     bp,bx                   ;point to first byte in stack save area
D83A                  C  g_f_rv_cont:                 ;entry point for a reverse video check
D83A  8B F5           C       mov     si,bp                   ;pointer to font bytes from graf ram
D83C  32 C0           C       xor     al,al                   ;index to font bytes (start with 0)
D83E                  C  g_f_cont:                    ;Entry point for second 128 set.
D83E  16              C       push    ss                      ;setup string compare registers
D83F  1F              C       pop     ds                      ;DS:SI -> stack area w/grafix ram bytes
D840  B9 0080         C       mov     cx,128                  ;loop control = 1st 128 ascii chars.
                      C
D843                  C  g_f_mach:                    ;Get Font Byte Match Loop
D843  51              C       push    cx                      ;save loop counter
D844  8B CB           C       mov     cx,bx                   ;counter for string compare
D846  57              C       push    di                      ;save font pointer
D847  56              C       push    si                      ;save pointer to stack save area
D848  F3/ A6          C       repe    cmpsb                   ;screen bytes match font bytes ?
D84A  5E              C       pop     si                      ;retrieve pointer to stack save area
D84B  5F              C       pop     di                      ;retrieve pointer to fonmt byte table
D84C  59              C       pop     cx                      ;restore loop counter
D84D  74 40           C       je      g_f_exit        ;if match go to exit code
D84F  03 FB           C       add     di,bx                   ;address next font in table
D851  FE C0           C       inc     al                      ;bump ascii index
D853  E2 EE           C       loop    g_f_mach        ;go back if more chars to search for
                      C
                      C  ; no match in first 128 ascii character set - look for user's second set
                      C
D855  0A C0           C       or      al,al                   ;have we scanned both 128 char 1/2s ?
```

```
D857  74 23              C       jz      g_f_not_found           ;jump if yes
D859  83 FB 10           C       cmp     bx,16                   ;are we in mode 64 ?
D85C  74 08              C       je      g_8x16_2        ;jump if yes
                         C
D85E  8E D9              C       mov     ds,cx                   ;move zero to segment register
                         C       assume  ds:abs0
D860  C4 3E 007C R       C       les     di,dword ptr ds:[int1Flocn] ;get pointer to 2nd half
                         C                                       ;  of 8x8 font
                         C       assume  ds:data
D864  EB 0C              C       jmp short g_test_addr           ;see if font is really there
                         C
D866                     C g_8x16_2:                     ;get 2nd half of 8x16 font
D866  2E: 8E 1E E574 R   C       mov     ds,word ptr cs:[set_ds_word]    ;set DS to data segment
D86B  C5 3E 0084 R       C       lds     di,dword ptr ds:[master_tbl_ptr] ;get pointer to master table
D86F  C4 7D 0E           C       les     di,dword ptr ds:[di+14] ;get pointer to 2nd half of 8x16 font
                         C
D872                     C g_test_addr:
D872  8C C0              C       mov     ax,es                   ;check if font table is set up
D874  0B C7              C       or      ax,di                   ;if zeros, then no user font table
D876  74 04              C       jz      g_f_not_found           ;no table, just go to exit
D878  B0 80              C       mov     al,128                  ;offset to 2nd 1/2 of ascii set
D87A  EB C2              C       jmp short g_f_cont      ;go back to try rest of ascii set
                         C
D87C                     C g_f_not_found:
D87C  4A                 C       dec     dx                      ;if value of dx was 1, set to
D87D  75 10              C       jnz     g_f_exit        ;0 and go look for a reverse
                         C                                       ;video character.
D87F  5F                 C       pop     di
D880  07                 C       pop     es                      ;recover di register
D881  06                 C       push    es
D882  57                 C       push    di                      ;put on stack so exit is consistent.
                         C
D883  8B F5              C       mov     si,bp                   ;point to first byte in stack save area
D885  8B CB              C       mov     cx,bx                   ;number of char bytes counter
D887                     C g_unreverse_video_loop:
D887  36: F6 14          C       not     byte ptr ss:[si]  ;reverse the reversed byte for matching
D88A  46                 C       inc     si                      ;address next char byte
D88B  E2 FA              C       loop    g_unreverse_video_loop
D88D  EB AB              C       jmp short g_f_rv_cont           ;now find the char in the font table
                         C
D88F                     C g_f_exit:             ;either the character is found, or al = 0
D88F  5A                 C       pop     dx      ;discard es register on stack.
D890  5A                 C       pop     dx      ;discard di register on stack.
D891  03 E3              C       add     sp,bx                   ;restore stack pointer
D893  5A                 C       pop     dx                      ;restore
D894  59                 C       pop     cx                      ;         the
D895  5B                 C       pop     bx                      ;             registers
D896  C3                 C       ret
                         C
                         C grf_graphics_read     endp
                         C
                         C page
                         C ;===========================================================================
                         C ;
                         C ;    graphics_write - write a character to the screen
```

```
                        C  ;
                        C  ;    Input:  AL =      character to write
                        C  ;            BL =      Foreground color attribute
                        C  ;                      bit 7 = 1: xor character with graphics ram
                        C  ;            CX =      number of characters to write
                        C  ;            DS =      data segment
                        C  ;            ES =      graphics ram segment
                        C  ;
                        C  ;    Saved:  BX, CX, DX (video dispatcher saves the rest)
                        C  ;
                        C  ;--------------------------------------------------------------------------
                        C
D897                    C  grf_graphics_write     proc     near
                        C
D897  53                C       push     bx                        ;save
D898  51                C       push     cx                        ;    the
D899  52                C       push     dx                        ;       registers
                        C
D89A  8B D0             C       mov      dx,ax                     ;DH= crt mode, DL= char to write
                        C  ; locate beginning of character in graphics ram
D89C  A1 0050 R         C       mov      ax,word ptr ds:[v_curpos]
D89F  E8 D99D R         C       call     g_curs_off                ;get address of cursor position
D8A2  8B F8             C       mov      di,ax                     ; pointer to graphics location
                        C  ; determine if character is from 1st or 2nd half of table
D8A4  80 FA 80          C       cmp      dl,128                    ;is it in first 1/2 of ASCII set ?
D8A7  72 26             C       jb       g_selfont         ;jump if in 1st 1/2 (0 -> 127)
                        C
                        C  ; character (128 -> 255) is in 2nd 1/2 of font table
                        C
D8A9  80 EA 80          C       sub      dl,128                    ;make zero origin for font table lookup
D8AC  80 FE 40          C       cmp      dh,64                     ;are we in mode 64 ?
D8AF  75 09             C       jne      g_8x8_2                   ;jump if no
                        C
D8B1  C5 36 0084 R      C       lds      si,dword ptr ds:[master_tbl_ptr] ;get pointer to master table
D8B5  C5 74 0E          C       lds      si,dword ptr ds:[si+14]   ;get pointer to 2nd half of 8x16 font
                        C
D8B8  EB 08             C       jmp short g_addr_test              ;see if font table is really there
                        C
D8BA                    C  g_8x8_2:                         ;get 2nd half of 8x8 font table
D8BA  33 F6             C       xor      si,si                     ;move zero to segment register
D8BC  8E DE             C       mov      ds,si
                        C       assume   ds:abs0
D8BE  C5 36 007C R      C       lds      si,dword ptr ds:[int1Flocn] ;get pointer to 2nd half of 8x8 font
                        C       assume   ds:data
                        C
D8C2                    C  g_addr_test:
D8C2  8C D8             C       mov      ax,ds                     ;check if font table is set up
D8C4  0B C6             C       or       ax,si                     ;if zeros, then no 2nd half of table
D8C6  75 1D             C       jnz      g_detmode         ;continue if font table is present
D8C8  32 D2             C       xor      dl,dl                     ;substitute null character
D8CA  2E: 8E 1E E574 R  C       mov      ds,word ptr cs:[set_ds_word]       ;restore data segment register
                        C  ;    jmp short g_selfont                ;and continue in 1st half of font table
                        C
                        C  ; character (0 -> 127) is in 1st 1/2 of font table
                        C
```

```
D8CF                    C  g_selfont:
D8CF  53                C       push    bx                      ;preserve register
                        C
D8D0  33 DB             C       xor     bx,bx                   ;make BX=0 for modes 4,5,6
D8D2  80 FE 40          C       cmp     dh,64
D8D5  72 06             C       jb      g_lds_w                 ;jump if mode = 4,5,6
D8D7  B3 04             C       mov     bl,4                    ;make BX=4 for mode 64
D8D9  74 02             C       je      g_lds_w                 ;jump if mode = 64
D8DB  33 DB             C       xor     bx,bx                   ;make BX=0 for mode 72
                        C
D8DD                    C  g_lds_w:             ;(BX= font pointer offset in master table)
D8DD  C5 36 0084 R      C       lds     si,dword ptr ds:[master_tbl_ptr] ;get pointer to master table
D8E1  C5 70 06          C       lds     si,dword ptr ds:[si+6][bx] ;get pointer to font's 1st 128 chars
                        C
D8E4  5B                C       pop     bx                      ;restore register
                        C
D8E5                    C  g_detmode:                   ;determine graphics mode
D8E5  51                C       push    cx
D8E6  33 C0             C       xor     ax,ax                   ;get ascii code in AX
D8E8  8A C2             C       mov     al,dl
D8EA  B1 03             C       mov     cl,3                    ; to multiply by
D8EC  D3 E0             C       sal     ax,cl                   ;    8 (font bytes per character)
D8EE  59                C       pop     cx
D8EF  03 F0             C       add     si,ax                   ;and add to address of font table
                        C
D8F1  B2 04             C       mov     dl,4                    ;# scanlines per i.a. (modes 4,5,6,64)
                        C
D8F3  80 FE 06          C       cmp     dh,6                    ;which resolution are we using?
D8F6  7C 49             C       jl      g_med_wr        ;jump if medium resolution (modes 4 & 5)
D8F8  74 0F             C       je      g_hi_wr                 ;jump if 640x200 resolution (mode 6)
                        C
                        C                               ;we're in 640x400 resolution
D8FA  80 FE 48          C       cmp     dh,72                   ;mode 72?
D8FD  B6 04             C       mov     dh,4                    ;# interlace areas = 4 for modes 64 & 72
D8FF  75 04             C       jne     g_super_wr
                        C
D901                    C  g_tinytext:                  ;640x400 resolution (mode 72)
D901  B2 02             C       mov     dl,2                    ;# scanlines per i.a. = 2 for mode 72
D903  EB 09             C       jmp short g_repchar
                        C
D905                    C  g_super_wr:                  ;640x400 resolution (mode 64)
D905  03 F0             C       add     si,ax                   ;multiply ascii code by 16 bytes/char
D907  EB 05             C       jmp short g_repchar
                        C
D909                    C  g_hi_wr:                     ;Hi-resolution Character Write
D909  B6 02             C       mov     dh,2                    ;interlace areas count (even/odd)
D90B  80 CB 01          C       or      bl,1                    ;mode 6 doesn't allow reverse video
                        C
D90E                    C  g_repchar:                   ;Repeat Character Loop
D90E  51                C       push    cx                      ;save character repeat count
D90F  56                C       push    si                      ;save source address (font table)
D910  57                C       push    di                      ;save destination addr. (grafix ram)
D911  33 C9             C       xor     cx,cx                   ;prepare loop counter
D913  8A CA             C       mov     cl,dl                   ;scanlines per interlace area counter
                        C
```

```
D915                    C  g_linelp:                    ;Scanline Loop
D915  51                C       push    cx                      ;save scanlines per i.a. counter
D916  57                C       push    di                      ;save interlace area #1 address
D917  8A CE             C       mov     cl,dh                   ;init. interlace areas counter (2 or 4)
                        C
D919                    C  g_i_a_lp:                    ;Interlace Area Loop
D919  AC                C       lodsb                           ;get byte from font table
D91A  F6 C3 01          C       test    bl,1                    ;reverse video?
D91D  75 02             C       jnz     g_t_xor                 ;jump if no
D91F  F6 D0             C       not     al                      ;reverse video
D921                    C  g_t_xor:                     ;Test For XOR
D921  0A DB             C       or      bl,bl                   ;XOR the char. with grafix ram?
D923  79 03             C       jns     g_w_byte        ;jump if no
D925  26: 32 05         C       xor     al,es:[di]              ;XOR with grafix ram
D928                    C  g_w_byte:                    ;Write Byte
D928  26: 88 05         C       mov     es:[di],al              ;write byte in grafix ram
D92B  81 C7 2000        C       add     di,2000H        ;address next interlace area
D92F  E2 E8             C       loop    g_i_a_lp
                        C
D931  5F                C       pop     di                      ;restore interlace area #1 address
D932  83 C7 50          C       add     di,80                   ;address next scanline in each i.a.
D935  59                C       pop     cx                      ;restore scanlines per i.a. counter
D936  E2 DD             C       loop    g_linelp
                        C
D938  5F                C       pop     di                      ;restore char's grafix ram address
D939  47                C       inc     di                      ;address next character in grafix ram
D93A  5E                C       pop     si                      ;restore char's font table address
D93B  59                C       pop     cx                      ;restore character repeat count
D93C  E2 D0             C       loop    g_repchar       ;repeat the character
D93E  EB 55 90          C       jmp     g_return        ;exit
                        C
D941                    C  g_med_wr:                    ;Medium Resolution Character Write
D941  D1 E7             C       sal     di,1                    ;double graf ram pointer (2 bytes/char)
D943  8A D3             C       mov     dl,bl                   ;DL saves XOR bit input param (bit 7)
D945  81 E3 0003        C       and     bx,0003H        ;BX= foreground color (& table offset)
D949  2E: 8A 9F D999 R  C       mov     bl,cs:g_color_table[bx]     ;propagate color through byte
D94E  8A FB             C       mov     bh,bl                   ;propagate color through word
D950  8B EB             C       mov     bp,bx                   ;BP saves word of color masks
                        C
D952                    C  g_char_lp:                   ;Repeat Character Loop
D952  51                C       push    cx                      ;save character repeat counter
D953  56                C       push    si                      ;save source address (font table)
D954  57                C       push    di                      ;save destination addr. (grafix ram)
D955  B9 0004           C       mov     cx,4                    ;init. scanlines per i.a. counter
                        C
D958                    C  g_scan_lp:                   ;Scanline Loop
D958  51                C       push    cx                      ;save scanlines per i.a. counter
D959  57                C       push    di                      ;save interlace area #1 address
D95A  B9 0002           C       mov     cx,2                    ;init. interlace areas counter
                        C
D95D                    C  g_ia_lp:                     ;Interlace Area Loop
D95D  51                C       push    cx                      ;save i.a. counter
D95E  AC                C       lodsb                           ;get a byte from the font table
D95F  8A E0             C       mov     ah,al                   ;copy it
D961  B9 0008           C       mov     cx,8                    ;init. loop counter (8 bits/byte)
```

```
                        C
D964                    C  g_exp_byt:                   ;Expand Byte Loop
D964  D0 EC             C       shr     ah,1                    ;load carry with font byte bit
D966  D1 DB             C       rcr     bx,1                    ;rotate it into expansion accumulator
D968  D0 E8             C       shr     al,1                    ;load carry with same bit as before
D96A  D1 DB             C       rcr     bx,1                    ;double the bit
D96C  E2 F6             C       loop    g_exp_byt       ;expand font byte bits
                        C
D96E  23 DD             C       and     bx,bp                   ;color pixels with foreground color
D970  86 FB             C       xchg    bh,bl                   ;reorder the bytes for grafix ram
D972  0A D2             C       or      dl,dl                   ;is the XOR bit set ?
D974  79 03             C       jns     g_med_store             ;jump if no
D976  26: 33 1D         C       xor     bx,es:[di]              ;XOR with grafix ram
D979                    C  g_med_store:
D979  26: 89 1D         C       mov     es:[di],bx              ;update grafix ram
D97C  81 C7 2000        C       add     di,2000H        ;address next interlace area
D980  59                C       pop     cx                      ;restore i.a. loop counter
D981  E2 DA             C       loop    g_ia_lp                 ;next interlace area
                        C
D983  5F                C       pop     di                      ;restore interlace area #1 address
D984  83 C7 50          C       add     di,80                   ;address next scanline in each i.a.
D987  59                C       pop     cx                      ;restore scanline loop counter
D988  E2 CE             C       loop    g_scan_lp       ;next scanline
                        C
D98A  5F                C       pop     di                      ;restore char's grafix ram address
D98B  47                C       inc     di                      ;address next character in grafix ram
D98C  47                C       inc     di
D98D  5E                C       pop     si                      ;restore char's font table address
D98E  59                C       pop     cx                      ;restore character repeat count
D98F  E2 C1             C       loop    g_char_lp       ;repeat the character
                        C
D991  8A E3             C       mov     ah,bl                   ;return AX with last word written
D993  8A C7             C       mov     al,bh
                        C
D995                    C  g_return:                    ;Return from Write Char
D995  5A                C       pop     dx                      ;restore
D996  59                C       pop     cx                      ;       the
D997  5B                C       pop     bx                      ;           registers
D998  C3                C       ret
                        C
D999                    C  g_color_table label    byte  ;Table of foreground colors extended to byte
                        C
D999  00                C       db      00000000B       ;color 0 (bit pattern: 00)
D99A  55                C       db      01010101B       ;color 1 (bit pattern: 10)
D99B  AA                C       db      10101010B       ;color 2 (bit pattern: 01)
D99C  FF                C       db      11111111B       ;color 3 (bit pattern: 11)
                        C
                        C  grf_graphics_write     endp
                        C
                        C  page
                        C  ;===========================================================================
                        C  ;
                        C  ;    Get offset into graphics ram refresh memory which corresponds to
                        C  ;    the current cursor position (or any arbitrary character position).
                        C  ;
```

```
                        C  ;    Input:   AX  = current cursor position (AL = Column #, AH = Row #)
                        C  ;
                        C  ;    Output:  AX =  offset into graphics ram
                        C  ;
                        C  ;--------------------------------------------------------------------------
                        C
D99D                    C  g_curs_off   proc    near
                        C
D99D  51                C      push     cx                      ;save work register
D99E  8A E8             C      mov      ch,al                   ;hold column number
D9A0  8A C4             C      mov      al,ah                   ;row number to al
                        C
D9A2  B1 01             C      mov      cl,1                    ;mode 72 shift count (multiply * 2)
D9A4  80 3E 0049 R 48   C      cmp      byte ptr ds:[v_mode],72
D9A9  74 02             C      je       g_72                    ;jump if mode 72
D9AB  FE C1             C      inc      cl                      ;mode ~72 shift count (multiply * 4)
D9AD                    C  g_72:
D9AD  D2 E0             C      shl      al,cl                   ;multiply row # by rows per byte
D9AF  32 C9             C      xor      cl,cl                   ;zero out the shift count
D9B1  86 E9             C      xchg     ch,cl                   ;move column number for add
D9B3  F6 26 004A R      C      mul      byte ptr ds:[v_width]   ;multiply by bytes per columnn
D9B7  03 C1             C      add      ax,cx                   ;compute offset into refresh ram
D9B9  59                C      pop      cx                      ;restore register
D9BA  C3                C      ret                              ;and return to caller
                        C
                        C  g_curs_off   endp
                        C
D9BB                    C  code ends

                        C  include pwrup1.asm
                        C
                        C
                        C  ;=========================================================================
                        C  ;    Filename:pwrup1.src
                        C  ;
                        C  ;    This module includes CPU, ROM, 8253 p_dma p_timer, & 8237 p_dma
                        C  ;    Controller tests.
                        C  ;
                        C  ;=========================================================================
                        C
D9BB                    C  code segment  public 'ROM'
                        C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
= 0000                  C  PARITY = 0                                    ;; CONDITIONAL ASSEMBLY
                        C
D9BB                    C  p1_data1      proc    near
                        C
D9BB  90                C  even                                          ; word-align stack_rom
                        C
D9BC  DBE9 R            C  stack_rom     dw      i_rom                   ; return from i_cpu
D9BE  DBEF R            C                dw      i_rom_ret       ; return from rom_checksum
D9C0  DC00 R            C                dw      i_dmat                  ; return from i_rom
D9C2  DC0C R            C                dw      i_dmat_ret              ; return from rtc_chk
D9C4  DC1D R            C                dw      i_dmac                  ; return from i_dmat
D9C6  DC88 R            C                dw      i_dmac_ret              ; return from memtst
```

```
D9C8  DCB0 R                C                dw      i_pic                   ; return from i_dmac
                            C
D9CA  52 65 73 69 64 65 6E  C  banner_m      db      'Resident Diagnostics',CR,LF
      74 20 44 69 61 67 6E  C
      6F 73 74 69 63 73 0D  C
      0A                    C
D9E0  52 65 76 20 31 2E 34  C                db      'Rev 1.43',CR,LF,LF
      33 0D 0A 0A           C
D9EB  00                    C                db      NUL
                            C
D9EC  0D 0A 50 72 69 6D 61  C  bt_m          db      CR,LF,'Primary Boot-Strap...',CR,LF,NUL
      72 79 20 42 6F 6F 74  C
      2D 53 74 72 61 70 2E  C
      2E 2E 0D 0A 00        C
DA06  50 72 69 6D 61 72 79  C  bt_merr               db      'Primary Boot-Strap DISK READ ERROR.',CR,NUL
      20 42 6F 6F 74 2D 53  C
      74 72 61 70 20 44 49  C
      53 4B 20 52 45 41 44  C
      20 45 52 52 4F 52 2E  C
      0D 00                 C
                            C  ; This line must have same number of blanks as the preceding has characters:
DA2B  20 20 20 20 20 20 20  C  bt_spaces     db      '                                       ',CR,NUL
      20 20 20 20 20 20 20  C
      20 20 20 20 20 20 20  C
      20 20 20 20 20 20 20  C
      20 20 20 20 20 20 20  C
      0D 00                 C
                            C
DA50  2A 20 49 6C 6C 65 67  C  ill_m1                db      '* Illegal Interrupt No. ',NUL
      61 6C 20 49 6E 74 65  C
      72 72 75 70 74 20 4E  C
      6F 2E 20 00           C
DA69  68 20 61 74 20 00     C  ill_m2                db      'h at ',NUL
DA6F  20 2A 00              C  ill_m3                db      ' *',NUL
                            C
DA72  20 50 61 73 73 0D 0A  C  pass_m                db      ' Pass',CR,LF,NUL
      00                    C
DA7A  20 46 61 69 6C 00     C  fail_m                db      ' Fail',NUL
                            C
DA80  43 50 55 20 28 69 38  C  i_cpu_m               db      'CPU (i8086) ',NUL; Pass/Fail
      30 38 36 29 20 00     C
DA8D  52 4F 4D 20 4D 6F 64  C  i_rom_m               db      'ROM Module  ',NUL; Pass/Fail
      75 6C 65 20 20 00     C
DA9A  44 4D 41 20 54 69 6D  C  i_dmat_m              db      'DMA Timer   ',NUL; Pass/Fail
      65 72 20 20 20 00     C
DAA7  44 4D 41 20 43 6F 6E  C  i_dmac_m              db      'DMA Control ',NUL; Pass/Fail
      74 72 6F 6C 20 00     C
DAB4  49 6E 74 65 72 72 75  C  i_pic_m               db      'Interrupts  ',NUL; Pass/Fail/Fail:Hx
      70 74 73 20 20 00     C
DAC1  56 69 64 65 6F 20 42  C  i_d_m         db      'Video Board ',NUL; Pass/Fail
      6F 61 72 64 20 00     C
DACE  4E 50 55 20 28 69 38  C  i_npu_m               db      'NPU (i8087) ',NUL; Pass/Fail
      30 38 37 29 20 00     C
DADB  52 54 20 43 6C 6F 63  C  i_rtc_m               db      'RT Clock    ',NUL; Pass/Fail/Fail:LO,HI,NR
      6B 20 20 20 20 00     C
```

```
DAE8  3A 4C 4F 00          C  i_rtc_lo_m    db      ':LO',NUL          ; Error1 (must remain in
DAEC  3A 48 49 00          C  i_rtc_hi_m    db      ':HI',NUL          ; Error2 order for addr.
DAF0  3A 4E 52 00          C  i_rtc_nr_m    db      ':NR',NUL          ; Error3 calculation!)
DAF4  4B 65 79 62 6F 61 72 C  i_kb_m                db      'Keyboard    ',NUL; Pass/Fail/Fail:ST
      64 20 20 20 20 00    C
DB01  3A 53 54 00          C  i_kb_st_m     db      ':ST',NUL
DB05  50 72 69 6E 74 65 72 C  i_prt_m               db      'Printer Port',NUL; Pass/Fail:xx
      20 50 6F 72 74 00    C
DB12  53 65 72 69 61 6C 20 C  i_com_m               db      'Serial Comm.',NUL; Pass/Fail:xx
      43 6F 6D 6D 2E 00    C
DB1F  20 6B 62 20 52 41 4D C  i_RAM_m               db      ' kb RAM  ',NUL   ;Pass/Fail:cc:y000:zzzz:wwww:rrrr
      20 20 00             C
DB29  4F 70 74 69 6F 6E 61 C  i_optROM_m    db      'Optional ROM',NUL; Pass/Fail:xxxx
      6C 20 52 4F 4D 00    C
DB36  46 6C 6F 70 70 79 20 C  i_fduA_m      db      'Floppy (A:) ',NUL; Ready/Not Ready/Fail:xx
      28 41 3A 29 20 00    C
DB43  46 6C 6F 70 70 79 20 C  i_fduB_m      db      'Floppy (B:) ',NUL
      28 42 3A 29 20 00    C
DB50  20 4E 6F 74          C  i_fdu_not_m   db      ' Not'                     ; purposely no NUL!!!!!
DB54  20 52 65 61 64 79 0D C  i_fdu_rdy_m   db      ' Ready',CR,LF,NUL
      0A 00                C
DB5D  46 69 78 65 64 20 44 C  i_hdu_m               db      'Fixed Disk  ',NUL; Pass/Fail:xx
      69 73 6B 20 20 00    C
DB6A  53 65 6C 65 63 74 20 C  i_alt_select_mdb      'Select Alternate CPU (y/n)?',NUL
      41 6C 74 65 72 6E 61 C
      74 65 20 43 50 55 20 C
      28 79 2F 6E 29 3F 00 C
                           C
                           C  ;             db      'Disk Diagnostics xxxxx'
                           C  ;             db      'Loop Diagnostics xxxxx'
                           C  ;             db      'Primary BootStrap Floppy (A:)  Not Ready
                           C  ;             db      'Insert system disk and type any key.'
                           C  ;             db      'Primary BootStrap Floppy (A:)  Fail:xx'
                           C  ;             db      'Primary BootStrap Floppy (B:)  Fail:xx'
                           C  ;             db      'Primary BootStrap Fixed Disk   Fail:xx'
                           C  ;             db      'Select Operating System?
                           C  ;             db      'Serial BootStrap'
                           C
DB86  0F                   C  i_cal_val     db      0Fh     ; port 074h = units of minutes     (0-9)
DB87  07                   C                db      07h     ; port 075h = tens of minutes      (0-5)
DB88  0F                   C                db      0Fh     ; port 076h = units of hours       (0-9)
DB89  03                   C                db      03h     ; port 077h = tens of hours(0-2)
DB8A  0F                   C                db      0Fh     ; port 078h = units of days(0-9)
DB8B  03                   C                db      03h     ; port 079h = tens of days (0-3)
DB8C  07                   C                db      07h     ; port 07Ah = day of week  (0-7)
DB8D  0F                   C                db      0Fh     ; port 07Bh = tens of months       (0-9)
DB8E  01                   C                db      01h     ; port 07Ch = units of months      (0-1)
                           C
                           C  p1_data1      endp
                           C
                           C  ;--------------------------------------------------------------------
                           C
DB8F                       C  diagnostics_1 proc    near
                           C                assume  cs:code, ds:nothing, es:nothing, ss:nothing
                           C
```

```
DB8F  FA              C       cli                             ; disable interrupts
DB90  B0 40           C       mov     al,40h                  ; Check Point0
DB92  BA 0378         C       mov     dx,378h                 ; parallel port data port address
DB95  EE              C       out     dx,al                   ; output " Running- Checkpoint 0"
DB96  FC              C       cld                             ; clear string direction flag
                      C
                      C ;--------------------------------------------------------------------------
                      C ;    8086 CPU Test
                      C ;--------------------------------------------------------------------------
                      C
DB97                  C i_cpu:
                      C
                      C ; Flags Test (All Set):  SF, ZF, AF, PF, CF & OF.
                      C ; (Exercises flags and accumulator only.)
                      C
DB97  B8 FF00         C       mov     ax,0FF00h       ; ah = all 1's; al = all 0's
DB9A  9E              C       sahf                            ; set SF, ZF, AF, PF, & CF
DB9B  79 25           C       jns     i_cpu_err       ; SF set?  if not, abort
DB9D  75 23           C       jnz     i_cpu_err       ; ZF set?  if not, abort
DB9F  7B 21           C       jnp     i_cpu_err       ; PF set?  if not, abort
DBA1  73 1F           C       jnb     i_cpu_err       ; CF set?  if not, abort
                      C
DBA3  37              C       aaa                             ; to test if AF is set, al must be <= 9
                      C                                       ; if AF set, then:  (ah +=1) == 0;
                      C                                       ; al = ((al+6) & 0Fh)== 6; CF = AF == 1
DBA4  73 1C           C       jnb     i_cpu_err       ; CF = AF set?  if not, abort
DBA6  0A E4           C       or      ah,ah                   ; ah = 0?  if not, abort
DBA8  75 18           C       jnz     i_cpu_err
                      C
DBAA  B0 40           C       mov     al,40h                  ; ah = 0
DBAC  02 C0           C       add     al,al                   ; al = 40h + 40h = 80h = -128
DBAE  71 12           C       jno     i_cpu_err       ; OF set?  if not, abort
                      C
                      C ; Flags Test (All Reset):  SF, ZF, AF, PF, CF & OF.
                      C ; (Exercises flags and accumulator only.)
                      C
DBB0  33 C0           C       xor     ax,ax                   ; ax = 0
DBB2  9E              C       sahf                            ; reset SF, ZF, AF, PF, & CF
DBB3  78 0D           C       js      i_cpu_err       ; SF reset?  if not, abort
DBB5  76 0B           C       jbe     i_cpu_err       ; ZF or CF reset?  if not, abort
DBB7  7A 09           C       jp      i_cpu_err       ; PF reset?  if not, abort
                      C
DBB9  37              C       aaa                             ; to test if AF reset, al must be <= 9
                      C                                       ; if AF reset, then:  ah unchanged;
                      C                                       ; al = (al & 0Fh) == 0; CF = AF == 0
DBBA  72 06           C       jb      i_cpu_err       ; CF = AF reset?  if not, abort
DBBC  03 C0           C       add     ax,ax                   ; ax = 0?  if not, abort
DBBE  75 02           C       jnz     i_cpu_err
                      C                                       ; ax = 0 + 0 = 0, so should be no OF.
DBC0  71 13           C       jno     i_cpu_ok        ; OF reset?  if not, abort
                      C
                      C               assume  cs:code, ds:code, es:abs0, ss:code
                      C
DBC2                  C i_cpu_err:
DBC2  8C C8           C       mov     ax,cs                   ; satisfy assumptions
```

```
DBC4  8E D8              C       mov     ds,ax
DBC6  8E D0              C       mov     ss,ax                   ; use ROM 'stack'
DBC8  BC D9BC R          C       mov     sp,cs:(offset stack_rom)
                         C
DBCB  BE DA80 R          C       mov     si,cs:(offset i_cpu_m)
DBCE  B1 01              C       mov     cl,1                    ; Diagnostic checkpoint num. for mfg_tst
DBD0  32 E4              C       xor     ah,ah                   ; clear ah (no error number to report)
DBD2  E9 DF86 R          C       jmp     i_fatal                 ; i_fatal will halt, not 'ret' to i_rom
                         C
DBD5                     C  i_cpu_ok:
DBD5  8C C8              C       mov     ax,cs                   ; satisfy assumptions
DBD7  8E D8              C       mov     ds,ax
DBD9  8E D0              C       mov     ss,ax                   ; use ROM 'stack'
DBDB  BC D9BC R          C       mov     sp,cs:(offset stack_rom)
                         C
DBDE  B0 41              C       mov     al,41h                  ; Check Point1
DBE0  BA 0378            C       mov     dx,378h                 ; parallel port data port address
DBE3  EE                 C       out     dx,al                   ; output " Running- Checkpoint 1"
                         C  ; Reset the keyboard
                         C
DBE4  B0 30              C       mov     al,parity_disable ;reset kb, disable parities
DBE6  E6 61              C       out     p_kctrl,al
DBE8  C3                 C       ret                             ; will 'ret' to i_rom
                         C
                         C  ;--------------------------------------------------------------------------
                         C  ;    ROM Module Test
                         C  ;--------------------------------------------------------------------------
                         C
                         C               assume  cs:code, ds:code, es:abs0, ss:code
                         C
DBE9                     C  i_rom:
                         C
                         C  ; Calculate Checksum of ROM.
                         C
DBE9  BE C000            C       mov     si,0C000h       ; ROM starts at ds:si = F000:C000
DBEC  E9 E566 R          C       jmp     rom_checksum            ; 'call' rom_checksum
DBEF                     C  i_rom_ret:                           ; will 'ret' here
DBEF  74 08              C       jz      i_rom_ok
                         C
                         C
DBF1                     C  i_rom_err:
DBF1  BE DA8D R          C       mov     si,cs:(offset i_rom_m)
DBF4  B1 02              C       mov     cl,2                    ; diagnostic checkpoint num. for mfg_tst
DBF6  E9 DF86 R          C       jmp     i_fatal                 ; ah has illegal checksum
                         C                                       ; i_fatal will halt, not 'ret' to i_dmat
                         C
DBF9                     C  i_rom_ok:
DBF9  B0 42              C       mov     al,42h                  ; Check Point2
DBFB  BA 0378            C       mov     dx,378h                 ; parallel port data port address
DBFE  EE                 C       out     dx,al                   ; output " Running- Checkpoint 2"
DBFF  C3                 C       ret                             ; will 'ret' to i_dmat
                         C
                         C  ;--------------------------------------------------------------------------
                         C  ;    8253 p_dma p_timer Test
                         C  ;--------------------------------------------------------------------------
```

```
                        C
                        C               assume   cs:code, ds:code, es:abs0, ss:code
                        C
DC00                    C  i_dmat:
                        C
                        C  ; Disable 8237A p_dma Controller before the testing of the 8253 p_dma p_timer channel.
                        C
DC00  B0 04             C       mov     al,dma_cmd_disable; disable p_dma controller command
DC02  E6 08             C       out     dma_command,al
                        C
                        C  ; Proceed with the testing of the 8253 p_dma p_timer channel ()p_8253_1.
                        C
DC04  B0 74             C       mov     al,074h          ; 01 11 010 0 -> p_8253_1, lsb 1st, mode 2, no BCD
DC06  BA 0041           C       mov     dx,p_8253_1              ; select p_dma refresh counter
DC09  E9 E165 R         C       jmp     rtc_chk                  ; 'call' rtc_chk
DC0C                    C  i_dmat_ret:                           ; will 'ret' here
DC0C  74 08             C       jz      i_dmat_ok
                        C
                        C
DC0E                    C  i_dmat_err:
DC0E  BE DA9A R         C       mov     si,cs:(offset i_dmat_m)
DC11  B1 03             C       mov     cl,3                     ; diagnostic checkpoint num. for mfg_tst
DC13  E9 DF86 R         C       jmp     i_fatal                  ; ah has error code to report.
                        C                                        ; i_fatal will halt, not 'ret' to i_dmac
                        C
DC16                    C  i_dmat_ok:
DC16  B0 43             C       mov     al,43h                   ; Check Point3
DC18  BA 0378           C       mov     dx,378h                  ; parallel port data port address
DC1B  EE                C       out     dx,al                    ; output " Running- Checkpoint 3"
DC1C  C3                C       ret                              ; will 'ret' to i_dmac
                        C
                        C  ;--------------------------------------------------------------------------
                        C  ;    8237 p_dma Controller Test -- Test Chip's Operation & Channel Registers
                        C  ;
                        C  ;    The 8237A p_dma Controller was disabled before the testing of the
                        C  ;    8253 p_dma p_timer channel.
                        C  ;--------------------------------------------------------------------------
                        C
                        C               assume   cs:code, ds:code, es:abs0, ss:code
                        C
DC1D                    C  i_dmac:
                        C
                        C  ; Send a 'master clear' to 8237 p_dma Controller.
                        C
DC1D  E6 0D             C       out     dma_master_clr,al        ; send master clear to  port
                        C
                        C  ; The dma_command, dma_status, dma_request, dma_temp, and dma_ff registers
                        C  ; are cleared, and the dma_mask register is set (all off).
                        C  ; Test readable control registers:  dma_status & dma_temp)
                        C
DC1F  B4 01             C       mov     ah,1                     ; TEMP Error1
                        C
DC21  E4 0D             C       in      al,dma_temp
DC23  0A C0             C       or      al,al                    ; al = 0?
DC25  75 6E             C       jnz     i_dmac_err               ; if not, abort
```

```
                        C
                        C  ; Test all 8 16-bit readable/writeable channel registers (address and count
                        C  ; registers for all 4 channels, i.e., ports 0 through 7) with register bit test:
                        C  ; (dma_addr_x & dma_count_x are tested with OFFFFh and then 0h for x = 0 to 3.)
                        C
DC27  BB FFFF           C       mov     bx,0FFFFh       ; bx = 0 all bits reset
                        C
DC2A                    C  i_dmac_pass2:                        ; outer loop
                        C                                       ; if 1st pass, bx = OFFFFh
                        C                                       ; if 2nd pass, bx = 0h
DC2A  BA 0007           C       mov     dx,7                    ; loop counter and port address!!!
                        C
DC2D                    C  i_dmac_lp:                           ; inner loop
DC2D  8B C3             C       mov     ax,bx                   ; get bit test pattern
DC2F  EE                C       out     dx,al                   ; write low byte of address/count
DC30  EE                C       out     dx,al                   ; write high byte of address/count
DC31  EC                C       in      al,dx                   ; read low byte of address/count
DC32  8A E0             C       mov     ah,al                   ; save low byte in ah
DC34  EC                C       in      al,dx                   ; read high byte of address/count
                        C
DC35  3B C3             C       cmp     ax,bx                   ; does what's been read = test pattern?
DC37  B4 02             C       mov     ah,2                    ; TEMP Error2
DC39  75 5A             C       jnz     i_dmac_err              ; if not, abort
                        C
DC3B  4A                C       dec     dx                      ; did we decrement past port address 0?
DC3C  79 EF             C       jns     i_dmac_lp       ; if not, continue same pass for all 8.
                        C
DC3E  43                C       inc     bx                      ; 1st pass?  if so, bx = OFFFFh & loop
DC3F  74 E9             C       jz      i_dmac_pass2            ; 2nd pass?  if so, bx = 0 & continue
                        C
                        C  ; We are done testing all 8 16-bit readable/writeable channel registers (address
                        C  ; and count registers for all 4 channels) with the following results:  All the
                        C  ; address registers (dma_addr_x) and count registers (dma_count_x) have been
                        C  ; initialized to zero.
                        C
                        C  ; Load 64k (OFFFFh+1) count for RAM refresh p_dma controller channel.
                        C
DC41  B0 FF             C       mov     al,0FFh
DC43  E6 01             C       out     dma_count_0,al          ; low byte of count for 64k RAM refresh
DC45  E6 01             C       out     dma_count_0,al          ; high byte of count for 64k RAM refresh
                        C
                        C  ; Load mode for RAM refresh p_dma controller channel:  channel 0, read, auto-
                        C  ; initialize, increment, single mode.
                        C
DC47  B0 58             C       mov     al,dma_mode_0           ; mode for RAM refresh
DC49  E6 0B             C       out     dma_mode,al
                        C
                        C  ; Enable p_dma controller:  memory-to-I/O, controller enable, normal, fixed
                        C  ; priority, late write, and DREQ/~DACK.
                        C
DC4B  B0 00             C       mov     al,dma_cmd_enable ; enable p_dma controller
DC4D  E6 08             C       out     dma_command,al
                        C
                        C  ; The master clear command above has masked off all channels. Now, we 'unmask'
                        C  ; the RAM refresh dma_mask bit.  p_dma RAM refresh begins for the first time!
```

```
                        C
DC4F  B0 00             C       mov     al,dma_unmask_0         ; turn on RAM refresh channel 0
DC51  E6 0A             C       out     dma_mask_bit,al
                        C
                        C  ; Program p_8253_1 of i8253 p_timer to proper value for RAM refresh.
                        C
DC53  B0 54             C       mov     al,t3cmd        ; select p_dma refresh counter
DC55  E6 43             C       out     p_8253_ctrl,al
                        C
DC57  B8 0013           C       mov     ax,t1count              ; load p_dma refresh count
DC5A  E6 41             C       out     p_8253_1,al
                        C  ;mov al,ah
                        C  ;out p_8253_1,al
                        C
                        C  ; Check dma_status for 'hot' p_dma request from p_8253_1.
                        C
DC5C  B4 03             C       mov     ah,3                    ; TEMP Error3
DC5E  E4 08             C       in      al,dma_status           ; test for RAM refresh request in status
DC60  A8 10             C       test    al,010h                 ; bit4 -> channel 0 request
DC62  75 31             C       jnz     i_dmac_err              ; if 'hot' p_dma request is there, abort
                        C
                        C  ; Initialize other p_dma counters and modes.
                        C
DC64  BA 000B           C       mov     dx,dma_mode
                        C
                        C  ; Initialize p_dma channel 1 not used.
                        C
DC67  B0 41             C       mov     al,dma_mode_1           ; mode for not used
DC69  EE                C       out     dx,al
                        C
                        C  ; Initialize p_dma channel 2 FDU.
                        C
DC6A  B0 56             C       mov     al,dma_mode_2           ; mode for FDU
DC6C  EE                C       out     dx,al
                        C
                        C  ; Initialize p_dma channel 3 display.
                        C
DC6D  B0 43             C       mov     al,dma_mode_3           ; mode for display
DC6F  EE                C       out     dx,al
                        C
                        C  ; Initialize p_dma Segment Nibble Latches to zero.
                        C  ; (dma_segm_x for x = 0 to 3 is port addresses 80h to 83h).
                        C
DC70  32 C0             C       xor     al,al                   ; al = 0
DC72  BA 0083           C       mov     dx,083h                 ; loop counter and port address!
                        C                                       ; dx = dma_segm_3 = 083h
DC75                    C  i_dmac_nib:
DC75  EE                C       out     dx,al
DC76  FE CA             C       dec     dl                      ; when dl goes from 80h (-128) to
DC78  78 FB             C       js      i_dmac_nib              ; 07Fh (+127) we will exit
                        C
                        C  ;-------------------------------------------------------------------------
                        C  ;    8237 p_dma Controller Test -- Test Lowest 64k bank of RAM
                        C  ;-------------------------------------------------------------------------
                        C
```

```
                        C               assume   cs:code, ds:data, es:abs0, ss:code
                        C
DC7A  2E: 8E 1E E574 R  C       mov     ds,word ptr cs:[set_ds_word]        ; satisfy assumptions
DC7F  8B 36 0072 R      C       mov     si,word ptr ds:[reset_flag]; save reset_flag
                        C
DC83  33 D2             C       xor     dx,dx                       ; dx = 0; test 0000:0 to 0000:FFFF
DC85  E9 E1D4 R         C       jmp     memtst                      ; 'call' memtst
DC88                    C  i_dmac_ret:                              ; will 'ret' here
                        C
DC88  2E: 8E 1E E574 R  C       mov     ds,word ptr cs:[set_ds_word]        ; satisfy assumptions
DC8D  89 36 0072 R      C       mov     word ptr ds:[reset_flag],si; restore reset_flag
                        C
DC91  B4 04             C       mov     ah,4                        ; TEMP Error4
DC93  74 0F             C       jz      i_dmac_ok
                        C
                        C
DC95                    C  i_dmac_err:
DC95  BE DAA7 R         C       mov     si,cs:(offset i_dmac_m)
DC98  0B C9             C       or      cx,cx                       ;; if zero then it was a parity error.
DC9A  75 03             C       jnz     i_d_e                       ;;
DC9C  BE E625 R         C       mov     si,cs:(offset parity1_m);;
DC9F                    C  i_d_e:                                           ;;
DC9F  B1 04             C       mov     cl,4                        ; diagnostic checkpoint num. for mfg_tst
DCA1  E9 DF86 R         C       jmp     i_fatal                     ; ah has error code to report.
                        C                                           ; i_fatal will halt, not 'ret' to i_pic
                        C
DCA4                    C  i_dmac_ok:
DCA4  32 C0             C       xor     al,al                       ; Initialize mfg_tst flag = No Errors
DCA6  A2 0012 R         C       mov     mfg_tst,al
                        C
DCA9  B0 44             C       mov     al,44h                      ; Check Point4
DCAB  BA 0378           C       mov     dx,378h                     ; parallel port data port address
DCAE  EE                C       out     dx,al                       ; output " Running- Checkpoint 4"
DCAF  C3                C       ret                                 ; will 'ret' to i_pic
                        C
                        C  ;---------------------------------------------------------------------------
                        C  ;    8259A Programmable Interrupt Controller Test.
                        C  ;---------------------------------------------------------------------------
                        C
                        C               assume   cs:code, ds:data, es:abs0, ss:stack_ram
                        C
DCB0                    C  i_pic:
DCB0  B8 0030           C       mov     ax,stack_seg                ;Initialize RAM Stack
DCB3  8E D0             C       mov     ss,ax                       ;  on lower tested memory
DCB5  BC 0100           C       mov     sp,100h
                        C
                        C  ;    Initialize & Disable 8259A Programmable Interrupt Controller.
                        C
DCB8  E8 E14B R         C       call    i_pic_init
                        C
                        C  ; Install Interrupt Vectors for diagnostics.
                        C
                        C  ; Install unexpected diagnostic interrupt vectors.
                        C
DCBB  33 F6             C       xor     si,si                               ; es:si = abs0_seg:int00locn
```

```
DCBD  8B FE              C       mov     di,si                           ; es:di = abs0_seg:int00locn
DCBF  B9 01FE            C       mov     cx,(0400h-0004h)/2      ; words from 0:0004h to 0:0400h
                         C
DCC2  B8 DD21 R          C       mov     ax,cs:(offset i_pic_err)  ; store offset i_pic_err
DCC5  AB                 C       stosw
DCC6  8C C8              C       mov     ax,cs                           ; store segment address
DCC8  AB                 C       stosw                                   ; es:di = abs0_seg:int00locn + 4
DCC9  F3/ 26: A5         C       rep     movs    word ptr es:0004h,word ptr es:0000h   ; replicate vector
                         C
                         C
                         C  ; Install software diagnostic interrupt vectors.
                         C
DCCC  B8 DCEE R          C       mov     ax,cs:(offset i_pic_0_ok)  ; ax = offset i_pic_0_ok
DCCF  33 FF              C       xor     di,di                           ; es:di = abs0_seg:int00locn
DCD1  B1 05              C       mov     cl,5                            ; load INT's 0h through 4h.
                         C
DCD3                     C  i_pic_soft:                                  ; ax = (4*x)+(i_pic_0_ok)
DCD3  AB                 C       stosw                                   ; es:di gets offset i_pic_x_ok
DCD4  47                 C       inc     di                              ; skip segment (already = cs)
DCD5  47                 C       inc     di
DCD6  05 0004            C       add     ax,4                            ; i_pic_x_ok are 4 bytes apart
DCD9  E2 F8              C       loop    i_pic_soft                      ; until cx = 0.
                         C
                         C  ; Install hardware diagnostic interrupt vectors.
                         C
DCDB  B8 FF23 R          C       mov     ax,cs:(offset ill_int)          ; ax = offset ill_int
DCDE  BF 0020 R          C       mov     di,es:(offset int08locn)  ; es:di = abs0_seg:int08locn
DCE1  B1 08              C       mov     cl,8                            ; load INT's 8h through Fh.
                         C
DCE3                     C  i_pic_hard:
DCE3  AB                 C       stosw                                   ; es:di gets offset ill_int
DCE4  47                 C       inc     di                              ; skip segment (already = cs)
DCE5  47                 C       inc     di
DCE6  E2 FB              C       loop    i_pic_hard                      ; until cx = 0.
                         C
                         C  ; Test software interrupts first (cl = 0 if error).
                         C
DCE8  B7 09              C       mov     bh,09h                  ; bx has its 8th and 11th bits set.
                         C
                         C                                       ; cx = 0 from loop above.
DCEA  F6 F5              C       div     ch                      ; generate a divide-by-zero INT 00h
DCEC  EB 33              C       jmp     short i_pic_err
DCEE                     C  i_pic_0_ok:
                         C                                       ; bh = 09h. set trap & OF flags in bx
                         C                                       ; (bits 8 and 11 of flags).
DCEE  53                 C       push    bx                      ; put trap flags on stack
DCEF  9D                 C       popf                            ; generate a single-step trap INT 01h
DCF0  EB 2F              C       jmp     short i_pic_err         ; must be 4 bytes long!
DCF2                     C  i_pic_1_ok:
                         C
DCF2  CD 02              C       INT     02h                     ; generate a software interrupt INT 02h
DCF4  EB 2B              C       jmp     short i_pic_err         ; must be 4 bytes long!
DCF6                     C  i_pic_2_ok:
                         C
DCF6  CC                 C       INT     03h                     ; generate a 1-byte break-point INT 03h
```

```
DCF7  90              C       nop                             ; must be 4 bytes long!
DCF8  EB 27           C       jmp     short i_pic_err
DCFA                  C  i_pic_3_ok:
                      C                                       ; OF overflow flag is still set.
DCFA  CE              C       INTO                            ; generate an overflow interrupt INT 04h
DCFB  90              C       nop                             ; must be 4 bytes long!
DCFC  EB 23           C       jmp     short i_pic_err
DCFE                  C  i_pic_4_ok:
                      C
                      C  ; Test hardware interrupts second.
                      C
                      C  ; Test 8259A PIC interrupt mask with test patterns (cl = 0 if error).
                      C
DCFE  B0 01           C       mov     al,1                    ; initialize mask value  = 1
                      C
DD00                  C  i_pic_test:
DD00  E8 E15C R       C       call    i_out_mask              ; output pattern, test input
DD03  75 1C           C       jne     i_pic_err       ; if not same pattern, abort
DD05  D0 D0           C       rcl     al,1                    ; rotate test pattern
DD07  73 F7           C       jnc     i_pic_test              ; test again, if not finished
                      C
DD09  B0 FF           C       mov     al,0FFh                 ; test pattern of all ones
DD0B  E8 E15C R       C       call    i_out_mask              ; output pattern, test input
DD0E  75 11           C       jne     i_pic_err       ; if not same pattern, abort
                      C
                      C  ; Look for 'hot' (active though masked off) PIC interrupts (cl = I if error).
                      C
                      C  ; Enable Interrupts for the very first time!
                      C
DD10  FB              C       sti                                     ; enable interrupts
DD11  33 C9           C       xor     cx,cx                           ; delay awhile, waiting for
DD13  E2 FE           C       loop    $                               ; a 'hot' interrupt.
                      C
DD15  A0 006B R       C       mov     al,byte ptr ds:[intr_flag] ; get the flag from ill_int
DD18  0A C0           C       or      al,al                           ; intr_flag = 0?
DD1A  74 39           C       jz      i_pic_ok                ; if so, we're all done.
                      C
                      C  ; Convert 'hot' interrupt mask (bit pattern) to I (1 to 8 error code).
                      C
DD1C                  C  i_pic_hot:                                   ; cx = 0 from loop above.
DD1C  41              C       inc     cx                              ; increment cx (I+1).
DD1D  D0 D8           C       rcr     al,1                            ; mask's least significant bit.
DD1F  73 FB           C       jnb     i_pic_hot               ; if not set, continue.
                      C                                               ; (exit with cl = 1 to 8.)
                      C
DD21                  C  i_pic_err:                           ; cl = 0 or failing PIC 'hot' active I
DD21  BC 0100         C       mov     sp,100h                 ; re-initialize stack
                      C
DD24  51              C       push    cx                      ; save error code.
                      C
                      C  ; Install Vector Table.              ; set int10locn = code_seg:v_io, and
                      C                                       ; set int1Dlocn = code_seg:v_parms.
DD25  E8 E10F R       C       call    i_vector
                      C
                      C  ; Initialize Video.
```

```
                        C
DD28  E8 E05E R         C       call    i_d_init
                        C
                        C  ; Display error message.
                        C
DD2B  BE DAB4 R         C       mov     si,cs:(offset i_pic_m)
DD2E  E8 DFFA R         C       call    DRomString              ; display failing test message.
                        C
DD31  BE DA7A R         C       mov     si,cs:(offset fail_m)
DD34  E8 DFFA R         C       call    DRomString              ; display fail message.
                        C
DD37  59                C       pop     cx                      ; restore error code.
DD38  0A C9             C       or      cl,cl                   ; cl = 0?
DD3A  74 0E             C       jz      i_pic_no_hot            ; if so, we're done.
                        C
                        C  ; Display 'hot' interrupt number :Hx. (where x is the I from 0 to 7)
                        C
DD3C  E8 E589 R         C       call    DColon                  ; display a colon.
DD3F  B8 0E48           C       mov     ax,(0Eh*100h)+'H' ; display 'hot' interrupt symbol.
DD42  CD 10             C       int     10h
DD44  8A C1             C       mov     al,cl                   ; transfer error code.
DD46  48                C       dec     ax                      ; error code (1 to 8) to (0 to 7) I.
DD47  E8 E5C1 R         C       call    DHexNib                 ; display lowest nibble.
                        C
DD4A                    C  i_pic_no_hot:
DD4A  E8 E57C R         C       call    DCrLf
DD4D  EB 0C             C       jmp     short i_pic_end
                        C
DD4F  B1 05             C       mov     cl,5                    ; diagnostic checkpoint number
DD51  E8 DFEE R         C       call    set_mfg_tst             ; put it in mfg_tst IFF mfg_tst = 0
                        C
DD54  F4                C       hlt
                        C
DD55                    C  i_pic_ok:
DD55  B0 45             C       mov     al,45h                  ; Check Point5
DD57  BA 0378           C       mov     dx,378h                 ; parallel port data port address
DD5A  EE                C       out     dx,al                   ; output " Running- Checkpoint 5"
                        C
DD5B                    C  i_pic_end:
                        C
                        C  ;-------------------------------------------------------------------------
                        C  ;    Install Vector Table.
                        C  ;-------------------------------------------------------------------------
                        C
DD5B  BC 0100           C       mov     sp,100h                 ; re-initialize stack
DD5E  E8 E10F R         C       call    i_vector
                        C
                        C  ;-------------------------------------------------------------------------
                        C  ;    Determine System Configuration from Switches and Initialize Video.
                        C  ;-------------------------------------------------------------------------
                        C
DD61  E8 E05E R         C       call    i_d_init
                        C
                        C  ;-------------------------------------------------------------------------
                        C  ;    Display Passing Error Messages
```

```
                        C  ;--------------------------------------------------------------------------
                        C
DD64                    C  disp_pass:
DD64  B4 0F             C       mov     ah,15                           ; Get Mode function code
DD66  CD 10             C       int     10h                             ; Get current video mode
DD68  32 E4             C       xor     ah,ah                           ; Set Mode function code
DD6A  CD 10             C       int     10h                             ; Re-set mode & clear screen.
                        C
DD6C  BE D9CA R         C       mov     si,cs:(offset banner_m)
DD6F  E8 DFFA R         C       call    DRomString
                        C
DD72  BE DA80 R         C       mov     si,cs:(offset i_cpu_m)
DD75  E8 DFFA R         C       call    DRomString
DD78  BE DA72 R         C       mov     si,cs:(offset pass_m)
DD7B  E8 DFFA R         C       call    DRomString
                        C
DD7E  BE DA8D R         C       mov     si,cs:(offset i_rom_m)
DD81  E8 DFFA R         C       call    DRomString
DD84  BE DA72 R         C       mov     si,cs:(offset pass_m)
DD87  E8 DFFA R         C       call    DRomString
                        C
DD8A  BE DA9A R         C       mov     si,cs:(offset i_dmat_m)
DD8D  E8 DFFA R         C       call    DRomString
DD90  BE DA72 R         C       mov     si,cs:(offset pass_m)
DD93  E8 DFFA R         C       call    DRomString
                        C
DD96  BE DAA7 R         C       mov     si,cs:(offset i_dmac_m)
DD99  E8 DFFA R         C       call    DRomString
DD9C  BE DA72 R         C       mov     si,cs:(offset pass_m)
DD9F  E8 DFFA R         C       call    DRomString
                        C
DDA2  BE DAB4 R         C       mov     si,cs:(offset i_pic_m)
DDA5  E8 DFFA R         C       call    DRomString
DDA8  BE DA72 R         C       mov     si,cs:(offset pass_m)
DDAB  E8 DFFA R         C       call    DRomString
                        C
                        C  ;--------------------------------------------------------------------------
                        C  ;    Size & clear RAM at every 64k byte bank past the lowest 64k.
                        C  ;--------------------------------------------------------------------------
                        C
DDAE                    C  RAM_size_tst:
                        C               assume  cs:code, ds:data, es:abs0, ss:stack_ram
                        C
DDAE  BD 0040           C       mov     bp,64                           ;initialize memory count
                        C
DDB1  8B 36 0072 R      C       mov     si,word ptr ds:[reset_flag]; get warm bootflag
DDB5  81 EE 1234        C       sub     si,01234h       ; si=0 iff CTL ALT DEL sequence.
                        C  ENDIF
DDB9                    C  skip_parity:
                        C
DDB9  E8 E666 R         C       call    disable_parity          ;disable parity flip-flops
                        C
                        C  ; Size the RAM
DDBC  33 FF             C       xor     di,di                   ; offset = 0000h
DDBE  BA 1000           C       mov     dx,1000h        ; start at 1000:0000 (dx keeps segment)
```

```
DDC1                    C  RAM_size_lp:
DDC1  8E C2             C       mov     es,dx                   ; get segment
                        C
DDC3  26: 8B 05         C       mov     ax,word ptr es:[di]     ; read existing ram value
DDC6  F7 D0             C       not     ax                      ; complement it
DDC8  26: 89 05         C       mov     word ptr es:[di],ax     ; write complement back to RAM
                        C
DDCB  26: 8B 1D         C       mov     bx,word ptr es:[di]     ; read back from RAM
                        C
DDCE  3B C3             C       cmp     ax,bx                   ; verify to test for end of RAM
DDD0  F7 D0             C       not     ax                      ; recreate original value
DDD2  75 2C             C       jne     RAM_size_end            ; if verify fails, at end of RAM
                        C
DDD4  26: 89 05         C       mov     word ptr es:[di],ax     ; restore original value back to RAM
                        C
DDD7  0B F6             C       or      si,si                   ; test warm boot flag (now 0 = Warm)
DDD9  74 1D             C       jz      RAM_size_nxt            ; if CTL ALT DEL sequence,
                        C                                       ; don't clear memory
DDDB  E8 E1D4 R         C       call    memtst                  ; test and clear 64k of memory
DDDE  75 44             C       jnz     RAM_error       ; test flag from storage test
                        C
DDE0  83 C5 40          C       add     bp,64                   ; increment size
DDE3  56                C       push    si                      ; save Warm Boot Flag - 1234h
DDE4  B8 0E0D           C       mov     ax,0E0Dh        ; put out a CR
DDE7  CD 10             C       INT     10H
                        C
DDE9  8B C5             C       mov     ax,bp                   ; display tested RAM
DDEB  BB 0003           C       mov     bx,3
DDEE  E8 E5DE R         C       call    DNumW
DDF1  BE DB1F R         C       mov     si,cs:(offset i_RAM_m)
DDF4  E8 DFFA R         C       call    DRomString
                        C
DDF7  5E                C       pop     si                      ; retrieve Warm Boot Flag - 1234h
                        C
DDF8                    C  RAM_size_nxt:                        ; maximum RAM = 640k = 10 * 64k
DDF8  80 C6 10          C       add     dh,10h                  ; next segment
DDFB  80 FE A0          C       cmp     dh,0A0h                 ; top of RAM yet (A000:0000)?
DDFE  72 C1             C       jb      RAM_size_lp             ; if not, continue.
                        C
                        C               assume  cs:code, ds:data, es:abs0, ss:stack_ram
DE00                    C  RAM_size_end:
DE00  0B F6             C       or      si,si                   ; test warm boot flag (0 = Warm)
DE02  74 06             C       jz      RAM_size_end_1          ; if not CTL ALT DEL sequence
DE04  BE DA72 R         C       mov     si,cs:(offset pass_m)   ;   display OK
DE07  E8 DFFA R         C       call    DRomString
                        C
DE0A                    C  RAM_size_end_1:                              ; bx = RAM size/16
DE0A  33 C0             C       xor     ax,ax                   ; ax = 0
DE0C  8E C0             C       mov     es,ax                   ; satisfy assumptions es = 0 = abs0_seg
DE0E  8A C6             C       mov     al,dh                   ; ax = (RAM size/16)/256 = RAM size/4k
DE10  D1 E0             C       shl     ax,1                    ; ax = (RAM size/4k) * 2 = RAM size/2k
DE12  D1 E0             C       shl     ax,1                    ; ax = (RAM size/2k) * 2 = RAM size/1k
                        C
DE14  2E: 8E 1E E574 R  C       mov     ds,word ptr cs:[set_ds_word]    ; restore data segment pointer
DE19  A3 0013 R         C       mov     word ptr ds:[memory_size],ax
```

```
                        C
DE1C  B0 4B             C       mov     al,4bh                  ;Checkpoint 11 (0Bh)
DE1E  BA 0378           C       mov     dx,378h                 ;parallel port data port address
DE21  EE                C       out     dx,al                   ;Output " Running, Checkpoint 11"
                        C
DE22  EB 35             C       jmp short i_cal                 ;Go check clock calendar
                        C
                        C
DE24                    C RAM_error:            ; Error message looks like: Fail:cc:y000:zzzz:wwww:rrrr
                        C                       ; where:  cc = RAM configuration number
                        C                       ;         y000 = Segment of failure  = dx = es
                        C                       ;         zzzz = Offset of failure   = di
                        C                       ;         wwww = Data that was written       = ax
                        C                       ;         rrrr = Data that was read  = bx
                        C
DE24  52                C       push    dx                      ;save failing segment
DE25  1E                C       push    ds                      ;save ds
DE26  50                C       push    ax                      ; save failing test pattern.
                        C
DE27  E8 E57C R         C       call    DCrLf                   ;Carriage Return, Line Feed
DE2A  BE DA7A R         C       mov     si,cs:(offset fail_m)
DE2D  E8 DFFA R         C       call    DRomString              ; display fail message.
                        C
DE30  E8 E589 R         C       call    DColon                  ; display a colon
                        C
DE33  E4 66             C       in      al,sys_conf_a           ; get RAM configuration.
DE35  24 0F             C       and     al,0Fh                  ; mask valid bits.
DE37  E8 E5AD R         C       call    DHexByte        ; display RAM configuration.
                        C
DE3A  E8 E589 R         C       call    DColon                  ; display a colon
                        C
DE3D  8E DA             C       mov     ds,dx                   ; ds = failing segment = dx
DE3F  8B C7             C       mov     ax,di                   ; ax = failing segment = di
DE41  E8 E595 R         C       call    DHexLong        ; display ds:ax
                        C
DE44  E8 E589 R         C       call    DColon                  ; display a colon
                        C
DE47  1F                C       pop     ds                      ; ds = failing test pattern = on stack
DE48  8B C3             C       mov     ax,bx                   ; ax = what was read = bx
DE4A  E8 E595 R         C       call    DHexLong        ; display ds:ax
DE4D  E8 E57C R         C       call    DCrLf
                        C
DE50  1F                C       pop     ds                      ;restore ds
DE51  5A                C       pop     dx                      ;restore failing segment
                        C
DE52  B1 0B             C       mov     cl,0Bh                  ; diagnostic checkpoint number 11
DE54  E8 DFEE R         C       call    set_mfg_tst             ; put it in mfg_tst IFF mfg_tst = 0
                        C
DE57  EB B1             C       jmp     short RAM_size_end_1
                        C
                        C ;--------------------------------------------------------------------------
                        C ;    MM58174 Clock Calendar Device Test
                        C ;--------------------------------------------------------------------------
                        C
DE59                    C i_cal:
```

```
                        C
                        C  ; Read Clock Calendar Device.          ; bx = day (from 1-1 of leap year)
                        C                                         ; ch = hour
DE59  B4 FE             C       mov     ah,-2                     ; cl = minutes
DE5B  CD 1A             C       int     1Ah                       ; dh = seconds
                        C                                         ; dl = hundredths of seconds
                        C  ; Check time & date read.
                        C
                        C  ; Hundredths of Seconds.
                        C
DE5D  B8 000A           C       mov     ax,10                     ; ax = 10; dl = hundredths
DE60  86 C2             C       xchg    al,dl                     ; ax = hundredths; dl = 10
DE62  F6 F2             C       div     dl                        ; ah = remainder, al = quotient
                        C                                         ; (can only read tenths of seconds.)
DE64  3D 000A           C       cmp     ax,10                     ; ah should = 0, and al should be < 10.
DE67  73 15             C       jae     i_cal_1_1_80              ; if not, test chip & write 1-1-80.
                        C
                        C  ; Seconds.
                        C
DE69  80 FE 3C          C       cmp     dh,60                     ; dh = seconds should be < 60.
DE6C  73 10             C       jae     i_cal_1_1_80              ; if not, test chip & write 1-1-80.
                        C
                        C
                        C  ; Minutes.
                        C
DE6E  80 F9 3C          C       cmp     cl,60                     ; cl = minutes should be < 60.
DE71  73 0B             C       jae     i_cal_1_1_80              ; if not, test chip & write 1-1-80.
                        C
                        C  ; Hours.
                        C
DE73  80 FD 18          C       cmp     ch,24                     ; ch = hour should be < 24.
DE76  73 06             C       jae     i_cal_1_1_80              ; if not, test chip & write 1-1-80.
                        C
                        C  ; Days.
                        C
DE78  81 FB 0B6A        C       cmp     bx,(2*366)+(6*365); bx = day from leap year mod 8 should
                        C                                         ;    be < (0-2921) = (0-0B69h)
DE7C  72 5D             C       jb      i_cal_end         ; if so, valid time & day, skip test.
                        C
DE7E                    C  i_cal_1_1_80:                          ; else invalid time & day, write 1-1-80.
                        C
                        C  ; Initialize and Stop Clock.
                        C
DE7E  33 C0             C       xor     ax,ax                     ; ax = 0
DE80  E6 70             C       out     70h,al                    ; test only port = out of test mode
DE82  E6 7E             C       out     7Eh,al                    ; stop/start port = stop clock
                        C
                        C  ; Output test pattern of maximum value with all bits set to read/writable ports.
                        C
DE84  BE DB86 R         C       mov     si,cs:(offset i_cal_val)
DE87  B9 0009           C       mov     cx,9                      ; ch keeps 0; cx = 9
DE8A  BA 0074           C       mov     dx,0074h          ; dh keeps 0; dx = 74h
                        C
DE8D                    C  i_cal_max:
DE8D  2E: AC            C       lods    byte ptr cs:[si]  ; al get cs:si (ds overridden!).
```

```
DE8F  8A E0             C       mov     ah,al                   ; save maximum value.
DE91  EE                C       out     dx,al                   ; ports 74 through 7C (units of minutes
                        C                                       ; to tens of months) get max value.
DE92  EC                C       in      al,dx                   ; read it back.
DE93  22 C4             C       and     al,ah                   ; mask valid bits.
DE95  3A C4             C       cmp     al,ah                   ; is it equal to the value written?
DE97  75 28             C       jnz     i_cal_err       ; if not, abort.
                        C
DE99  42                C       inc     dx                      ; increment to next port
DE9A  E2 F1             C       loop    i_cal_max
                        C
DE9C  B0 07             C       mov     al,07h
DE9E  B2 7F             C       mov     dl,7Fh                  ; dh kept 0; dx = 7Fh
DEA0  EE                C       out     dx,al                   ; interrupt (year mod 8) gets 0Fh.
DEA1  EC                C       in      al,dx                   ; must do 'in' from this port 3 times.
DEA2  EC                C       in      al,dx
DEA3  EC                C       in      al,dx
DEA4  24 07             C       and     al,07h                  ; mask valid bits.
DEA6  2C 07             C       sub     al,07h                  ; is it equal to the value written?
DEA8  75 17             C       jnz     i_cal_err       ; if not, abort.
                        C
                        C ; Write out 0h (bits of lower nibble reset) test pattern to read/writable ports.
                        C
                        C                                       ; al kept 0h.
DEAA  B1 09             C       mov     cl,9                    ; ch kept 0; cx = 9
DEAC  B2 74             C       mov     dl,74h                  ; dh kept 0; dx = 74h
DEAE                    C i_cal_0:
DEAE  EE                C       out     dx,al                   ; ports 74 through 7C (units of minutes
                        C                                       ; to tens of months) get 0h.
DEAF  EC                C       in      al,dx                   ; read it back.
DEB0  24 0F             C       and     al,0Fh                  ; mask valid bits (lower nibble) = 0h?
DEB2  75 0D             C       jnz     i_cal_err       ; if not, abort.
                        C
DEB4  42                C       inc     dx                      ; increment to next port
DEB5  E2 F7             C       loop    i_cal_0
                        C
DEB7  B2 7F             C       mov     dl,7Fh                  ; dh kept 0; dx = 7Fh
DEB9  EE                C       out     dx,al                   ; interrupt (year mod 8) gets 0Fh.
DEBA  EC                C       in      al,dx                   ; must do 'in' from this port 3 times.
DEBB  EC                C       in      al,dx
DEBC  EC                C       in      al,dx
DEBD  24 0F             C       and     al,0Fh                  ; mask valid bits (lower nibble) = 0h?
DEBF  74 12             C       jz      i_cal_ok        ; if so, we're all done.
                        C                                       ; else, abort.
DEC1                    C i_cal_err:
DEC1  BE DADB R         C       mov     si,cs:(offset i_rtc_m)
DEC4  E8 DFFA R         C       call    DRomString
DEC7  BE DA7A R         C       mov     si,cs:(offset fail_m)
DECA  E8 DFFA R         C       call    DRomString              ; display fail message.
DECD  E8 E57C R         C       call    DCrLf
DED0  EB 01             C       jmp     short i_cal_ok          ; try to write 1-1-80, regardless...
                        C
DED2  F4                C       hlt
                        C
DED3                    C i_cal_ok:
```

```
                        C
DED3  33 DB             C       xor     bx,bx                   ; 1-1-80 is day 0.
DED5  33 C9             C       xor     cx,cx                   ; hours & minutes = 0.
                        C
                        C  ; Write & Start Clock Calendar Device.
                        C                                       ; bx = day (from 1-1 of leap year)
DED7  B4 FF             C       mov     ah,-1                   ; ch = hour
DED9  CD 1A             C       int     1Ah                     ; cl = minutes
                        C                                       ; Output: ah = -1 implies date/time err.
                        C                                       ;          ah = 0  implies date/time OK.
                        C
DEDB                    C  i_cal_end:
                        C
                        C  ;--------------------------------------------------------------------------
                        C  ;    i8253 Real-Time Time Clock Test (p_8253_1 tested in i_dmat)
                        C  ;--------------------------------------------------------------------------
                        C
                        C               assume  cs:code, ds:data, es:abs0, ss:stack_ram
                        C
DEDB                    C  i_rtc:
                        C
                        C  ; String to be displayed regardless of results.
                        C
DEDB  BE DADB R         C       mov     si,cs:(offset i_rtc_m)
DEDE  E8 DFFA R         C       call    DRomString
                        C
                        C  ; Test i8253 real-time clock interrupt p_timer counter (p_8253_0).
                        C
DEE1  B0 34             C       mov     al,034h         ; 00 11 010 0 -> p_8253_0, lsb 1st, mode 2, no BCD
DEE3  BA 0040           C       mov     dx,p_8253_0     ; select real-time clock counter
DEE6  E8 E165 R         C       call    rtc_chk
DEE9  75 2E             C       jnz     i_rtc_err; if nz, ah has error code to report.
                        C
                        C  ; Test i8253 tone generator p_timer counter (p_8253_2).
                        C
DEEB  B0 31             C       mov     al,31h          ; clear kb intrpts, reset kb, disable all parity
DEED  E6 61             C       out     p_kctrl,al      ; turn off speaker data -- bit1
                        C                               ; turn on  speaker gate to p_8253_2 -- bit0
                        C
DEEF  B0 B4             C       mov     al,0B4h         ; 10 11 010 0 -> p_8253_2, lsb 1st, mode 2, no BCD
DEF1  BA 0042           C   mov     dx,p_8253_2     ; select
tone generator counter
DEF4  E8 E165 R         C       call    rtc_chk
                        C
DEF7  B0 30             C       mov     al,30h          ;clr kb interrupts, reset kb, disable all parity
DEF9  E6 61             C       out     p_kctrl,al      ; turn off speaker data & gate -- bits1 &0
                        C
DEFB  75 1C             C       jnz     i_rtc_err; if nz, ah has error code to report.
                        C
                        C  ; Initialize i8253 real-time clock interrupt p_timer counter (p_8253_0).
                        C
DEFD  B0 36             C       mov     al,t0cmd        ; select real time clock counter
DEFF  E6 43             C       out     p_8253_ctrl,al
                        C
DF01  B8 0000           C       mov     ax,t0count              ; load real time clock count
```

```
DF04  E6 40               C       out     p_8253_0,al
DF06  8A C4               C       mov     al,ah
DF08  E6 40               C       out     p_8253_0,al
                          C
                          C  ; Initialize i8253 tone generator p_timer counter (p_8253_2).
                          C
DF0A  B0 B6               C       mov     al,t2cmd        ; select tone generator counter
DF0C  E6 43               C       out     p_8253_ctrl,al
                          C
DF0E  B8 0266             C       mov     ax,t2count              ; load tone generator count
DF11  E6 42               C       out     p_8253_2,al
DF13  8A C4               C       mov     al,ah
DF15  E6 42               C       out     p_8253_2,al
                          C
DF17  EB 21               C       jmp     short i_rtc_ok
                          C
                          C
DF19                      C  i_rtc_err:                           ; ah has error code to report.
DF19  BE DA7A R           C       mov     si,cs:(offset fail_m)
DF1C  E8 DFFA R           C       call    DRomString              ; display fail message.
                          C
DF1F  BE DAE4 R           C       mov     si,cs:(offset i_rtc_lo_m-(4*1))
DF22  8A C4               C       mov     al,ah                   ; al = error code = (1, 2 or, 3)
DF24  32 E4               C       xor     ah,ah                   ; ax = error code = (1, 2 or, 3)
DF26  D1 E0               C       shl     ax,1                    ; ax = 2*(error code) = (2, 4, or 6)
DF28  D1 E0               C       shl     ax,1                    ; ax = 4*(error code) = (4, 8, or 12)
DF2A  03 F0               C       add     si,ax                   ; index to (LO, HI, or NR message.)
DF2C  E8 DFFA R           C       call    DRomString              ; display failing mode.
DF2F  E8 E57C R           C       call    DCrLf
                          C
DF32  B1 08               C       mov     cl,8                    ; diagnostic checkpoint number
DF34  E8 DFEE R           C       call    set_mfg_tst             ; put it in mfg_tst IFF mfg_tst = 0
                          C
DF37  EB 0D               C       jmp     short i_rtc_end
                          C
DF39  F4                  C       hlt
                          C
DF3A                      C  i_rtc_ok:
DF3A  BE DA72 R           C       mov     si,cs:(offset pass_m)
DF3D  E8 DFFA R           C       call    DRomString
                          C
DF40  B0 48               C       mov     al,48h                  ; Check Point8
DF42  BA 0378             C       mov     dx,378h                 ; parallel port data port address
DF45  EE                  C       out     dx,al                   ; output " Running- Checkpoint 8"
                          C
DF46                      C  i_rtc_end:
                          C
                          C  ;--------------------------------------------------------------------------
                          C  ;
                          C  ; If present, test 8087 NPU  and display PASS or FAIL message
                          C  ;
                          C  ;--------------------------------------------------------------------------
                          C
DF46  80 26 0098 R FE     C       and     byte ptr ds:[disable_nmi_flag],0FEh ;assume we will enable
DF4B  E4 66               C       in      al,sys_conf_a                       ;read port 66h (switches)
```

```
DF4D  A8 10              C       test    al,switch_8087                  ;test 8087 switch bit
DF4F  74 2D              C       jz      diag1_end               ;if no 8087, we're through
                         C
DF51  BE DACE R          C       mov     si,cs:(offset i_npu_m)  ;display 'NPU (i8087)' message
DF54  E8 DFFA R          C       call    DRomString
                         C
DF57  DB E3              C       esc     28,bx                   ;FNINIT to 8087 (bx) = irrelevant
DF59  33 C0              C       xor     ax,ax
DF5B  A3 0089 R          C       mov     control,ax              ;delay for 8087 initialization
                         C
DF5E  D9 3E 0089 R       C       esc     15,control              ;FNSTCW if 8087 is present
DF62  0B 06 0089 R       C       or      ax,control
DF66  74 08              C       jz      npu_err                 ;if zeros, 8087 not here
                         C
DF68  BE DA72 R          C       mov     si,cs:(offset pass_m)   ;display PASS message
DF6B  E8 DFFA R          C       call    DRomString
DF6E  EB 0E              C       jmp short diag1_end
                         C
DF70                     C  npu_err:
DF70  80 0E 0098 R 01    C       or      byte ptr ds:[disable_nmi_flag],1 ;disable nmi
DF75  BE DA7A R          C       mov     si,cs:(offset fail_m)   ;display FAIL message
DF78  E8 DFFA R          C       call    DRomString
DF7B  E8 E57C R          C       call    DCrLf                   ;Carriage Return, Line Feed
                         C
                         C  ;--------------------------------------------------------------------------
                         C  ; End of power up diagnostics - beep and go to initialization routine
                         C  ;--------------------------------------------------------------------------
                         C               assume  cs:code, ds:nothing, es:nothing, ss:stack_ram
DF7E                     C  diag1_end:
DF7E  B8 0E07            C       mov     ax,(0Eh*100h)+BEL ; beep keyboard
DF81  CD 10              C       int     10h
DF83  E9 E252 R          C       jmp     pcinit
                         C
                         C  diagnostics_1 endp
                         C
                         C  ;-----------------------------------------------------------------
                         C  ;    Fatal Error Routine.
                         C  ;
                         C  ;    Input:  cs:si   = points to offset of failing error message
                         C  ;            cl = diagnostic checkpoint number to store in mfg_tst (40:0012h)
                         C  ;            if ah <> 0, do DHexByte of ah.
                         C  ;            if ah =  0, do nothing (just print error).
                         C  ;    Output: None.
                         C  ;
                         C  ;    Trash:  al, dx, & si destroyed.
                         C  ;-----------------------------------------------------------------
                         C
DF86                     C  i_fatal      proc    near
                         C               assume  cs:code, ds:data, es:nothing, ss:nothing
                         C
                    C  ; Satisfy assumptions and set diagnostic checkpoint number in mfg_tst
                         C
DF86  2E: 8E 1E E574 R   C       mov     ds,word ptr cs:[set_ds_word] ;assure ds set correctly
                         C
DF8B  88 0E 0012 R       C       mov     mfg_tst,cl              ;checkpoint number of failing diagnostic
```

```
                        C
                        C  ; Disable 8237A p_dma Controller.
                        C
DF8F  B0 04             C       mov     al,dma_cmd_disable; disable p_dma controller command
DF91  E6 08             C       out     dma_command,al
                        C
                        C  ; Send a 'master clear' to 8237 p_dma Controller.
                        C
DF93  E6 0D             C       out     dma_master_clr,al ; send master clear port any garbage
                        C
                        C  ; Load 64k (0FFFFh+1) count for RAM refresh p_dma controller channel.
                        C
DF95  B0 FF             C       mov     al,0FFh
DF97  E6 01             C       out     dma_count_0,al          ; low byte of count for 64k RAM refresh
DF99  E6 01             C       out     dma_count_0,al          ; high byte of count for 64k RAM refresh
                        C
                        C  ; Load mode for RAM refresh p_dma controller channel:  channel 0, read, auto-
                        C  ; initialize, increment, single mode.
                        C
DF9B  B0 58             C       mov     al,dma_mode_0           ; mode for RAM refresh
DF9D  E6 0B             C       out     dma_mode,al
                        C
                        C  ; Enable p_dma controller:  memory-to-I/O, controller enable, normal, fixed
                        C  ; priority, late write, and DREQ/~DACK.
                        C
DF9F  B0 00             C       mov     al,dma_cmd_enable ; enable p_dma controller
DFA1  E6 08             C       out     dma_command,al
                        C
                        C  ; The master clear command above has masked off all channels. Now, we 'unmask'
                        C  ; the RAM refresh dma_mask bit.  p_dma RAM refresh begins for the first time!
                        C
DFA3  B0 00             C       mov     al,dma_unmask_0         ; turn on RAM refresh channel 0
DFA5  E6 0A             C       out     dma_mask_bit,al
                        C
                        C  ; Program p_8253_1 of i8253 p_timer to proper value for RAM refresh.
                        C
DFA7  B0 74             C       mov     al,t1cmd        ; select p_dma refresh counter
DFA9  E6 43             C       out     p_8253_ctrl,al
                        C
DFAB  B0 13             C       mov     al,t1count              ; load p_dma refresh count
DFAD  E6 41             C       out     p_8253_1,al
DFAF  32 C0             C       xor     al,al
DFB1  E6 41             C       out     p_8253_1,al
                        C
                        C               assume  cs:code, ds:nothing, es:nothing, ss:stack_ram
                        C
DFB3  8C D7             C       mov     di,ss                   ; save stack pointer
DFB5  8B EC             C       mov     bp,sp
DFB7  BA 0030           C       mov     dx,stack_seg
DFBA  8E D2             C       mov     ss,dx
DFBC  BC 0100           C       mov     sp,100h
                        C
DFBF  50                C       push    ax                      ; save error code
                        C
                        C  ; Initialize & Disable 8259A Programmable Interrupt Controller.
```

```
                        C
DFC0  E8 E14B R         C       call    i_pic_init
                        C
                        C  ; Install Vector Table.              ; set int10locn = code_seg:v_io, and
                        C                                       ; set int1Dlocn = code_seg:v_parms.
DFC3  E8 E10F R         C       call    i_vector
                        C
                        C  ; Initialize Video.
                        C
DFC6  E8 E05E R         C       call    i_d_init
                        C
                        C  ; Display error message.
                        C
DFC9  58                C       pop     ax                      ; restore error code
                        C
DFCA  E8 DFFA R         C       call    DRomString              ; display string at cs:si.
                        C
DFCD  BE DA7A R         C       mov     si,cs:(offset fail_m)
DFD0  E8 DFFA R         C       call    DRomString              ; display fail message.
                        C
DFD3  0A E4             C       or      ah,ah                   ; ah = 0?
DFD5  74 08             C       jz      i_fatal_ret             ; if so, no arguments
                        C
DFD7  E8 E589 R         C       call    DColon                  ; display a colon
                        C
DFDA  8A C4             C       mov     al,ah                   ; display error code
DFDC  E8 E5AD R         C       call    DHexByte
                        C
DFDF                    C  i_fatal_ret:
DFDF  E8 E57C R         C       call    DCrLf
                        C
                        C  ;Output fatal error status for manufacturing tests
                        C
DFE2  BA 0378           C       mov     dx,378h                 ; parallel port address
DFE5  EC                C       in      al,dx                   ; read last checkpoint value
DFE6  34 3F             C       xor     al,03fh                 ; extract checkpoint number from status
DFE8  EE                C       out     dx,al                   ; output " Not OK - number"
                        C
                        C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
DFE9  8E D7             C       mov     ss,di                   ; restore stack pointer
DFEB  8B E5             C       mov     sp,bp
                        C  ;    ret
                        C
DFED  F4                C       hlt
                        C
                        C  i_fatal      endp
                        C
                        C  ;--------------------------------------------------------------------------;
                        C  ; Set the Manufacturing Test Diagnostic Flag ( mfg_tst at 0040:0012h) to
                        C  ; the diagnostic checkpoint number of the first failing diagnostic (or
                        C  ; it will be zero if there are no diagnostic failures during power up tests).
                        C  ;
                        C  ;    cl = the diagnostic checkpoint number
                        C  ;
```

```
                        C ;--------------------------------------------------------------------------------
                        C
DFEE                    C set_mfg_tst   proc    near
                        C
                        C               assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
DFEE  80 3E 0012 R 00   C       cmp     mfg_tst,0               ; Is the flag clear (no error) ?
DFF3  75 04             C       jnz     mfg_tst_ret                     ; if not clear, just return
DFF5  88 0E 0012 R      C       mov     mfg_tst,cl                      ; else set to checkpoint number
DFF9                    C mfg_tst_ret:
DFF9  C3                C       ret
                        C
                        C set_mfg_tst   endp
                        C
                        C
                        C ;============================================================================
                        C ;    Display ASCII String Utilities
                        C ;
                        C ;    Calls TstVid to test for no video controller
                        C ;    board, and suppress the message if the there is no display.
                        C ;
                        C ;============================================================================
                        C
DFFA                    C DRomString    proc    near    ; Displays NUL terminated string at cs:si
                        C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
DFFA  50                C       push    ax              ; save working register
DFFB  E8 E020 R         C       call    TstVid  ; is there a video controller ?
DFFE  72 07             C       jc      DRomExit ; if carry, no video in system: goto exit
E000  1E                C       push    ds
E001  0E                C       push    cs              ; ds gets cs
E002  1F                C       pop     ds
E003  E8 E009 R         C       call    DString ; registers saved in DString
E006  1F                C       pop     ds              ; restore ds
E007                    C DRomExit:
E007  58                C       pop     ax              ; restore AX
E008  C3                C       ret
                        C DRomString    endp
                        C
                        C
                        C
                        C ;============================================================================
                        C
E009                    C DString               proc    near    ; Displays NUL terminated string at ds:si
                        C
                        C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E009  50                C       push    ax              ; all registers & flags saved
E00A  53                C       push    bx
E00B  56                C       push    si
E00C  9C                C       pushf
E00D  FC                C       cld                     ; auto increment
E00E  B3 01             C       mov     bl,1            ; select foreground color for grafix modes
E010  AC                C DS_lp:        lodsb                   ; al gets ds:si and si++
E011  0A C0             C       or      al,al           ; NUL ?
```

```
E013  74 06               C       je      DS_ret
E015  B4 0E               C       mov     ah,0Eh          ; tty emulator
E017  CD 10               C       INT     10h
E019  EB F5               C       jmp     short DS_lp
E01B                      C  DS_ret:
E01B  9D                  C       popf                    ; restore registers & flags
E01C  5E                  C       pop     si
E01D  5B                  C       pop     bx
E01E  58                  C       pop     ax
E01F  C3                  C       ret
                          C  DString              endp
                          C
                          C  ;=============================================================================;
                          C  ;    Tests for switch setting for E.G.A., or no video controller
                          C  ;    board (Switches 5,6 = ON, ON at 7W on motherboard). (Port 67h
                          C  ;    bits 2,3 = 00).
                          C  ;
                          C  ;    Output:
                          C  ;            Carry Flag = SET if the switches say no display and
                          C  ;            the INT 10h vector has not been replaced by the EGA ROM BIOS.
                          C  ;
                          C  ;            Carry Flag = Reset if switches are not ON, ON
                          C  ;=============================================================================
                          C
E020                      C  TstVid       proc    near
                          C
                          C               assume  cs:code, ds:nothing, es:abs0, ss:nothing
                          C
E020  06                  C       push    es              ; save to restore at conclusion
E021  50                  C       push    ax              ; save working register
E022  33 C0               C       xor     ax,ax           ; zeros to reference...
E024  8E C0               C       mov     es,ax           ; abs0 segment
E026  E4 67               C       in      al,sys_conf_b   ; get switch settings for port 67h
E028  24 30               C       and     al,030h         ; isolate video specification
E02A  A8 30               C       test    al,030h         ; if not zeros,
E02C  75 0A               C       jnz     TstVidOK ; OK to display
E02E  26: A1 0040 R       C       mov     ax,word ptr es:int10locn ; if zeros, get INT10h vector
E032  3D F065 R           C       cmp     ax,offset v_io  ; still set to ROM BIOS INT 10h ?
E035  F9                  C       stc                     ; Set Carry Flag = no video controller
E036  74 01               C       je      TstVidExit      ; if no video in system goto exit
                          C
E038                      C  TstVidOK:
E038  F8                  C       clc                     ; clear (reset) carry = video present
                          C
E039                      C  TstVidExit:
E039  58                  C       pop     ax              ; restore AX
E03A  07                  C       pop     es              ; restore es
E03B  C3                  C       ret
                          C
                          C  TstVid       endp
                          C
                          C  ;=============================================================================
                          C  ;
                          C  ;    Reinitialize int0Dlocn, int13locn, int19locn
                          C  ;
```

```
                        C  ;    Saves and restores DS
                        C  ;
                        C  ;    DESTROYS: DI, SI, AX, ES, CL
                        C  ;
                        C  ;
                        C  ;=============================================================================
                        C
E03C                    C  re_i_vector  proc    near
                        C
E03C  FA                C       cli                                     ;can't have interrupts!
                        C
                        C               assume  cs:code, ds:code, es:abs0, ss:stack_ram
                        C
E03D  1E                C       push    ds                              ;save dataseg to restore at end
                        C
E03E  BF 0034 R         C       mov     di,offset int0dlocn             ; offset for INT 0Dh vector
E041  33 C0             C       xor     ax,ax                           ; zeros for abs0 segment
E043  8E C0             C       mov     es,ax                           ; es:di = abs0_seg:int0Dlocn
E045  8C C8             C       mov     ax,cs                           ; code segment
E047  8E D8             C       mov     ds,ax                           ; ds = ax = cs
E049  BE FEFD R         C       mov     si,cs:(offset i_vec_tbl+10); ds:si = code_seg:i_vec_tbl(5)
E04C  B1 03             C       mov     cl,3                            ; loop count value
                        C
E04E                    C  re_init_vec:  ;reinitialize int0Dlocn, int13locn, int19locn
                        C
E04E  A5                C       movsw                                   ; es:di++ gets ds:si++ (offset)
                        C
E04F  AB                C       stosw                                   ; es:di++ gets ax = cs (segment)
                        C
E050  83 C6 0A          C       add     si,10                           ; ds:si ++ 5 words
E053  83 C7 14          C       add     di,20                           ; es:di ++ 10 words
                        C
E056  E2 F6             C       loop    re_init_vec
                        C
E058  1F                C       pop     ds                              ;restore data segment
                        C
E059  FB                C       sti                                     ;re-enable interrupts
                        C
E05A  C3                C       ret
                        C
                        C  re_i_vector  endp
                        C
E05B                    C  code ends
                        C

                        C  include pwrup1a.asm
                        C
                        C
                        C  ;=============================================================================
                        C  ;    Filename:pwrup1a.src
                        C  ;
                        C  ;    This module sets variables which determine the system
                        C  ;    configuration, based on switch settings, as well as
                        C  ;    enabling the video portion.  It also includes temporary
                        C  ;    initialization of the Interrupt Vector, initalization of
```

```
                        C  ;    the PIC, and checks on the RTC and RAM.
                        C  ;
                        C  ;=============================================================================
                        C
E05B                    C  code segment  public 'ROM'
                        C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C       XORG     0E05Bh                    ;;
E05B                    C1      org      0E05Bh
                        C
E05B                    C  i_hard_reset  proc                      ;;
E05B  E9 DB8F R         C       jmp      diagnostics_1             ;;
                        C  i_hard_reset  endp                      ;;
                        C
                        C  ;-----------------------------------------------------------------------------
                        C  ;    Determine System Configuration from Switches and Enable Video.
                        C  ;
                        C  ;    Input:   None.
                        C  ;    Output:  None.
                        C  ;
                        C  ;    Trash:   ax & cx destroyed.
                        C  ;-----------------------------------------------------------------------------
                        C
E05E                    C  i_d_init      proc     near
                        C                assume   cs:code, ds:nothing, es:nothing, ss:stack_ram
                        C
                        C  ;; Here is the code for putting the DEB in Transparent Mode.
                        C
E05E  50                C       push     ax
E05F  52                C       push     dx
E060  BA 03DD           C       mov      dx,03DDh          ;; DEB I/O Address Register port address
E063  B0 01             C       mov      al,1                      ;; select Mode Control Register
E065  EE                C       out      dx,al
                        C
E066  42                C       inc      dx                        ;; DEB Mode Control Register address
E067  42                C       inc      dx
E068  FE C8             C       dec      al                        ;; set DEB Transparent Mode
E06A  EE                C       out      dx,al                     ;;
E06B  5A                C       pop      dx
E06C  58                C       pop      ax
                        C
                        C  ; Initialize data segment
                        C
                        C                assume   cs:code, ds:data, es:nothing, ss:stack_ram
                        C
E06D  1E                C       push     ds                                ; save register
                        C
E06E  B8 0040           C       mov      ax,data_seg                       ; ds = ax = data_seg = 0040h.
E071  8E D8             C       mov      ds,ax                             ; (ah = 0.)
                        C
                        C  ; Initialize monochrome board.
                        C
E073  B0 30             C       mov      al,30h                            ; switch_bits for monochrome.
E075  A3 0010 R         C       mov      word ptr ds:[switch_bits],ax      ; set data for monochrome.
E078  CD 10             C       INT      10h                               ; ah = 0 = v_set_mode.
```

```
                        C
                        C  ; Initialize color board.
                        C
                        C  ;mov ax,0003h                    ;switch_bits for not monochrome.
                        C  ;mov word ptr ds:[switch_bits],ax        ; set data for color.
                        C  ;INT 10h                                 ; ah = 0 = v_set_mode.
                        C
                        C  ; Determine system configuration from switches (low byte of switch_bits).
                        C
E07A  E4 66             C       in      al,sys_conf_a                       ; Read port 66h.
E07C  24 10             C       and     al,010h                             ; Keep 8087 bit only.
E07E  D0 E8             C       shr     al,1                                ; Move to bit one.
E080  D0 E8             C       shr     al,1
E082  D0 E8             C       shr     al,1
E084  8A C8             C       mov     cl,al                       ; cl now contains 8087 bit for
                        C                                           ; high nibble of switch_bits
                        C
E086  E4 67             C       in      al,sys_conf_b               ; read Port 67h
                        C                                           ; system configuration switches.
                        C                                           ; bits7 -6:       (number of FDU's)-1
                        C                                           ; bits5 -4:       monitor type
E088  24 F0             C       and     al,0F0h                     ; mask off low nibble (keep high nibble)
                        C                                           ; of low byte: clear high byte.
E08A  0C 0D             C       or      al,00Dh                     ; ALWAYS 64k planar RAM and >= 1 FDU!
                        C
                        C                                           ;  plus indication of 64K planar RAM
                        C
E08C  0A C8             C       or      cl,al                       ; get data from other switches.
                        C
E08E  32 ED             C       xor     ch,ch                       ; clear for setting switch_bits
E090  89 0E 0010 R      C       mov     word ptr ds:[switch_bits].cx ;initialize configuration bits
E094  B5 03             C       mov     ch,03h                      ;initialize display to mode 3 (default)
                        C
E096  24 30             C       and     al,030h                     ;isolate display switches(bits5 &4).
E098  74 28             C       jz      i_d_ck_EGA                  ;if zero, go check for EGA board
                        C
                        C  ; Initialize Pointer to Master Table for indigenous video.
                        C
E09A  C7 06 0084 R E23C R C       mov     word ptr ds:[master_tbl_ptr+0000h].cs:(offset mastab)
E0A0  8C 0E 0086 R      C       mov     word ptr ds:[master_tbl_ptr+0002h],cs
                        C
                        C  ; Determine if Monochrome Adapter or Color Adapter
                        C
E0A4  3C 30             C       cmp     al,030h                     ; is it the monochrome board?
E0A6  75 4B             C       jne     i_d_ok                      ; if not, 80x25 or 40x25 color
                        C
                        C  ; The switches say it's the Monochrome Adapter
                        C
                        C  ; Initialize monochrome board.
                        C
E0A8  CD 10             C       INT     10h                                 ; ah = 0 = v_set_mode.
                        C
                        C               assume  cs:code, ds:v_ram, es:nothing, ss:stack_ram
E0AA                    C  i_d_mono:
E0AA  1E                C       push    ds                                  ; save ds = data_seg = 0040h.
```

```
E0AB  B8 B000            C       mov     ax,para_mono                    ; satisfy assumptions
E0AE  8E D8              C       mov     ds,ax
E0B0  B0 A5              C       mov     al,0A5h                         ; test pattern
E0B2  A2 0000            C       mov     byte ptr ds:[0000h],al          ; if so, is monochrome there?
E0B5  8A 26 0000         C       mov     ah,byte ptr ds:[0000h]          ; read monochrome RAM
E0B9  1F                 C       pop     ds                              ; restore ds = data_seg = 0040h.
                         C
                         C               assume  cs:code, ds:data, es:nothing, ss:stack_ram
                         C
                         C ; Test for Monochrome Adapteur RAM present
                         C
E0BA  3A C4              C       cmp     al,ah                           ; is monochrome RAM there ?
E0BC  75 2B              C       jne     i_d_80x25               ; if not, default to 80x25 color
E0BE  B5 07              C       mov     ch,07h                          ; if there, we believe switches,
E0C0  EB 31              C       jmp     short i_d_ok                    ; initialize display to mode7.
                         C
E0C2                     C i_d_ck_EGA:
                         C               assume  cs:code, ds:nothing, es:nothing, ss:stack_ram
                         C
E0C2  56                 C       push    si                              ;maybe an error msg pointer
E0C3  1E                 C       push    ds                              ;save -> data segment
E0C4  BB C000            C       mov     bx,0C000h               ; load starting segment
                         C
E0C7                     C i_d_EGA_rom_loop:
E0C7  8E DB              C       mov     ds,bx                           ; bx has pending segment
E0C9  33 F6              C       xor     si,si                           ; offset 0000h
E0CB  81 3C AA55         C       cmp     word ptr ds:[si],0AA55h         ;if no optional video controller
E0CF  75 05              C       jne     i_d_no_EGA                      ;ROM, default to 80 X 25 color
E0D1  E8 E4CC R          C       call    i_rom_check                     ;checksum & initialize EGA ROM
E0D4  EB 04              C       jmp short i_d_nxt_ROM                   ;and go check next ROM space
E0D6                     C i_d_no_EGA:
E0D6  81 C3 0080         C       add     bx,(800h/10h)                   ;add 2K to pending segment
E0DA                     C i_d_nxt_ROM:
E0DA  81 FB C800         C       cmp     bx,0C800h               ;end of video ROM space ?
E0DE  72 E7              C       jb      i_d_EGA_rom_loop        ;if not, continue checking
E0E0  1F                 C       pop     ds                              ;restore data segment pointer
E0E1  5E                 C       pop     si                              ;restore err msg (?) pointer
E0E2  A1 0010 R          C       mov     ax,word ptr ds:[switch_bits]    ; retrieve data from switches
E0E5  8A C8              C       mov     cl,al                           ; ch contains video mode
E0E7  EB 0A              C       jmp short i_d_ok
                         C
                         C               assume  cs:code, ds:data, es:nothing, ss:stack_ram
                         C
E0E9                     C i_d_80x25:
E0E9  80 E1 EF           C       and     cl,0EFh                         ; reset bit4 for 80x25 color.
E0EC  80 C9 20           C       or      cl,020h                         ;   set bit5 for 80x25 color.
E0EF  88 0E 0010 R       C       mov     byte ptr ds:[switch_bits],cl    ; Mono switch setting would
                         C                                               ; override value in al on
                         C                                               ; int 10, function 0.
                         C                                               ; change only the low byte
                         C                                               ; of the switch settings.
                         C
E0F3                     C i_d_ok:
                         C
E0F3  32 E4              C       xor     ah,ah                           ; ah = 0 (Used at i_d_mode)
```

```
                        C
                        C  ; Determine mode to initialize display monitor (from switches).
                        C
                        C               assume   cs:code, ds:data, es:abs0, ss:stack_ram
                        C
EOF5  80 E1 F0          C       and     cl,0F0h                         ; extract video switch values
EOF8  80 F9 30          C       cmp     cl,030h                         ; test for bits 4 & 5 = 11
EOFB  74 0C             C       je      i_d_mode                ; if so, Mono, ch is already set
EOFD  F6 C1 30          C       test    cl,030h                         ; EGA or no controller ?
E100  74 0B             C       jz      i_d_ret                         ; yes: GOTO exit
                        C
                        C  ; Fall-through means not EGA
                        C
E102                    C  i_d_color:
E102  F6 C1 20          C       test    cl,020h                         ; does user want 80x25 color?
E105  75 02             C       jnz     i_d_mode                ; if so, CH is set for Mode 3
E107  B5 01             C       mov     ch,01h                          ; else, initialize Mode 1.
E109                    C  i_d_mode:
                        C
                        C  ; Initialize desired board (from switches).
                        C
E109  8A C5             C       mov     al,ch                           ; transfer display mode to al.
E10B  CD 10             C       INT     10h                             ; ah = 0 = v_set_mode.
                        C
E10D                    C  i_d_ret:
E10D  1F                C       pop     ds                              ; restore registers
E10E  C3                C       ret
                        C
                        C  i_d_init     endp
                        C
                        C  ;--------------------------------------------------------------------------
                        C  ;    Install Vector Table
                        C  ;
                        C  ;    Input:  None.
                        C  ;    Output: None.
                        C  ;
                        C  ;    Trash:  ax = cx = 0 destroyed.
                        C  ;--------------------------------------------------------------------------
                        C
E10F                    C  i_vector     proc    near
                        C               assume  cs:code, ds:nothing, es:nothing, ss:stack_ram
                        C
E10F  1E                C       push    ds                              ; save registers
E110  06                C       push    es
E111  57                C       push    di
E112  56                C       push    si
                        C
                        C  ; Initialize Interrupt Vectors 00h through 07h to known routines.
                        C
                        C               assume  cs:code, ds:abs0, es:abs0, ss:stack_ram
                        C
E113  33 FF             C       xor     di,di                           ; satisfy assumptions
E115  8E DF             C       mov     ds,di                           ; ds = es = ax = abs0_seg = 0
E117  8E C7             C       mov     es,di                           ; es:di = abs0_seg:int00locn
E119  B8 FF23 R         C       mov     ax,cs:(offset ill_int)          ; ax = offset ill_int
```

```
E11C  B9 0008              C       mov     cx,(07h-00h)+1                  ; load INT's 00h through 07h.
                           C
E11F  AB                   C  i_vec0:      stosw                                   ; es:di++ gets offset ill_int
E120  8C 0D                C       mov     word ptr ds:[di],cs             ; es:di   gets cs
E122  47                   C       inc     di                              ; di++
E123  47                   C       inc     di
E124  E2 F9                C       loop    i_vec0                          ; until cx = 0.
                           C
                           C                                               ; load INT's 02h and 05h
E126  C7 06 0014 R FF54 R  C       mov     word ptr ds:[int05locn].cs:(offset s_int)
E12C  C7 06 0008 R F85F R  C       mov     word ptr ds:[int02locn].cs:(offset n_int)
                           C
                           C  ; Initialize Interrupt Vectors 08h through 1Eh to known routines.
                           C
                           C               assume  cs:code, ds:code, es:abs0, ss:stack_ram
                           C
E132  8C C8                C       mov     ax,cs                           ; satisfy assumptions
E134  8E D8                C       mov     ds,ax                           ; ds = ax = cs
E136  BE FEF3 R            C       mov     si,cs:(offset i_vec_tbl)    ; ds:si = code_seg:i_vec_tbl
                           C                                               ; es:di = abs0_seg:int08locn
E139  B1 18                C       mov     cl,(1Fh-08h)+1                  ; load INT's 08h through 1Fh.
                           C
E13B  A5                   C  i_vec8:      movsw                                   ; es:di++ gets ds:si   (offset)
E13C  AB                   C       stosw                                   ; es:di++ gets ax = cs (segment)
E13D  E2 FC                C       loop    i_vec8                          ; until cx = 0.
                           C
                           C  ; Initialize Interrupt Vectors 20h and above to zero.
                           C
E13F  33 C0                C       xor     ax,ax                           ; ax = 0
                           C                                               ; es:di = abs0_seg:int20locn
E141  B9 01B8              C       mov     cx,((03F0h-0080h)/2)            ; clear 0:0080h to 0:03F0h
                           C                                               ; don't blow away stack!
E144  F3/ AB               C       rep     stosw                           ; es:di++ gets 0
                           C
E146  5E                   C       pop     si                              ; restore registers
E147  5F                   C       pop     di
E148  07                   C       pop     es
E149  1F                   C       pop     ds
E14A  C3                   C       ret
                           C
                           C  i_vector     endp
                           C
                           C  ;---------------------------------------------------------------------------
                           C  ;    Initialize & Disable 8259A Programmable Interrupt Controller.
                           C  ;
                           C  ;    Input:  None.
                           C  ;    Output: None.
                           C  ;
                           C  ;    Trash:  al & dx destroyed.
                           C  ;---------------------------------------------------------------------------
                           C
E14B                       C  i_pic_init   proc    near
                           C               assume  cs:code, ds:nothing, es:nothing, ss:stack_ram
                           C
E14B  BA 0020              C       mov     dx,pic_0        ; dx = pic_0 (8259A 'control' port)
```

```
E14E  B0 13           C       mov     al,pic_icw1             ; edge triggered, single, icw4 to follow
E150  EE              C       out     dx,al
                      C
E151  42              C       inc     dx                      ; dx = pic_1 (8259A 'data' port)
E152  B0 08           C       mov     al,pic_icw2             ; interrupt vector base address
E154  EE              C       out     dx,al
                      C                                   ; since we are single mode (no slave), skip icw3
                      C
                      C                                       ; dx = pic_1 (8259A 'data' port)
E155  B0 0D           C       mov     al,pic_icw4             ; not special fully nested, buffered,
E157  EE              C       out     dx,al                   ; master, normal end_of_int, 8086 mode
                      C
E158  B0 FF           C       mov     al,pic_off_msk          ; mask all interrupts off for now
E15A  EE              C       out     dx,al                   ; dx = pic_1 (8259A 'data' port)
E15B  C3              C       ret
                      C
                      C  i_pic_init   endp
                      C
                      C  ;--------------------------------------------------------------------------
                      C  ;    Output Mask to 8259A Programmable Interrupt Controller.
                      C  ;
                      C  ;    Input:  AL = mask pattern
                      C  ;
                      C  ;    Output: Flags
                      C  ;
                      C  ;    Trash:  ah destroyed.
                      C  ;--------------------------------------------------------------------------
                      C
E15C                  C  i_out_mask   proc    near
                      C               assume  cs:code, ds:nothing, es:nothing, ss:stack_ram
                      C
E15C  E6 21           C       out     pic_1,al        ;output interrupt mask pattern
E15E  8A E0           C       mov     ah,al                   ;save pattern for compar
E160  E4 21           C       in      al,pic_1        ;get mask from 8259
E162  3A E0           C       cmp     ah,al                   ;the same ?
E164  C3              C       ret                             ;return flags = result of compare
                      C
                      C  i_out_mask   endp
                      C
                      C
                      C  ;--------------------------------------------------------------------
                      C  ;    8253 p_timer test for one p_timer counter channel
                      C  ;
                      C  ;    Input:  al      = 8253 p_timer control byte
                      C  ;            dx      = port address of 8253 p_timer data (counter)
                      C  ;
                      C  ;    Output: zf      = set (z status) if no error; reset (nz status) if error
                      C  ;            ah      = Error codes:  0 -> No Error!
                      C  ;                                    1 -> Low below time interval window.
                      C  ;                                    2 -> High above time interval window.
                      C  ;                                    3 -> No Response.
                      C  ;
                      C  ;    Trash:  al, bx & cx destroyed.
                      C  ;--------------------------------------------------------------------
                      C
```

```
E165                    C  rtc_chk      proc    near
                        C      assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E165  8A E0             C      mov      ah,al                   ; save control byte for later.
E167  B9 FFFF           C      mov      cx,0FFFFh       ; time out for both Register Bit Tests.
                        C
                        C  ; Register Bit Test (All Reset):  Count down from 100h until all bits reset.
                        C
E16A  8B D9             C      mov      bx,cx                   ; bx gets all its bits set.
                        C
E16C  E6 43             C      out      p_8253_ctrl,al          ; send i8253 p_timer control byte.
                        C
E16E  32 C0             C      xor      al,al                   ; al = 00h
E170  EE                C      out      dx,al                   ; load low byte of p_timer count.
E171  FE C0             C      inc      al                      ; al = 01h
E173  EE                C      out      dx,al                   ; load high byte of p_timer count.
                        C
E174                    C  rtc_chk_reset_lp:
E174  8A C4             C      mov      al,ah                   ; get control byte for read.
E176  24 C0             C      and      al,0C0h                 ; mask off all but top 2 bits.
E178  E6 43             C      out      p_8253_ctrl,al          ; send latching control byte for read.
                        C
E17A  EC                C      in       al,dx                   ; get low byte of p_timer count.
E17B  22 D8             C      and      bl,al                   ; 'and' low byte.
E17D  EC                C      in       al,dx                   ; get high byte of p_timer count.
E17E  22 F8             C      and      bh,al                   ; 'and' high byte.
                        C
E180  0B DB             C      or       bx,bx                   ; is bx = 0?
E182  74 05             C      jz       rtc_chk_reset_ok  ; if so, we're done.
E184  E2 EE             C      loop     rtc_chk_reset_lp  ; if not, continue reading.
                        C                                       ; (Note: loops less than 16 times.)
                        C
E186                    C  rtc_chk_reset_err:                   ; time out.
E186  B4 03             C      mov      ah,3                    ; Error3. (No Response.)
E188  C3                C      ret                              ; return nz status (loop leaves zf ok).
                        C
E189                    C  rtc_chk_reset_ok:
                        C
                        C  ; Register Bit Test (All Set):  Count down from 0h (FFFFh+1) until all bits set.
                        C
E189  33 DB             C      xor      bx,bx                   ; bx gets all its bits reset.
                        C
E18B  8A C4             C      mov      al,ah                   ; get control byte for load.
E18D  E6 43             C      out      p_8253_ctrl,al          ; send i8253 p_timer control byte.
                        C
E18F  32 C0             C      xor      al,al                   ; al = 00h
E191  EE                C      out      dx,al                   ; load low byte of p_timer count.
E192  EE                C      out      dx,al                   ; load high byte of p_timer count.
                        C
E193                    C  rtc_chk_set_lp:
E193  8A C4             C      mov      al,ah                   ; get control byte for read.
E195  24 C0             C      and      al,0C0h                 ; mask off all but top 2 bits.
E197  E6 43             C      out      p_8253_ctrl,al          ; send latching control byte for read.
                        C
E199  EC                C      in       al,dx                   ; get low byte of p_timer count.
```

```
E165                    C rtc_chk       proc    near
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E165  8A E0             C       mov     ah,al                   ; save control byte for later.
E167  B9 FFFF           C       mov     cx,0FFFFh       ; time out for both Register Bit Tests.
                        C
                        C ; Register Bit Test (All Reset):  Count down from 100h until all bits reset.
                        C
E16A  8B D9             C       mov     bx,cx                   ; bx gets all its bits set.
                        C
E16C  E6 43             C       out     p_8253_ctrl,al          ; send i8253 p_timer control byte.
                        C
E16E  32 C0             C       xor     al,al                   ; al = 00h
E170  EE                C       out     dx,al                   ; load low byte of p_timer count.
E171  FE C0             C       inc     al                      ; al = 01h
E173  EE                C       out     dx,al                   ; load high byte of p_timer count.
                        C
E174                    C rtc_chk_reset_lp:
E174  8A C4             C       mov     al,ah                   ; get control byte for read.
E176  24 C0             C       and     al,0C0h                 ; mask off all but top 2 bits.
E178  E6 43             C       out     p_8253_ctrl,al          ; send latching control byte for read.
                        C
E17A  EC                C       in      al,dx                   ; get low byte of p_timer count.
E17B  22 D8             C       and     bl,al                   ; 'and' low byte.
E17D  EC                C       in      al,dx                   ; get high byte of p_timer count.
E17E  22 F8             C       and     bh,al                   ; 'and' high byte.
                        C
E180  0B DB             C       or      bx,bx                   ; is bx = 0?
E182  74 05             C       jz      rtc_chk_reset_ok  ; if so, we're done.
E184  E2 EE             C       loop    rtc_chk_reset_lp  ; if not, continue reading.
                        C                                       ; (Note: loops less than 16 times.)
                        C
E186                    C rtc_chk_reset_err:                    ; time out.
E186  B4 03             C       mov     ah,3                    ; Error3. (No Response.)
E188  C3                C       ret                             ; return nz status (loop leaves zf ok).
                        C
E189                    C rtc_chk_reset_ok:
                        C
                        C ; Register Bit Test (All Set):  Count down from 0h (FFFFh+1) until all bits set.
                        C
E189  33 DB             C       xor     bx,bx                   ; bx gets all its bits reset.
                        C
E18B  8A C4             C       mov     al,ah                   ; get control byte for load.
E18D  E6 43             C       out     p_8253_ctrl,al          ; send i8253 p_timer control byte.
                        C
E18F  32 C0             C       xor     al,al                   ; al = 00h
E191  EE                C       out     dx,al                   ; load low byte of p_timer count.
E192  EE                C       out     dx,al                   ; load high byte of p_timer count.
                        C
E193                    C rtc_chk_set_lp:
E193  8A C4             C       mov     al,ah                   ; get control byte for read.
E195  24 C0             C       and     al,0C0h                 ; mask off all but top 2 bits.
E197  E6 43             C       out     p_8253_ctrl,al          ; send latching control byte for read.
                        C
E199  EC                C       in      al,dx                   ; get low byte of p_timer count.
```

```
E19A  0A D8             C       or      bl,al                   ; 'or' low byte.
E19C  EC                C       in      al,dx                   ; get high byte of p_timer count.
E19D  0A F8             C       or      bh,al                   ; 'or' high byte.
                        C
E19F  83 FB FF          C       cmp     bx,0FFFFh       ; is bx = 0FFFFh?
E1A2  74 05             C       jz      rtc_chk_set_ok          ; if so, we're done.
E1A4  E2 ED             C       loop    rtc_chk_set_lp          ; if not, continue reading.
                        C                                       ; (Note: loops less than 16 times.)
                        C
E1A6                    C rtc_chk_set_err:                      ; time out.
E1A6  B4 03             C       mov     ah,3                    ; Error3. (No Response.)
E1A8  C3                C       ret                             ; return nz status (loop leaves zf ok).
                        C
E1A9                    C rtc_chk_set_ok:
                        C
                        C ; p_timer Time Window Test:  Test p_timer versus CPU & see if it falls within spec.
                        C
E1A9  8A C4             C       mov     al,ah                   ; get control byte for read.
E1AB  24 C0             C       and     al,0C0h                 ; mask off all but top 2 bits.
E1AD  E6 43             C       out     p_8253_ctrl,al          ; send latching control byte for read.
                        C
E1AF  EC                C       in      al,dx                   ; get low byte of p_timer count.
E1B0  8A D8             C       mov     bl,al                   ; save low byte.
E1B2  EC                C       in      al,dx                   ; get high byte of p_timer count.
E1B3  8A F8             C       mov     bh,al                   ; save high byte.
                        C
E1B5  8A C4             C       mov     al,ah                   ; get control byte for read.
E1B7  24 C0             C       and     al,0C0h                 ; mask off all but top 2 bits.
E1B9  E6 43             C       out     p_8253_ctrl,al          ; send latching control byte for read.
                        C
E1BB  EC                C       in      al,dx                   ; get low byte of p_timer count.
E1BC  8A C8             C       mov     cl,al                   ; save low byte.
E1BE  EC                C       in      al,dx                   ; get high byte of p_timer count.
E1BF  8A E8             C       mov     ch,al                   ; save high byte.
                        C
E1C1  2B D9             C       sub     bx,cx                   ; calculate time difference.
                        C
                        C ; Do Time Range Checking (4 <= bx <= 28).
                        C
E1C3  B4 02             C       mov     ah,2                    ; Error2. (High above time window.)
E1C5  83 FB 1C          C       cmp     bx,28
E1C8  77 09             C       ja      rtc_chk_high            ; return nz status. (ja has zf reset.)
                        C
E1CA  FE CC             C       dec     ah                      ; Error1. (Low below time window.)
E1CC  83 FB 04          C       cmp     bx,4
E1CF  72 02             C       jb      rtc_chk_low             ; return nz status. (jb has zf reset.)
                        C
E1D1  FE CC             C       dec     ah                      ; Error0. (No Error!) return z status.
                        C
E1D3                    C rtc_chk_high:
E1D3                    C rtc_chk_low:
E1D3  C3                C       ret
                        C
                        C rtc_chk       endp
                        C
```

```
                        C ;--------------------------------------------------------------------
                        C ;    RAM (64k) Storage Test.
                        C ;
                        C ;    Input:   dx       = segment of RAM to be tested
                        C ;
                        C ;    Output:  zf       = set (z status) if no error; reset (nz status) if error
                        C ;             es:di    = dx:di = failing RAM location if error; else di = 0.
                        C ;             ax       = test pattern (what was written).
                        C ;             bx       = if error, what was read.
                        C ;             cx       = number left to test if error; else cx = 0.
                        C ;    NOTE:             If CX is zero and ZF is nz then parity error occured.
                        C ;
                        C ;    Trash:   None.
                        C ;--------------------------------------------------------------------
                        C
E1D4                    C memtst                proc     near
                        C                       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E1D4  B9 8000           C       mov     cx,08000h       ; get word count
                        C                                       ;       (64k = 32k * 2 bytes/word)
E1D7  8E C2             C       mov     es,dx
E1D9  33 FF             C       xor     di,di                   ; es:di = address
E1DB                    C memtst_w1:
E1DB  8B C7             C       mov     ax,di                   ;data = offset
E1DD  AB                C       stosw
E1DE  E2 FB             C       loop    memtst_w1
                        C
                        C ENDIF
E1E0  8E DA             C       mov     ds,dx
E1E2  33 DB             C       xor     bx,bx                   ;ds:bx = address
E1E4  B9 8000           C       mov     cx,08000h       ; word count
E1E7                    C memtst_r1:
E1E7  8B 07             C       mov     ax,[bx]                 ;read data
E1E9  3B C3             C       cmp     ax,bx                   ;verify data
E1EB  75 2C             C       jne     memtst_err
E1ED  43                C       inc     bx
E1EE  43                C       inc     bx                      ;next address
E1EF  E2 F6             C       loop    memtst_r1
                        C
E1F1  B9 8000           C       mov     cx,08000h       ; word count
E1F4                    C memtst_w2:                            ; address is already ok
E1F4  8B C7             C       mov     ax,di                   ; data = offset
E1F6  F7 D0             C       not     ax                      ; complement it
E1F8  AB                C       stosw                           ;fill memory
E1F9  E2 F9             C       loop    memtst_w2
                        C
E1FB  B9 8000           C       mov     cx,08000h       ; word count
                        C
E1FE                    C memtst_r2:
E1FE  8B 07             C       mov     ax,[bx]                 ; read data
E200  F7 D0             C       not     ax                      ; complement
E202  3B C3             C       cmp     ax,bx                   ; verify
E204  75 0F             C       jne     memtst_err_c
                        C
E206  43                C       inc     bx                      ; update address
```

```
E207  43                  C       inc     bx
E208  E2 F4               C       loop    memtst_r2
E20A  B8 0000             C       mov     ax,0                    ; to clear memory
E20D  B9 8000             C       mov     cx,08000h       ; word count
E210  F3/ AB              C       rep     stosw
E212  0B C0               C       or      ax,ax                   ;set ZF
                          C ENDIF
E214  C3                  C       ret
                          C
E215                      C memtst_err_c:
E215  F7 D0               C       not     ax                      ; error during complemented
E217  F7 D3               C       not     bx                      ;  address test
                          C
E219                      C memtst_err:
E219  93                  C       xchg    ax,bx                   ;return registers as specified
E21A  C3                  C       ret
                          C
                          C memtst                endp
                          C
E21B                      C code ends

                          C include pwrup0.asm
                          C
                          C ;=====================================================================
                          C ;    Filename:pwrup0.src
                          C ;
                          C ;    This module includes temporary hardware initialization.
                          C ;
                          C ;    includes Global constant definitions
                          C ;                    Master table definition
                          C ;                    Temporary initialization for floppy and
                          C ;                    hdu controllers, PIC, keyboard controller.
                          C ;                    Variable initialization for printer and
                          C ;                    RS232.
                          C ;                    Handshake w/APB.
                          C ;
                          C ;=====================================================================
                          C
E21B                      C code segment  public 'ROM'
                          C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                          C
                          C
E21B                      C p0_data1      proc    near
                          C
E21B  4F 70 74 69 6F 6E 61 C opt_ROM_m    db      'Optional ROM at '.NUL
      6C 20 52 4F 4D 20 61 C
      74 20 00            C
                          C
                          C
E22C  03BC                C p_tbldw       prt_data_a              ; printer in   port address space
E22E  0378                C       dw      prt_data_b              ; always on mother board.
E230  027A                C       dw      prt_data_c              ; printer in   port address space
E232  0000                C       dw      0                       ; no printer
                          C
E234  0052                C scc_tbl       dw      scc_ctl_b       ; rs232 SCC channel B
E236  0050                C       dw      scc_ctl_a       ; rs232 SCC channel A
```

```
                        C
E238  E477 R            C  alt_ret      dw      i_alt_restart           ; 00h Z8000 restart sequence offset
E23A  F000              C      dw       code_seg          ; 04h Z8000 restart sequence segment
                        C
E23C  0016              C  mastab       dw      ((mt_end)-(mastab))       ; 00h master table byte length
                        C
E23E  CC67 R            C      dw       kb_data_table             ; 02h kb xlation table offset
E240  F000              C      dw       code_seg          ; 04h kb xlation table segment
                        C
E242  FA6E R            C      dw       font_lo_8x8               ; 06h 1st 128 char.s 8x8 font offset
E244  F000              C      dw       code_seg          ; 08h 1st 128 char.s 8x8 font segment
                        C
E246  C060 R            C      dw       font_lo_8x16              ; 0Ah 1st 128 char.s 8x16 font offset
E248  F000              C      dw       code_seg          ; 0Ch 1st 128 char.s 8x16 font segment
                        C
E24A  0000              C      dw       0                         ; 0Eh 2nd 128 char.s 8x16 font offset
E24C  0000              C      dw       0                         ; 10h 2nd 128 char.s 8x16 font segment
                        C
E24E  0000              C      dw       0                         ; 12h soft font utility offset
E250  0000              C      dw       0                         ; 14h soft font utility segment
                        C
                        C                                        ; 16h etc...
E252                    C  mt_end       label   word
                        C
                        C  p0_data1     endp
                        C
                        C  ;---------------------------------------------------------------------------
                        C  ;    Initialize the basic hardware,
                        C  ;    Set up the interrupt pointers,
                        C  ;    Initialize all RAM variables,
                        C  ;    Clear the screen,
                        C  ;    Initialize the disk drivers,
                        C  ;    and perform the cold boot.
                        C  ;---------------------------------------------------------------------------
                        C
E252                    C  pcinit       proc    near
                        C
                        C      assume   cs:code, ds:data, es:data, ss:stack_ram
                        C
E252  B8 0040           C      mov      ax,data_seg                    ; satisfy assumptions
E255  8E D8             C      mov      ds,ax
E257  8E C0             C      mov      es,ax
                        C
                        C  ; Initialize Keyboard Controller.
                        C
E259  9C                C      pushf                                   ; save flags and
E25A  FA                C      cli                                     ; disable interrupts
                        C
E25B  BA 0061           C      mov      dx,p_kctrl                     ; dx = p_kctrl
E25E  B0 70             C      mov      al,70h                         ;; remove keyboard reset, but
E260  EE                C      out      dx,al                          ;; reset parity
                        C
E261  33 C9             C      xor      cx,cx                          ; delay
E263  E2 FE             C      loop     $
                        C
```

```
E265  B4 01              C       mov     ah,1                            ; enable self test
E267  E8 E502 R          C       call    kb_cmd_send
E26A  88 0E 008B R       C       mov     byte ptr ds:[kb_here],cl    ; initialize to zero
                         C
E26E  33 C9              C       xor     cx,cx                           ; delay
E270  E2 FE              C       loop    $
                         C  ; Flush any keyboard scan code and store AAh if we get it.
                         C
E272                     C  kb_flush:
E272  E4 64              C       in      al,kb_status                    ; get 8041 status
E274  A8 01              C       test    al,1                            ; test output buffer bit
E276  74 09              C       jz      kb_flush_back                   ; jump if no character pending
E278  E4 60              C       in      al,p_kscan                      ; get scan code from data port
E27A  3C AA              C       cmp     al,0AAh                         ; verify keyboard present
E27C  75 F4              C       jne     kb_flush
E27E  A2 008B R          C       mov     byte ptr ds:[kb_here],al    ; keyboard present
E281                     C  kb_flush_back:
E281  E2 EF              C       loop    kb_flush                    ;loop if zero
                         C
E283  33 C9              C       xor     cx,cx                           ; delay
E285  E2 FE              C       loop    $
                         C
                         C  ; Initialize System Variables.
                         C
                         C  ;mov word ptr ds:[master_tbl_ptr+0000h],cs:(offset mastab)
                         C  ;mov word ptr ds:[master_tbl_ptr+0002h],cs
                         C
                         C  ; Initialize Keyboard Driver Variables.
                         C
E287  B8 001E R          C       mov     ax,ds:(offset kb_buffer)    ; pointer to beginning of buffer
E28A  A3 001A R          C       mov     word ptr ds:[buffer_head],ax        ; keyboard output pointer offset
E28D  A3 001C R          C       mov     word ptr ds:[buffer_tail],ax        ; keyboard input pointer offset
E290  A3 0080 R          C       mov     word ptr ds:[buffer_start],ax       ; keeps beginning of buffer
                         C
E293  C7 06 0082 R 003E R C      mov     word ptr ds:[buffer_end],ds:(offset kb_buffer)+(size kb_buffer)
                         C
                         C       ; assume that memtest of first 64k has zeroed out kb_flag, kb_flag_1,
                         C       ; alt_input, etc.
                         C
                         C       ; Assume first not Deluxe Keyboard
                         C
E299  80 26 0017 R DF    C       and     byte ptr ds:[kb_flag],(not num_lock_mode)
E29E  80 26 0018 R FE    C       and     byte ptr ds:[kb_flag_1],(not dlx_kb)
                         C
                         C       ; Send command to request ID code from keyboard.
                         C
E2A3  B4 05              C       mov     ah,05H                          ; Read keyboard type
E2A5  E8 E502 R          C       call    kb_cmd_send                     ; -- send command.
                         C
E2A8  33 C9              C       xor     cx,cx                           ; set up timeout count
E2AA                     C  kb_type_wait:
E2AA  E4 64              C       in      al,kb_status                    ; get port status
E2AC  A8 01              C       test    al,1                            ; data byte available?
E2AE  75 05              C       jnz     kb_type_read                    ; if so, go read it ..
E2B0  E2 F8              C        loop   kb_type_wait                    ; else wait awhile longer.
```

```
E2B2  EB 11 90            C       jmp     kb_not_dlx                      ; timeout, default to non-dlx
                          C
E2B5                      C  kb_type_read:
E2B5  E4 60               C       in      al,p_kscan                      ; read ID byte..
E2B7  A8 01               C       test    al,01H                          ; deluxe kbd. bit set?
E2B9  74 0A               C       jz      kb_not_dlx
                          C
                          C       ; 01H bit set, so initialize to Deluxe Keyboard.
                          C
E2BB  80 0E 0017 R 20     C       or      byte ptr ds:[kb_flag],num_lock_mode
E2C0  80 0E 0018 R 01     C       or      byte ptr ds:[kb_flag_1],dlx_kb
                          C
E2C5                      C  kb_not_dlx:
                          C
E2C5  9D                  C       popf                                    ; restore interrupt-
                          C                                               ; enable state.
                          C
                          C  ; Initialize Printer & Communication (RS-232) Driver Variables.
                          C
E2C6  B0 14               C       mov     al,14h                          ; printer default timeout = 20
E2C8  BF 0078 R           C       mov     di,ds:(offset printer_t_out)
E2CB  B9 0004             C       mov     cx,4
E2CE  F3/ AA              C       rep     stosb                           ; es:di gets al
                          C
E2D0  B0 01               C       mov     al,01h                          ; printer default timeout = 01
E2D2  BF 007C R           C       mov     di,ds:(offset serial_t_out)
E2D5  B9 0004             C       mov     cx,4
E2D8  F3/ AA              C       rep     stosb                           ; es:di gets al
                          C
                          C  ; Determine Parallel Port Configuration.
                          C
                          C       assume  cs:code, ds:code, es:data, ss:stack_ram
                          C
E2DA  8C C8               C       mov     ax,cs
E2DC  8E D8               C       mov     ds,ax                           ; satisfy assumptions
                          C
E2DE  32 DB               C       xor     bl,bl                           ; clear high byte of switch_bits
                          C
E2E0  BF 0008 R           C       mov     di,es:(offset printer_addr); es:di points at printer_base
E2E3  BE E22C R           C       mov     si,ds:(offset p_tbl)            ; addresses of printer ports
                          C
E2E6                      C  i_prt_loop:
E2E6  AD                  C       lodsw                                   ; ax gets ds:si port address.
E2E7  0B C0               C       or      ax,ax                           ; valid port address?
E2E9  74 14               C       jz      i_prt_exit                      ; exit if invalid port address
                          C
E2EB  8B D0               C       mov     dx,ax                           ; transfer to data register
E2ED  B0 A5               C       mov     al,0A5h                         ; load test pattern
E2EF  EE                  C       out     dx,al                           ; output test pattern.
E2F0  86 C4               C       xchg    al,ah                           ; mov ah,al; trash al; & delay.
E2F2  EC                  C       in      al,dx                           ; input test pattern back.
E2F3  3A C4               C       cmp     al,ah                           ; what we read = test pattern ?
E2F5  75 EF               C       jnz     i_prt_loop                      ; if not, loop as port is absent
                          C                                               ; else, printer port is present
E2F7  8B C2               C       mov     ax,dx                           ; retrieve port address
```

```
E2F9  AB                C       stosw                                   ; es:di gets ax.
                        C
E2FA  80 C3 40          C       add     bl,040h                         ;add to high byte of switch_bits
                        C
E2FD  EB E7             C       jmp     i_prt_loop                      ; will go around loop 3 times
                        C
E2FF                    C  i_prt_exit:
                        C
                        C  ; Determine Communication (RS-232) Configuration (SCC Z8530 & INS8250's).
                        C
E2FF  BF 0000 R         C       mov     di,es:(offset rs232_addr)  ; es:di points at rs232_base
                        C
E302  BA 03FA           C       mov     dx,com_id_a                     ; read interrupt I.D. register
E305  EC                C       in      al,dx                           ; for first 8250 port.
E305  EC                C       in      al,dx                           ; for first 8250 port.
E308  75 07             C       jnz     i_no_com_a                      ; installed.
                        C
E30A  B8 03F8           C       mov     ax,com_data_a                   ; if present, load address of
E30D  AB                C       stosw                                   ; first 8250 data port.
                        C                                               ; es:di gets ax.
E30E  80 C3 02          C       add     bl,002h                         ;add to high byte of switch_bits
                        C
E311                    C  i_no_com_a:                                  ; es:di points next empty word
                        C
E311  BA 02FA           C       mov     dx,com_id_b                     ; read interrupt I.D. register
E314  EC                C       in      al,dx                           ; for second 8250 port.
E315  A8 F8             C       test    al,0F8h                         ; bits3-7 are always low if
E317  75 07             C       jnz     i_no_com_b                      ; installed.
                        C
E319  B8 02F8           C       mov     ax,com_data_b                   ; if present, load address of
E31C  AB                C       stosw                                   ; second 8250 data port.
                        C                                               ; es:di gets ax.
E31D  80 C3 02          C       add     bl,002h                         ;add to high byte of switch_bits
                        C
E320                    C  i_no_com_b:                                  ; read switch settings to test
E320  E4 66             C       in      al,sys_conf_a                   ; for SCC Z8530 chip
E322  A8 20             C       test    al,020h                         ; bit5: SCC chip installed
E324  74 17             C       jz      i_no_sccs               ; if not, don't load SCC table
                        C
E326  B0 0F             C       mov     al,0Fh                          ;;
E328  E6 50             C       out     scc_ctl_a,al                    ;; read register 15.
E32A  B9 0032           C       mov     cx,50                           ;; delay count
E32D  E2 FE             C       loop    $                               ;; delay for 8530
E32F  E4 50             C       in      al,scc_ctl_a                    ;;
E331  A8 01             C       test    al,1                            ;;
E333  75 08             C       jnz     i_no_sccs               ;; LSB of rr15 is always 0.
                        C
E335  BE E234 R         C       mov     si,ds:(offset scc_tbl)          ; SCC Z8530 control ports
E338  A5                C       movsw                                   ; always 2 ports
E339  A5                C       movsw                                   ; es:di gets ds:si (twice)
                        C
E33A  80 C3 04          C       add     bl,2*(002h)                     ;add to high byte of switch_bits
E33D                    C  i_no_sccs:                                   ; es:di points next empty word
                        C  ; Determine Game Card Configuration.
                        C
```

```
E33D  BA 0201            C       mov     dx,game_card                    ; get game card address.
E340  EC                 C       in      al,dx                           ; bits0-3 are low if installed
E341  A8 0F              C       test    al,0Fh
E343  75 03              C       jnz     i_no_game_card                  ; skip, if not present
                         C
E345  80 C3 10           C       add     bl,010h                         ;add to high byte of switch_bits
                         C
E348                     C  i_no_game_card:
                         C
                         C  ; Initialize High Byte of switch_bits.
                         C
E348  26: 88 1E 0011 R   C       mov     byte ptr es:[switch_bits+1],bl  ; save high byte of switch_bits
                         C
                         C  ; Initialize i8259A PIC with appropriate interrupt mask and enable interrupts.
                         C
                         C  ;    mov     al,10111100b            ; p_timer & kb & dsk at this point
E34D  B0 FC              C       mov     al,11111100b            ; p_timer & kb only at this point.
E34F  BA 0021            C       mov     dx,pic_1        ; now set proper interrupt mask
E352  EE                 C       out     dx,al
                         C
                         C  ; Send specific end of interrupt (SEOI) to pic 'command' port for keyboard.
                         C
E353  B0 61              C       mov     al,pic_seoi_1                   ; specific end of interrupt
E355  BA 0020            C       mov     dx,pic_0                ; to pic 'command' port.
E358  EE                 C       out     dx,al
E359  FB                 C       sti                                     ; enable interrupts
                         C
                         C  ; Initialize Parallel Printer Interface.
                         C
E35A  B4 01              C       mov     ah,1                            ; initialize printer...
E35C  33 D2              C       xor     dx,dx                           ; ...port 0
E35E  CD 17              C       INT     17h
                         C
                         C  ; Initialize all 4 (2) Z8530 Serial Communication Controller.
                         C  ; NOTE: Special function code (FF) for power up ONLY initialization of 8530
E360  B9 0004            C       mov     cx,4
E363                     C  rs_init:
E363  B8 FFE3            C       mov     ax,1111111111100011b            ; initialize SCC RS-232 (FFE3h)
                         C                                               ; 9600 baud,none,1 stop & 8 data
E366  8B D1              C       mov     dx,cx                           ; port number = loop - 1
E368  4A                 C       dec     dx
E369  CD 14              C       INT     14h
E36B  E2 F6              C       loop    rs_init
                         C
                         C  ;---------------------------------------------------------------------------
                         C  ; Call internal HDU init code.
                         C  ;---------------------------------------------------------------------------
                         C
                         C
                         C       assume  cs:code, ds:abs0, es:nothing, ss:stack_ram
                         C
E36D  33 C0              C       xor     ax,ax                   ; satisfy assumptions
E36F  8E D8              C       mov     ds,ax
                         C
E371  E4 67              C       in      al,sys_conf_b                   ;; port 67h.
```

```
E373  A8 04              C       test    al,4                            ;; test switch bit 2
E375  75 1B              C       jnz     i_hdu_ok                ;; if set, skip init
                         C
E377  BE DB5D R          C       mov     si,cs:(offset i_hdu_m)
E37A  E8 DFFA R          C       call    DRomString                      ; print test message
                         C
E37D  E8 D09F R          C       call    h_init
                         C
                         C       assume  cs:code, ds:data, es:nothing, ss:stack_ram
                         C
E380  2E: 8E 1E E574 R   C       mov     ds,word ptr cs:[set_ds_word]    ; satisfy assumptions
E385  80 3E 0075 R 00    C       cmp     byte ptr ds:[hf_num],0          ; number of hard disks.
E38A  75 06              C       jnz     i_hdu_ok                ; if ok, leave everything alone.
                         C
E38C  BC 0100            C       mov     sp,100h                         ; re-initialize stack
                         C
                         C ; Reinitialize Interrupt Vectors 0Dh, 13h, 19h,  to previous values.
                         C
E38F  E8 E03C R          C       call    re_i_vector                     ; re-install old vectors
                         C
E392                     C i_hdu_ok:
                         C
                         C ;---------------------------------------------------------------------------
                         C ;Test for and Initialize optional ROMs
                         C ;---------------------------------------------------------------------------
                         C
                         C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
                         C
E392  BB C800            C       mov     bx,0C800h               ; load starting segment
                         C
E395                     C rom_scan_loop:
E395  8E DB              C       mov     ds,bx                           ; bx has pending segment
E397  33 F6              C       xor     si,si                           ; offset 0000h
                         C
E399  81 3C AA55         C       cmp     word ptr ds:[si],0AA55h
E39D  75 19              C       jne     rom_scan_next
                         C
E39F  BE E21B R          C       mov     si,cs:(offset opt_ROM_m)        ; indicate ROM detected
E3A2  E8 DFFA R          C       call    DRomString
                         C
E3A5  33 C0              C       xor     ax,ax
E3A7  E8 E595 R          C       call    DHexLong                ; ds:ax points at ROM
                         C
E3AA  B8 0E20            C       mov     ax,(0Eh*100h)+' '               ; put out SPACE
E3AD  CD 10              C       INT     10h
                         C
E3AF  E8 E4CC R          C       call    i_rom_check                     ;checksum & initialize opt. ROM
E3B2  53                 C       push    bx                              ; save the segment for next ROM
                         C
E3B3  E8 E57C R          C       call    DCrLf                           ; Carriage Return, Line Feed
E3B6  EB 05              C       jmp     short rom_scan_exit
                         C
E3B8                     C rom_scan_next:
E3B8  81 C3 0080         C       add     bx,(800h/10h)                   ; add 2k to the pending segment
E3BC  53                 C       push    bx                              ; save the segment for next ROM
```

```
                        C
E3BD                    C rom_scan_exit:
                        C
                        C ;callDCrLf                            ; Carriage Return, Line Feed
E3BD  5B                C       pop     bx                              ; restore segment for next ROM
E3BE  81 FB F600        C       cmp     bx,0F600h               ; are we done?
E3C2  7C D1             C       jnge    rom_scan_loop                   ; if not, continue
                        C
                        C ; Clean Up after Option ROM's
                        C
E3C4  FA                C       cli                             ; disable interrupts
E3C5  BA 0021           C       mov     dx,pic_1        ; get current interrupt mask
E3C8  EC                C       in      al,dx
E3C9  24 FC             C       and     al,11111100b            ; p_timer & kb must be on at this point.
E3CB  EE                C       out     dx,al
E3CC  FB                C       sti                             ; enable interrupts
                        C
                        C ;--------------------------------------------------------------------------
                        C ;     FDU Test
                        C ;--------------------------------------------------------------------------
                        C
                        C ; Initialize Floppy Disk Controller and related Driver Variables
                        C
                        C       assume  cs:code, ds:abs0, es:nothing, ss:stack_ram
                        C
E3CD  33 C0             C       xor     ax,ax                   ; initialize the disk routines
E3CF  33 DB             C       xor     bx,bx
E3D1  33 C9             C       xor     cx,cx
E3D3  33 D2             C       xor     dx,dx
E3D5  CD 13             C       INT     13h
                        C
                        C ; Dummy Disk Attachment Test to Spin Up Drive for INT 19h (boot-strap).
                        C
E3D7  BE DB36 R         C       mov     si,cs:(offset i_fduA_m)
E3DA  E8 DFFA R         C       call    DRomString                      ; print test message
                        C
E3DD  BD 0003           C       mov     bp,3                            ; loop counter
E3E0                    C i_fdu_lp:
E3E0  B8 0201           C       mov     ax,0201h                ; read one sector
                        C
E3E3  33 DB             C       xor     bx,bx
E3E5  8E DB             C       mov     ds,bx
E3E7  8E C3             C       mov     es,bx                           ; xfer_segment
E3E9  BB 7C00           C       mov     bx,7C00h                ; xfer_offset
                        C
E3EC  B9 0001           C       mov     cx,0001h                ; track 0; sector 1
E3EF  33 D2             C       xor     dx,dx                           ; head  0; drive  0
                        C
E3F1  55                C       push    bp                              ; save retry count
E3F2  50                C       push    ax                              ; save return registers
E3F3  06                C       push    es
E3F4  CD 13             C       INT     13h                             ; bx, cx, dx, & ds preserved
E3F6  07                C       pop     es                              ; restore return registers
E3F7  58                C       pop     ax
E3F8  5D                C       pop     bp                              ; restore retry count
```

```
                        C
E3F9  73 08             C       jnc     i_fdu_ok                ; error during read?
E3FB  4D                C       dec     bp                              ; if so, decrement retry count
E3FC  75 E2             C       jnz     i_fdu_lp                ; and try again
                        C
E3FE  BE DB50 R         C       mov     si,cs:(offset i_fdu_not_m) ; drive not ready message.
E401  EB 03             C       jmp     short i_fdu_end
                        C
E403                    C  i_fdu_ok:
E403  BE DB54 R         C       mov     si,cs:(offset i_fdu_rdy_m) ; drive ready message.
                        C
E406                    C  i_fdu_end:
E406  E8 DFFA R         C       call    DRomString
                        C
                        C  ; If alternate processor is available, let user select which one to use.
                        C
E409                    C  i_alt_cpu:
E409  9C                C       pushf                                   ; preserve state for int 19h
E40A  50                C       push    ax
E40B  53                C       push    bx
E40C  51                C       push    cx
E40D  52                C       push    dx
E40E  55                C       push    bp
E40F  57                C       push    di
E410  56                C       push    si
E411  1E                C       push    ds
E412  06                C       push    es
                        C
                        C       assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
E413  2E: 8E 1E E574 R  C       mov     ds,word ptr cs:[set_ds_word]    ; satisfy assumptions
E418  A0 008B R         C       mov     al,byte ptr ds:[kb_here]  ; get result of keyboard test
E41B  3C AA             C       cmp     al,0AAh                         ; is keyboard attached?
E41D  74 03             C       je      i_alt_cont                      ; jump if yes (continue)
E41F  EB 7D 90          C       jmp     i_init_end                      ; jump if no (in mfg)
                        C
E422                    C  i_alt_cont:
                        C
E422  FA                C       cli                                     ; disable interupts
                        C
                        C       assume  cs:code, ds:abs0, es:nothing, ss:nothing
                        C
E423  33 C0             C       xor     ax,ax                           ; set up absolute zero address
E425  8E D8             C       mov     ds,ax                           ;    for alternate cpu semaphore
E427  8B 16 0000        C       mov     dx,word ptr ds:0          ; save word at 0:0
E42B  52                C       push    dx
E42C  A2 0000           C       mov     byte ptr ds:0,al          ; request alt cpu id (0:0 = 0)
                        C
E42F  BA 80C1           C       mov     dx,80C1h                  ; Z8000 liaison port
E432  B0 01             C       mov     al,1                            ; AL=1 starts Z8000
E434  EE                C       out     dx,al                     ; try to boot Z8000
E435  B9 00FF           C       mov     cx,0FFh                         ; loop counter
                        C
E438                    C  i_alt_test:
E438  80 3E 0000 01     C       cmp     byte ptr ds:0,01h         ; has the semaphore changed?
```

```
E43D  74 04              C       je      i_alt_found                     ; jump if yes (Z8000 id = 1)
E43F  E2 F7              C       loop    i_alt_test                  ; wait for Z8000 to gain control
E441  EB 56              C       jmp short i_alt_end                     ; jump: no alternate cpu
                         C
E443                     C  i_alt_found:
E443  58                 C       pop     ax                              ; restore word at 0:0
E444  A3 0000            C       mov     word ptr ds:0,ax
E447  50                 C       push    ax                              ; resave for future pop
E448  FB                 C       sti                                     ; enable interupts
E449  E8 E57C R          C       call    DCrLf
E44C  BE DB6A R          C       mov     si,cs:(offset i_alt_select_m)   ; ask user if alt. cpu used
E44F  E8 DFFA R          C       call    DRomString
E452  33 D2              C       xor     dx,dx                           ; for int 14h
                         C
E454                     C  i_alt_inq:
E454  B4 01              C       mov     ah,1                            ; has a key been struck?
E456  CD 16              C       INT     16h
E458  74 FA              C       jz      i_alt_inq               ; if not, stay in loop
                         C
E45A  32 E4              C       xor     ah,ah                           ; if so, get the keystroke
E45C  CD 16              C       INT     16h
E45E  0C 20              C       or      al,00100000b                    ; force lower case
E460  3C 79              C       cmp     al,'y'                          ; did user mean " yes"
E462  74 02              C       je      i_alt_echo                      ; jump if yes
E464  B0 6E              C       mov     al,'n'                          ; force " no"
                         C
E466                     C  i_alt_echo:
E466  B4 0E              C       mov     ah,0Eh                          ; echo " y"  or " n"
E468  CD 10              C       INT     10h
E46A  3C 79              C       cmp     al,'y'                          ; was answer " yes"
E46C  75 2B              C       jne     i_alt_end               ; jump if no
E46E  E8 E57C R          C       call    DCrLf
                         C
E471  FA                 C       cli                                     ; disable interupts
E472  C6 06 0000 0F      C       mov     byte ptr ds:0,0Fh       ; tell Z8000 to take over
                         C
E477                     C  i_alt_restart:               ; entry for re-entering Z8000 (assume 0:0 setup)
E477  BA 80C1            C       mov     dx,80C1h                ; Z8000 liaison port
E47A  B0 01              C       mov     al,1                            ; AL=1 starts Z8000
E47C  EE                 C       out     dx,al                       ; pass control to Z8000
                         C
E47D  B9 00FF            C       mov     cx,0FFh                         ; loop counter
E480  E2 FE              C       loop    $                           ; wait for Z8000 to gain control
                         C
                         C                               ; we're here only if Z8000 released control
E482  33 C0              C       xor     ax,ax                           ; reset segment register
E484  8E D8              C       mov     ds,ax                           ;    to absolute zero
E486  80 3E 0000 10      C       cmp     byte ptr ds:0,10h       ; is a jump requested?
E48B  72 0C              C       jb      i_alt_end               ; no, try to boot (or crash)
E48D  8C C8              C       mov     ax,cs                           ; pass return registers so that:
E48F  8E C0              C       mov     es,ax                           ;         jmp es:[bp]
E491  BD E238 R          C       mov     bp,offset alt_ret       ;     restarts Z8000 as desired
                         C       jmpf    0,0                             ; intersegment direct jmp to 0:0
E494  EA                 C1      db      0EAh
E495  0000               C1      dw      0
```

```
E497  0000                C1      dw      0
                          C
                          C  ; End of Initialization.
                          C
E499                      C  i_alt_end:
E499  58                  C       pop     ax                              ; restore word at 0:0
E49A  A3 0000             C       mov     word ptr ds:0,ax
E49D  FB                  C       sti                                     ; enable interupts
                          C
E49E                      C  i_init_end:
E49E  BA 0378             C       mov     dx,0378h                ; printer port
E4A1  B0 3F               C       mov     al,3Fh                          ; OK status
E4A3  EE                  C       out     dx,al                           ; tell mfg tester
                          C
E4A4  07                  C       pop     es                              ; restore state for boot
E4A5  1F                  C       pop     ds
E4A6  5E                  C       pop     si
E4A7  5F                  C       pop     di
E4A8  5D                  C       pop     bp
E4A9  5A                  C       pop     dx
E4AA  59                  C       pop     cx
E4AB  5B                  C       pop     bx
E4AC  58                  C       pop     ax
E4AD  9D                  C       popf
                          C
E4AE  E8 E57C R           C       call    DCrLf
                          C
E4B1  1E                  C       push    ds
E4B2  2E: 8E 1E E574 R    C       mov     ds,word ptr cs:[set_ds_word]
                          C
                          C       assume  cs:code, ds:data, es:nothing, ss:stack_ram
                          C
E4B7  8A 26 0098 R        C       mov     ah,byte ptr ds:[disable_nmi_flag] ;input parameter
E4BB  E8 E66F R           C       call    enable_RAM_parity       ; disable I/O, enable RAM parity
                          C
                          C  ; Set the reset flag, so the hdu initialization code will recognize a
                          C  ; push-button reset (also modifies 1234h setting so it won't think it's an
                          C  ; ALT-CTRL-DEL reset)
                          C
                          C
E4BE  80 26 0098 R FE     C       and     byte ptr ds:[disable_nmi_flag],0FEh ;clear this variable.
E4C3  B8 4321             C       mov     ax,4321h                ; 4321 means pushbutton reset
E4C6  A3 0072 R           C       mov     reset_flag,ax                   ; set the reset flag
                          C
E4C9  1F                  C       pop     ds                              ; restore data segment
                          C
                          C
E4CA  CD 19               C       INT     19h                             ; go to boot-strap routine
                          C
                          C  pcinit       endp
                          C
                          C  ;=============================================================================
                          C  ;
                          C  ; Perform checksum on optional ROMs.  If checksum is OK, execute Optional
                          C  ; ROM's initialization code.
```

```
                        C  ;
                        C  ;    Input:
                        C  ;            DS = segment of optional ROM to be tested
                        C  ;    Output:
                        C  ;            BX = segment for next ROM to be tested
                        C  ;    Destroys:
                        C  ;            ES, SI
                        C  ;
                        C  ;--------------------------------------------------------------------------
                        C
E4CC                    C  i_rom_check  proc    near
                        C
E4CC  50                C      push     ax                              ; save to restore at exit
E4CD  51                C      push     cx                              ; save to restore at exit
E4CE  B8 0040           C      mov      ax,data_seg                     ; satisfy assumptions
E4D1  8E C0             C      mov      es,ax                           ; for es in rom_check
E4D3  33 F6             C      xor      si,si                           ; ds:si points to ROM to check
                        C
                        C  ;   Now:     ds:si    = pointer to ROM to be tested
                        C  ;            bx = ds  = pending segment of ROM under test
                        C  ;            es:      = data segment
                        C
                        C              assume   cs:code, ds:nothing, es:data, ss:nothing
                        C
E4D5  33 C0             C      xor      ax,ax                           ; clear al
E4D7  8A 64 02          C      mov      ah,byte ptr ds:[si+2]           ; ax = (ROM length/512) * 256
E4DA  D1 E0             C      shl      ax,1                            ; ax = (ROM length/512) * 512
                        C                                               ; ax = ROM length in bytes
E4DC  50                C      push     ax                              ; save ROM length
E4DD  B1 04             C      mov      cl,4
E4DF  D3 E8             C      shr      ax,cl                           ; advance segment for next ROM
E4E1  03 D8             C      add      bx,ax                           ; by the number of paragraphs
E4E3  59                C      pop      cx                              ; restore ROM length in cx
                        C
E4E4  E8 E569 R         C      call     rom_checksum_cnt          ; get the checksum of the cx-
                        C                                               ; byte ROM.
E4E7  74 03             C      jz       rom_chksum_ok                   ; OK if the checksum was zero
E4E9  EB 6B 90          C      jmp      rom_err                         ; error the checksum wasn't zero
                        C
E4EC                    C  rom_chksum_ok:
E4EC  53                C      push     bx                              ; save the segment for next ROM
                        C
E4ED  26: C7 06 0067 R 0003 C  mov      word ptr es:[io_rom_init],0003h
E4F4  26: 8C 1E 0069 R  C      mov      word ptr es:[io_rom_seg],ds
E4F9  26: FF 1E 0067 R  C      call     dword ptr es:[io_rom_init] ; initialize the ROM
                        C
E4FE  5B                C      pop      bx                              ; restore segment for next ROM
E4FF  59                C      pop      cx                              ; restore initial value
E500  58                C      pop      ax                              ; restore initial value
E501  C3                C      ret
                        C
                        C  i_rom_check  endp
                        C
                        C  ;--------------------------------------------------------------------------
                        C  ;
```

```
                        C  ; Send command in AH to keyboard interface processor.  AX is used.
                        C  ;
                        C  ;--------------------------------------------------------------------------
                        C
                        C
E502                    C  kb_cmd_send proc near
                        C
E502                    C  kb_cmd_wlup:
E502  E4 64             C       in      al,kb_status                    ; get 8041 port status
E504  A8 02             C       test    al,10b                          ; ready to receive?
E506  75 FA             C       jnz     kb_cmd_wlup
                        C
E508  8A C4             C       mov     al,ah                           ; ready, send command
E50A  E6 60             C       out     p_kscan,al
E50C  C3                C       ret
                        C
                        C  kb_cmd_send endp
                        C
E50D                    C  code ends

                        C  include pwrup2.asm
                        C
                        C  ;=========================================================================
                        C  ;    Filename:pwrup2.src
                        C  ;
                        C  ;    This module contains the illegal interrupt handling routine.
                        C  ;
                        C  ;=========================================================================
                        C
E50D                    C  code segment  public 'ROM'
                        C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C  ;=========================================================================
                        C  ;    Note:    We are called from ill_int ONLY (see vector.src), and
                        C  ;             stack looks like this:
                        C  ;                                                  High Address
                        C  ;                      |------------------| <-- sp before ill_int trap
                        C  ;             (10)     | return fsw flags |
                        C  ;                      |------------------|
                        C  ;             (0E)     | return cs segment|
                        C  ;                      |------------------|
                        C  ;             (0C)     | return ip offset |
                        C  ;                      |------------------| <-- sp after ill_int trap
                        C  ;             (0A)     |        ax        |
                        C  ;                      |------------------|
                        C  ;             (08)     |        ds        |
                        C  ;                      |------------------| <-- sp after ill_int pushes
                        C  ;             (06)     |  near call here  |
                        C  ;                      |------------------| <-- sp after ill_int calls ill_trap
                        C  ;             (04)     |        ax        |
                        C  ;                      |------------------|
                        C  ;             (02)     |        dx        |
                        C  ;                      |------------------|
                        C  ;             (00)     |        si        |
                        C  ;                      |------------------| <-- sp after ill_trap pushes
```

```
                        C  ;                                                  Low Address
                        C  ;=============================================================================
                        C
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E50D                    C  ill_trap     proc    near
                        C
                        C  ; Turn off floppy disk drives.
                        C
E50D  50                C       push    ax                              ; save registers
E50E  52                C       push    dx
E50F  56                C       push    si
                        C
E510  E8 ED1B R         C       call    stop_disk               ; destroys ax & dx
                        C
E513  E8 E545 R         C       call    ill_ln
E516  BE DA50 R         C       mov     si,cs:(offset ill_m1)           ; part 1 of message
E519  E8 DFFA R         C       call    DRomString
                        C
E51C  8B F4             C       mov     si,sp
E51E  36: 8E 5C 0E      C       mov     ds,word ptr ss:[si+0Eh]         ; cs past si,dx,ax,ret,ds,ax,ip
E522  36: 8B 74 0C      C       mov     si,word ptr ss:[si+0Ch]         ; ip past si,dx,ax,ret,ds,ax
                        C
E526  4E                C       dec     si                              ; si points to interrupt number
E527  8A 04             C       mov     al,byte ptr ds:[si]             ; print illegal interrupt number
E529  E8 E5AD R         C       call    DHexByte
E52C  4E                C       dec     si                              ; si points to interrupt instr.
E52D  8B C6             C       mov     ax,si                           ; save pointer
                        C
E52F  BE DA69 R         C       mov     si,cs:(offset ill_m2)           ; part 2 of message
E532  E8 DFFA R         C       call    DRomString
                        C
E535  E8 E595 R         C       call    DHexLong                ; print illegal cs:ip = ds:ax
                        C
E538  BE DA6F R         C       mov     si,cs:(offset ill_m3)           ; part 3 of message
E53B  E8 DFFA R         C       call    DRomString
E53E  E8 E545 R         C       call    ill_ln
                        C
E541  5E                C       pop     si                              ; restore registers
E542  5A                C       pop     dx
E543  58                C       pop     ax
E544  C3                C       ret
                        C
                        C
E545  E8 E57C R         C  ill_ln:      call    DCrLf                           ; prints a line of '*'s
E548  B2 2A             C       mov     dl,42
                        C
E54A  B8 0E2A           C  ill_lp: mov  ax,(0Eh*100h)+('*')
E54D  CD 10             C       INT     10h
E54F  FE CA             C       dec     dl
E551  75 F7             C       jnz     ill_lp
E553  EB 27 90          C       jmp     DCrLf
                        C
                        C  ill_trap     endp
                        C
```

```
E556                    C  code ends

                        C  include pwrup3.asm
                        C
                        C  ;===========================================================================
                        C  ;    Filename:pwrup3.src
                        C  ;
                        C  ;    This module contains part of the ROM checksum test.
                        C  ;
                        C  ;===========================================================================
                        C
E556                    C  code segment  public   'ROM'
                        C
                        C               assume   cs:code, ds:nothing, es:data, ss:nothing
                        C
                        C  ;---------------------------------------------------------------------------
                        C  ;    Input:   ds       = segment of ROM under test
                        C  ;             es       = firmware data segment
                        C  ;
                        C  ;    Trash:   All other registers except bx destroyed (in general).
                        C  ;---------------------------------------------------------------------------
                        C
E556                    C  rom_err               proc     near
                        C               assume   cs:code, ds:nothing, es:data, ss:nothing
                        C
E556  8C D8             C       mov      ax,ds
E558  26: 88 26 0015 R  C       mov      byte ptr es:[mfg_err_flag],ah       ; high byte of ROM address
                        C
E55D  BE DA7A R         C       mov      si,cs:(offset fail_m)              ; indicate ROM failed
E560  E8 DFFA R         C       call     DRomString
E563  EB 17 90          C       jmp      DCrLf
                        C
                        C  rom_err               endp
                        C
                        C
                        C
                        C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C  ;---------------------------------------------------------------------------
                        C  ;    Input:   ds:si    = pointer to ROM to be tested
                        C  ;
                        C  ;    Output:  ah       = checksum for the ROM
                        C  ;             cx       = 0
                        C  ;             si       = pointer to byte past ROM
                        C  ;             zf       = state of checksum for the ROM
                        C  ;
                        C  ;    Trash:   al destroyed.
                        C  ;---------------------------------------------------------------------------
                        C
E566                    C  rom_checksum  proc     near
                        C               assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E566  B9 4000           C       mov      cx,4000h
                        C
E569                    C  rom_checksum_cnt:
```

```
E569  33 C0               C       xor     ax,ax                           ; clear ah
                          C
E56B                      C  rom_checksum_loop:
E56B  AC                  C       lodsb                                   ; 12 al gets ds:si
E56C  02 E0               C       add     ah,al                           ;  3
E56E  E2 FB               C       loop    rom_checksum_loop          ; 17
                          C
E570  0A E4               C       or      ah,ah
E572  C3                  C       ret
                          C
                          C  rom_checksum  endp
                          C
E573                      C  code ends

                          C  include pwrup4.asm
                          C
                          C
                          C  ;===========================================================================
                          C  ;    Filename:pwrup4.src
                          C  ;
                          C  ;    This module contains utility routines which provide various
                          C  ;    ASCII display capabilities, particularly useable for test/
                          C  ;    error messages.
                          C  ;
                          C  ;===========================================================================
                          C
                          C
E573                      C  code segment  public 'ROM'
                          C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                          C
                          C  ;===========================================================================
                          C  ;    Utility Routines:
                          C  ;
                          C  ;    DCrLf             DColon
                          C  ;    DHexLong DHexWord DHexByte DHexNib
                          C  ;    DNum              DNumW
                          C  ;===========================================================================
                          C
E573                      C  p4_data1      proc     near
                          C
E573  90                  C  even
                          C
E574  0040                C  set_ds_word   dw       data_seg          ; 2 bytes=  0 clocks
                          C
                          C  p4_data1      endp
                          C
                          C  ;===========================================================================
                          C  ;
                          C  ; set_ds: procedure to laoad DS with data segment
                          C  ;
                          C  ;===========================================================================
                          C
E576                      C  set_ds                proc     near              ; set ds to firmware data segment
                          C                assume   cs:code, ds:nothing, es:nothing, ss:nothing
                          C
```

```
E576  2E: 8E 1E E574 R      C       mov     ds,word ptr cs:[set_ds_word]        ; 5 bytes 2+9+6 = 17 clocks
E57B  C3                    C       ret                                         ; 1 byte =  8 clocks
                            C                                                   ; -------  ---------
                            C  set_ds               endp
                            C
                            C  ;=============================================================================
                            C  ;    Display ASCII String Utilities
                            C  ;=============================================================================
                            C
                            C
E57C                        C  DCrLf        proc    near    ; Displays a CR & LF.
                            C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
E57C  50                    C       push    ax              ; all registers preserved
E57D  B8 0E0D               C       mov     ax,(0Eh*100h)+CR
E580  CD 10                 C       INT     10h             ; tty emulator
E582  B8 0E0A               C       mov     ax,(0Eh*100h)+LF
E585  CD 10                 C       INT     10h             ; tty emulator
E587  58                    C       pop     ax              ; restore ax
E588  C3                    C       ret
                            C  DCrLf        endp
                            C
E589                        C  DColon               proc    near    ; Displays a ':'.
                            C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
E589  50                    C       push    ax              ; all registers preserved
E58A  53                    C       push    bx
E58B  B3 01                 C       mov     bl,1            ; select foreground color for grafix modes
E58D  B8 0E3A               C       mov     ax,(0Eh*100h)+':'
E590  CD 10                 C       INT     10h             ; tty emulator
E592  5B                    C       pop     bx              ; restore registers
E593  58                    C       pop     ax
E594  C3                    C       ret
                            C  DColon               endp
                            C
                            C  ;=============================================================================
                            C  ;    Display Hexadecimal Number in ASCII Utilities
                            C  ;=============================================================================
                            C
E595                        C  DHexLong     proc    near    ; Displays ds:ax in ASCII
                            C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
E595  50                    C       push    ax              ; all registers preserved
E596  E8 E020 R             C       call    TstVid          ; is there a video controller present ?
E599  73 02                 C       jnc     DHexOK          ; OK if there is: go display Hex String
E59B  58                    C       pop     ax              ; if no video controller, restore Ax...
E59C  C3                    C       ret                     ; and just return.
                            C
E59D                        C  DHexOK:
E59D  8C D8                 C       mov     ax,ds           ; display segment first
E59F  E8 E5A6 R             C       call    DHexWord
                            C
E5A2  E8 E589 R             C       call    DColon          ; display a colon
                            C
E5A5  58                    C       pop     ax              ; restore ax
                            C  ;    jmp     short DHexWord  ; fall through:  display offset second
                            C
                            C  DHexLong     endp
```

```
                        C
E5A6                    C DHexWord      proc    near    ; Displays ax in ASCII
                        C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
E5A6  50                C       push    ax              ; all registers preserved
E5A7  8A C4             C       mov     al,ah
E5A9  E8 E5AD R         C       call    DHexByte ; display high byte first
E5AC  58                C       pop     ax              ; restore ax
                        C ;     jmp     short DHexByte  ; fall through:  display low byte second
                        C
                        C DHexWord      endp
                        C
E5AD                    C DHexByte      proc    near    ; Displays al in ASCII
                        C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
E5AD  50                C       push    ax              ; all registers preserved
E5AE  E8 E020 R         C       call    TstVid          ; is there a video controller present ?
E5B1  73 02             C       jnc     DHexBOK         ; OK if there is: go display Hex byte
E5B3  58                C       pop     ax              ; if no video controller, restore Ax...
E5B4  C3                C       ret                     ; and just return.
E5B5                    C DHexBOK:
E5B5  D0 C8             C       ror     al,1
E5B7  D0 C8             C       ror     al,1
E5B9  D0 C8             C       ror     al,1
E5BB  D0 C8             C       ror     al,1            ; move high nibble to low nibble
E5BD  E8 E5C1 R         C       call    DHexNib         ; display high nibble in ASCII
E5C0  58                C       pop     ax              ; restore ax
                        C ;     jmp     short DHexNib   ; fall through:  display low nibble in ASCII
                        C
                        C DHexByte      endp
                        C
E5C1                    C DHexNib               proc    near    ; Displays low nibble of al in ASCII
                        C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
E5C1  50                C       push    ax              ; all registers preserved
E5C2  53                C       push    bx
E5C3  B3 01             C       mov     bl,1            ; select foreground color for grafix modes
E5C5  24 0F             C       and     al,0fh          ; clear high nibble
E5C7  04 30             C       add     al,'0'
E5C9  3C 39             C       cmp     al,'9'
E5CB  76 02             C       jbe     NibOk           ; '0' <= al <= '9' ?
E5CD  04 07             C       add     al,'A'-'0'-10
E5CF  B4 0E             C NibOk:        mov     ah,0Eh          ; tty emulator
E5D1  CD 10             C       INT     10h
E5D3  5B                C       pop     bx              ; restore registers
E5D4  58                C       pop     ax
E5D5  C3                C       ret
                        C DHexNib               endp
                        C
                        C ;=============================================================================
                        C ;    Display Decimal Number in ASCII Utilities
                        C ;=============================================================================
                        C
E5D6                    C DNum          proc    near    ; Displays decimal of ax in ASCII in min width
                        C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
E5D6  53                C       push    bx              ; all registers preserved
E5D7  33 DB             C       xor     bx,bx           ; minimum width
E5D9  E8 E5DE R         C       call    DNumW           ; display ax
```

```
E5DC  5B                  C       pop     bx              ; restore bx
E5DD  C3                  C       ret
                          C  DNum         endp
                          C
E5DE                      C  DNumW        proc    near    ; Displays decimal of ax in ASCII of width bx
                          C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
                          C
E5DE  50                  C       push    ax              ; all registers preserved
E5DF  E8 E020 R           C       call    TstVid  ; is there a video controller present ?
E5E2  72 2E               C       jc      DNumWExit; if there isn't go exit
E5E4  53                  C       push    bx
E5E5  51                  C       push    cx
E5E6  52                  C       push    dx
E5E7  56                  C       push    si
                          C
E5E8  BE 000A             C       mov     si,10                   ; decimal modulus
E5EB  33 C9               C       xor     cx,cx                   ; clear digit counter
E5ED                      C  DNumW_loop:
E5ED  33 D2               C       xor     dx,dx                   ; dx:ax = decimal number
E5EF  F7 F6               C       div     si                      ; dh = 0
                          C                                       ; dl = remainder = (0-9)
                          C                                       ; ax = quotient  = higher order digits
E5F1  52                  C       push    dx                      ; save the digit on stack
E5F2  41                  C       inc     cx                      ; increment count of what's on stack
E5F3  0B C0               C       or      ax,ax                   ; are we done?
E5F5  75 F6               C       jnz     DNumW_loop
                          C
E5F7  2B D9               C       sub     bx,cx                   ; subtract digit count from width
E5F9  76 08               C       jbe     DNumW_skip              ; skip spaces if bx is not > cx
                          C
E5FB                      C  DNumW_spaces:
E5FB  B8 0E20             C       mov     ax,(0Eh*100h)+' ' ; display a space
E5FE  CD 10               C       INT     10h
E600  4B                  C       dec     bx                      ; decrement count of spaces
E601  75 F8               C       jnz     DNumW_spaces            ; keep going
                          C
E603                      C  DNumW_skip:
E603  B3 01               C       mov     bl,1                    ; foreground color for grafix modes
E605  58                  C       pop     ax                      ; remove digit from stack
E606  04 30               C       add     al,'0'                  ; convert to ASCII
E608  B4 0E               C       mov     ah,0Eh                  ; display the digit
E60A  CD 10               C       INT     10h
E60C  E2 F5               C       loop    DNumW_skip
                          C
E60E  5E                  C       pop     si                      ; restore registers
E60F  5A                  C       pop     dx
E610  59                  C       pop     cx
soft (R) Macro Assembler  Version 4.00              4/3/86 16:59:25

E612  58                  C       pop     ax
E613  C3                  C       ret
                          C  DNumWendp
                          C
E614                      C  enable_parity proc                   ::
                          C
```

```
E614  50                    C       push    ax
E615  E4 61                 C       in      al,p_kctrl                      ;; read B port. (61h)
E617  0C 30                 C       or      al,30h                          ;; enable bits 4 & 5.
E619  E6 61                 C       out     p_kctrl,al                      ;;
E61B  24 CF                 C       and     al,0CFh                         ;;
E61D  E6 61                 C       out     p_kctrl,al                      ;;
E61F  B0 80                 C       mov     al,nmi_enable                   ;; 80h.
E621  E6 A0                 C       out     nmi_enable_port,al      ;; defined in sysdata.src (A0h)
E623  58                    C       pop     ax                              ;;
E624  C3                    C       ret                                     ;;
                            C
E625  0D 0A 50 61 72 69 74  C parity1_m     db      0Dh,0Ah,'Parity error on system board',NUL   ;;
      79 20 65 72 72 6F 72  C
      20 6F 6E 20 73 79 73  C
      74 65 6D 20 62 6F 61  C
      72 64 00              C
E644  0D 0A 50 61 72 69 74  C parity2_m     db      0Dh,0Ah,'Parity error on expansion board',NUL;;
      79 20 65 72 72 6F 72  C
      20 6F 6E 20 65 78 70  C
      61 6E 73 69 6F 6E 20  C
      62 6F 61 72 64 00     C
                            C
                            C enable_parity endp                            ;;
                            C
                            C ;=============================================================================
                            C ;
                            C ; Disable the I/O and RAM Parity FlipFlops.
                            C ; This disallows a parity error to be generated from a memory add-on board
                            C ; on the bus converter board (I/O Parity), or from the motherboard (RAM
                            C ; Parity).
                            C ;-----------------------------------------------------------------------------
                            C
E666                        C disable_parityproc    near
                            C
E666  50                    C       push    ax
                            C
E667  E4 61                 C       in      al,p_kctrl              ;read parity error flipflop
E669  0C 30                 C       or      al,parity_disable ;set the Disable Parity bits
E66B  E6 61                 C       out     p_kctrl,al              ;disable parity
                            C
E66D  58                    C       pop     ax
E66E  C3                    C       ret
                            C
                            C disable_parityendp
                            C
                            C ;=============================================================================
                            C ;
                            C ; First clear any pending parity errors by disabling both Parity Bits in
                            C ; parity flip-flop.
                            C ; Then enable RAM parity, allowing a prity interrupt from the motherboard,
                            C ; while disabling I/O parity, disallowing a parity interrupt from an add-on
                            C ; memory board on the Bus Converter.
                            C ; Finally, enable NMI, which will allow the RAM Parity interrupt to occur, and
                            C ; also allow interrupts from the 8087 to occur.
                            C ;     input:  ah, bit 0 =1, don't enable NMI
```

```
                        C  ;                bit 0 =0, enable NMI
                        C  ;
                        C  ;--------------------------------------------------------------------------
                        C
E66F                    C  enable_RAM_parity      proc     near                    ;;
                        C
E66F  50                C       push     ax
                        C
E670  E4 61             C       in       al,p_kctrl                      ;; read B port. (61h)
E672  0C 30             C       or       al,parity_disable          ;; set the Disable Parity bits
E674  E6 61             C       out      p_kctrl,al                      ;; clear any pending parity
                        C
E676  24 CF             C       and      al,NOT parity_disable           ;; clear both parity bits
E678  0C 20             C       or       al,I_O_parity_disable           ;; enable RAM Parity, but
E67A  E6 61             C       out      p_kctrl,al                      ;; disable I/O Parity
                        C
E67C  F6 C4 01          C       test     ah,01h                          ;;enable NMI?
E67F  75 06             C       jnz      leave_nmi_disabled         ;;jump if not
E681  B0 80             C       mov      al,nmi_enable                   ;; 80h to enable NMI.
E683  E6 A0             C       out      nmi_enable_port,al         ;; defined in sysdata.src (A0h)
                        C
E685  EB 04             C       jmp      short enable_RAM_exit
E687                    C  leave_nmi_disabled:
E687  32 C0             C       xor      al,al
E689  E6 A0             C       out     nmi_enable_port,al
E68B                    C  enable_RAM_exit:
                        C
                        C
E68B  58                C       pop      ax                              ;;
E68C  C3                C       ret                                      ;;
                        C
                        C  enable_RAM_parity      endp
                        C
E68D                    C  code ends

                        C  include int15.asm
                        C
                        C
                        C
                        C  ;--------------------------------------------------------------------------
                        C  ;    INT 15h -- cassette I/O
                        C  ;--------------------------------------------------------------------------
                        C
E68D                    C  code segment public 'ROM'
                        C       assume   cs:code,ds:nothing,es:nothing,ss:nothing
                        C
                        C
E68D                    C  int15proc     far
E68D                    C  int15a        label    near
E68D  F9                C       stc
E68E  B4 86             C       mov      ah,86h
E690  CA 0002           C       ret      2
                        C  int15endp
                        C
E693                    C  code ends
```

```
                        C
                        C  include boot1.asm
                        C
                        C  ;=====================================================================
                        C  ;    Filename:boot1.src
                        C  ;
                        C  ;    This module includes the ORG'd jump to INT 19h (boot2.src)
                        C  ;
                        C  ;=====================================================================
                        C
E693                    C  code segment  public 'ROM'
                        C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C       XORG     0E6F2h
E6F2                    C1      org      0E6F2h
                        C
E6F2                    C  bt_jmp        proc     near
                        C
E6F2  E9 F87B R         C       jmp      bt_int                          ; necessary jump for ORG
                        C
                        C  bt_jmp        endp
                        C
E6F5                    C  code ends

                        C  include int18data.asm
                        C
                        C
E6F5                    C  code segment  public   'ROM'
                        C       assume   cs:code, ds:data
                        C
E6F5                    C  int18data     proc     near
                        C
E6F5  52 4F 4D 20 42 41 53  C  trap_mess     db       'ROM BASIC Not Available. ',CR,LF
      49 43 20 4E 6F 74 20  C
      41 76 61 69 3C 61 62  C
      6C 65 2C 20 0D 0A     C
E710  50 72 65 73 73 20 52  C              db       'Press Reset to re-boot',CR,LF,NUL
      65 73 65 74 20 74 6F  C
      20 72 65 2D 62 6F 6F  C
      74 0D 0A 00           C
                        C  int18data     endp
                        C
E729                    C  code ends

                        C  include comm1.asm
                        C
                        C  ;=====================================================================
                        C  ;    Filename:com1.src
                        C  ;
                        C  ;    This module, com2, and com3 supply INT 14h.
                        C  ;
                        C  ;=====================================================================
                        C
```

```
E729                    C  code segment  public 'ROM'
                        C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    INS8250 Compatible Line Status Bits (ah) for Z8530 SCC Re-Mapping
                        C  ;--------------------------------------------------------------------
                        C
= 0080                  C  com_te                       equ     80h             ; time out error (bit7)
= 0020                  C  com_txd                      equ     20h             ; transmit ready (bit5)
= 0008                  C  com_fe                       equ     08h             ; framing  error (bit3)
= 0004                  C  com_pe                       equ     04h             ; parity   error (bit2)
= 0002                  C  com_oe                       equ     02h             ; overrun  error (bit1)
= 0001                  C  com_rxd                      equ     01h             ; receive  ready (bit0)
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    INS8250 Compatible Modem Status Bits (al) for Z8530 SCC Re-Mapping
                        C  ;--------------------------------------------------------------------
                        C
= 0020                  C  com_dsr              equ     20h                     ; data set ready (bit5)
= 0010                  C  com_cts              equ     10h                     ; clear to send  (bit4)
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    INS8250 Compatible Modem Control Bits.
                        C  ;--------------------------------------------------------------------
                        C
= 0002                  C  com_rts              equ     02h                     ; request to send     (bit1)
= 0001                  C  com_dtr              equ     01h                     ; data terminal ready (bit0)
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Z8530 SCC Status Register (Read Register0)
                        C  ;--------------------------------------------------------------------
                        C
= 0004                  C  scc_txd                      equ     04h             ; transmit ready (bit2)
= 0001                  C  scc_rxd                      equ     01h             ; receive  ready (bit0)
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Z8530 SCC Error Register (Read Register1)
                        C  ;--------------------------------------------------------------------
                        C
= 0040                  C  scc_fe                       equ     40h             ; framing  error (bit6)
= 0020                  C  scc_oe                       equ     20h             ; overrun  error (bit5)
= 0010                  C  scc_pe                       equ     10h             ; parity   error (bit4)
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    INS8250 Asynchronous Communication Chip Baud Rate Time Constants
                        C  ;    (baud rate generator signal is 3.6864 MHz put through a
                        C  ;    divide-by-2 circuit.
                        C  ;
                        C  ;                        ((3,686,400 Hz)/2) = Input Freq.
                        C  ;    Time Constant = -------------------------------
                        C  ;                                (16)*(baud rate)
                        C  ;--------------------------------------------------------------------
                        C
                        C               XORG    0E729h
E729                    C1      org     0E729h
```

```
                        C
E729                    C  com_data1    proc
                        C
E729  0417              C  com_baud     dw      1047    ;   110 baud    (0)
E72B  0300              C              dw      768     ;   150 baud    (1)
E72D  0180              C              dw      384     ;   300 baud    (2)
E72F  00C0              C              dw      192     ;   600 baud    (3)
E731  0060              C              dw      96      ;  1200 baud    (4)
E733  0030              C              dw      48      ;  2400 baud    (5)
E735  0018              C              dw      24      ;  4800 baud    (6)
E737  000C              C              dw      12      ;  9600 baud    (7)
                        C
                        C  com_data1    endp
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;    Z8530 Serial Communication Controller Baud Rate Time Constants
                        C  ;            (baud rate generator signal is 3.6864 MHz)
                        C  ;            (NO divide-by-2 circuit!!!!)
                        C  ;
                        C  ;                    (3,686,400 Hz) = Input Freq.
                        C  ;    Time Constant = ---------------------------  - 2
                        C  ;                        (16)*(2)*(baud rate)
                        C  ;
                        C  ;    NOTE: These values are the SAME as the above EXCEPT for the - 2!!!!
                        C  ;---------------------------------------------------------------------
                        C  ;
                        C  ;scc_baud    dw      1045    ;   110 baud    (0)
                        C  ;            dw      766     ;   150 baud    (1)
                        C  ;            dw      382     ;   300 baud    (2)
                        C  ;            dw      190     ;   600 baud    (3)
                        C  ;            dw      94      ;  1200 baud    (4)
                        C  ;            dw      46      ;  2400 baud    (5)
                        C  ;            dw      22      ;  4800 baud    (6)
                        C  ;            dw      10      ;  9600 baud    (7)
                        C
                        C  ;=====================================================================
                        C  ;    INT 14h -- RS-232 Software Interrupt Request Routine
                        C  ;
                        C  ;    Assumes: INS8250 port addresses are > 256.  That is, the
                        C  ;                    high byte of the port address is nonzero, if and
                        C  ;                    only if, the port is a INS8250. (e.g. com_a ports
                        C  ;                    are 03F8h - 03FFh & com_b ports are 02F8h - 02FFh.)
                        C  ;
                        C  ;    Similarly:      Z8530 port addresses are < 256.  That is, the
                        C  ;                    high byte of the port address is zero, if and
                        C  ;                    only if, the port is a Z8530. (e.g. scc_a ports
                        C  ;                    are 0050h - 0051h & scc_b ports are 0052h - 0053h.)
                        C  ;
                        C  ;       Z8530 Note:     For the reset during power-up, DTR and RTS must be
                        C  ;                    set low which is the only difference from a normal
                        C  ;                     reset (AH=0).  This is accomplished by a special
                        C  ;                     function code (AH=0FFh).
                        C  ;=====================================================================
                        C
                        C       XORG    0E739h
```

```
E739                    C1      org     0E739h
                        C
E739                    C  serial_io    proc    near
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E739  FB                C       sti                                     ; enable interrupts
                        C
E73A  55                C       push    bp                              ; save register
                        C
E73B  83 FA 04          C       cmp     dx,4                            ; 4 RS-232 channels allowed max
E73E  73 3D             C       jae     rs_nop
                        C
E740  8B E8             C       mov     bp,ax                           ; save original function code
E742  80 FC FF          C       cmp     ah,0FFh                         ; power-up reset?
E745  75 02             C       jne     rs_norm                         ; jump if no
E747  32 E4             C       xor     ah,ah                           ; same as reset, BP remembers FF
                        C
E749                    C  rs_norm:
E749  80 FC 03          C       cmp     ah,03h                          ; input out of range?
E74C  77 2F             C       ja      rs_nop
                        C
                        C               assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
E74E  52                C       push    dx                              ; save registers
E74F  51                C       push    cx
E750  53                C       push    bx
E751  1E                C       push    ds                              ; save ds
E752  2E: 8E 1E E574 R  C       mov     ds,word ptr cs:[set_ds_word]    ; satisfy assumptions
                        C
E757  8B DA             C       mov     bx,dx                                   ; get port number (0-3)
E759  33 C9             C       xor     cx,cx                                   ; clear ch
E75B  8A 8F 007C R      C       mov     cl,byte ptr ds:[bx+serial_t_out]        ; get RS-232 time-out
E75F  D1 E3             C       shl     bx,1                                    ; make word index
E761  8B 97 0000 R      C       mov     dx,word ptr ds:[bx+rs232_addr]          ; get address of RS-232
                        C                                                       ; data port
E765  1F                C       pop     ds                                      ; restore ds
                        C
                        C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E766  0B D2             C       or      dx,dx                           ; RS-232 port present?
E768  74 10             C       jz      rs_ret                          ; if not, leave
                        C
E76A  0A F6             C       or      dh,dh                           ; are we a INS8250 chip?
E76C  75 03             C       jnz     rs_ok                           ; if so, take jump
                        C
E76E  80 C4 04          C       add     ah,4                            ; if SCC Z8530 and 4 to function
                        C
E771                    C  rs_ok:
E771  8A DC             C       mov     bl,ah                           ; bx = function number
E773  D1 E3             C       shl     bx,1                            ; bx = 2*(function number)
E775  2E: FF 97 E77F R  C       call    cs:[bx+(offset rs_tbl)]          ; perform rs232 function
                        C
E77A                    C  rs_ret:
E77A  5B                C       pop     bx                              ; restore registers
E77B  59                C       pop     cx
```

```
E77C  5A              C       pop     dx
E77D                  C  rs_nop:
E77D  5D              C       pop     bp
E77E  CF              C       iret
                      C
                      C  ;--------------------------------------------------------------------
                      C  ;    INT 14h Jump Table
                      C  ;--------------------------------------------------------------------
                      C
E77F  E78F R          C  rs_tbl       dw      com_init ; ah = 00h for INS8250
E781  E8F0 R          C       dw      com_pb           ; ah = 01h for INS8250
E783  E92B R          C       dw      com_gb           ; ah = 02h for INS8250
E785  E881 R          C       dw      com_stat ; ah = 03h for INS8250
E787  F6CB R          C       dw      scc_init ; ah = 00h for SCC Z8530
E789  E91C R          C       dw      scc_pb           ; ah = 01h for SCC Z8530
E78B  E94E R          C       dw      scc_gb           ; ah = 02h for SCC Z8530
E78D  E88C R          C       dw      scc_stat ; ah = 03h for SCC Z8530
                      C
                      C  serial_io    endp
                      C
                      C
                      C  ;--------------------------------------------------------------------
                      C  ;    Initialize RS-232 Interface.
                      C  ;
                      C  ;       Input:  al =    input parameters
                      C  ;               dx =    address of RS-232 channel
                      C  ;    Output: ax =    RS-232 channel status
                      C  ;
                      C  ;            al initializes port with: bit 7 6 5 4 3 2 1 0
                      C  ;                                      +-+-+-+-+-+-+-+-+
                      C  ;                                      |B|B|B|P|P|S|D|D|
                      C  ;                                      +-+-+-+-+-+-+-+-+
                      C  ;    Baud (BBB):             Parity (PP):    Stop Bits (S):   Data Bits (DD):
                      C  ;    0 = 110  4 = 1200 x0 = None0 = 1         10 = 7
                      C  ;    1 = 150  5 = 2400 01 = Odd 1 = 2         11 = 8
                      C  ;    2 = 300  6 = 4800 11 = Even              (00 = 5?)
                      C  ;    3 = 600  7 = 9600                                (01 = 6?)
                      C  ;
                      C  ;    Assumes: com_int_x  = com_data_x + 1 = dx + 1
                      C  ;                     com_lctl_x = com_data_x + 3 = dx + 3
                      C  ;--------------------------------------------------------------------
                      C
E78F                  C  com_init     proc    near                    ; ah = 00h
                      C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
                      C
E78F  52              C       push    dx                              ; save dx = com_data_x
                      C
E790  8A E8           C       mov     ch,al                   ; save input parameters.
                      C
E792  B0 80           C       mov     al,080h                         ; access divisor latch of
                      C                                               ; baud count register.
E794  83 C2 03        C       add     dx,3                            ; dx = com_lctl_x
E797  EE              C       out     dx,al                           ; write to line control register
E798  E8 E8E6 R       C       call    rs_dly
                      C
```

```
E79B  8A DD             C       mov     bl,ch                           ; get input parameters.
E79D  81 E3 00E0        C       and     bx,11100000b                    ; get bits5, 6, & 7 (clear bh)
E7A1  B1 04             C       mov     cl,4
E7A3  D2 EB             C       shr     bl,cl                           ; move to bits1,2,& 3
                        C                                               ; bx is word index
E7A5  2E: 8B 87 E729 R  C       mov     ax,word ptr cs:[bx+com_baud]    ; get 8250 baud count
                        C
E7AA  5A                C       pop     dx                              ; restore dx = com_data_x
E7AB  EE                C       out     dx,al                           ; output low byte of baud rate
E7AC  E8 E8E6 R         C       call    rs_dly
                        C
E7AF  8A C4             C       mov     al,ah                           ; transfer high byte to low byte
E7B1  42                C       inc     dx                              ; dx = com_int_x
E7B2  EE                C       out     dx,al                           ; output high byte of baud rate
E7B3  E8 E8E6 R         C       call    rs_dly
                        C
E7B6  8A C5             C       mov     al,ch                           ; get input parameters.
E7B8  24 1F             C       and     al,00011111b                    ; get bits0 thru4
E7BA  42                C       inc     dx
E7BB  42                C       inc     dx                              ; dx = com_lctl_x
E7BC  EE                C       out     dx,al                           ; write to line control register
E7BD  E8 E8E6 R         C       call    rs_dly                          ; disable access divisor latch,
                        C                                               ; set data & stop bits, & parity
                        C
E7C0  32 C0             C       xor     al,al                           ; disable all interrupts!!
E7C2  4A                C       dec     dx
E7C3  4A                C       dec     dx                              ; dx = com_int_x
E7C4  EE                C       out     dx,al                           ; write to interrupt ID register
                        C
E7C5  4A                C       dec     dx                              ; dx = com_data_x
E7C6  E9 E881 R         C       jmp     com_stat                        ; return status
                        C
                        C com_init      endp
                        C
E7C9                    C code ends

                        C include       fdu4.asm
                        C
                        C ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                        C
E7C9                    C code segment  public   'ROM'
                        C       assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
E7C9                    C f_nec_reset   proc     near
                        C
E7C9  BA 03F4           C       mov     dx,f_nec_status                 ; NEC status port
E7CC  EC                C       in      al,dx
E7CD  3C 10             C       cmp     al,10h                          ; NEC busy.
E7CF  75 0E             C       jne     f_nec_reset_ret                 ; no.
E7D1  A0 0041 R         C       mov     al,diskette_status              ; save from previous operation.
E7D4  50                C       push    ax
E7D5  33 C0             C       xor     ax,ax                           ; reset call.
E7D7  8B D0             C       mov     dx,ax
E7D9  CD 13             C       int     13h
E7DB  58                C       pop     ax
```

```
E7DC  A2 0041 R          C       mov     diskette_status,al         ; restore from previous op.
E7DF                     C  f_nec_reset_ret:
E7DF  C3                 C       ret
                         C
                         C  f_nec_reset   endp
                         C
                         C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         C  ;
                         C  ;        (f_motor_on)
                         C  ;
                         C  ;    INPUT:            none
                         C  ;
                         C  ;    OUTPUT:           Carry set if motor was off, cleared otherwise.
                         C  ;
                         C  ;    DESTROYS:AX, CX
                         C  ;
                         C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         C
E7E0                     C  f_motor_on    proc     near
                         C
E7E0  C6 06 0040 R FF    C       mov     motor_count,0FFh           ; max. time for motor.
E7E5  8A 4E 00           C       mov     cl,f_drive                     ; drive.
E7E8  B0 01              C       mov     al,1
E7EA  D2 E0              C       shl     al,cl                               ; mask for motor status.
E7EC  84 06 003F R       C       test    al,motor_status                     ; test always clears carry.
E7F0  75 12              C       jnz     f_mo_ret                   ; already running.
E7F2  A2 003F R          C       mov     motor_status,al                     ; set correct bit.
E7F5  8A E1              C       mov     ah,cl                               ; drive into AH.
E7F7  B1 04              C       mov     cl,4
E7F9  D2 E0              C       shl     al,cl                               ; motor on bit to high nibble.
E7FB  0C 0C              C       or      al,0Ch                              ; set bits 2 & 3 (0000 1100).
E7FD  0A C4              C       or      al,ah                               ; drive bits ( 0 & 1).
E7FF  BA 03F2            C       mov     dx,f_motor_port
E802  EE                 C       out     dx,al                               ; turn on the motor.
E803  F9                 C       stc                                    ; motor was off indicator.
                         C
E804                     C  f_mo_ret:
E804  C3                 C       ret
                         C  f_motor_on    endp
                         C
                         C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         C  ;
                         C  ;        Check Valid (f_check_valid)
                         C  ;
                         C  ;    This routine determines the validity of the request based on
                         C  ;    drive type (40 or 80 trk) and media type (40 or 80 trk)
                         C  ;
                         C  ;    Bit 7 of sys_conf_b port:  0 = 40 trk drive, 1 = 80 trk drive
                         C  ;    Bit 7 of head parameter(DH):       0 = 40 trk media, 1 = 80 trk media
                         C  ;
                         C  ;                            drive
                         C  ;                        |-----------|
                         C  ;                        |  0  |  1  |
                         C  ;                     ---|-----------|
                         C  ;                     | 0 | r/w | r/w |
```

```
                        C  ;            media     ---|-----------|
                        C  ;                      | 1 | n/a | r/w |
                        C  ;                      ---|-----------|
                        C  ;
                        C  ;    OUTPUT:           The MSB of the drive parameter at bp+1 is set
                        C  ;                      if the drive is 80 track and the media is 40 track.
                        C  ;                      This is used by seek to determine if double stepping
                        C  ;                      is necessary.
                        C  ;
                        C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                        C
E805                    C  f_check_valid proc     near
                        C
E805  50                C       push    ax
E806  E4 67             C       in      al,sys_conf_b                 ; switch port
E808  24 01             C       and     al,1                          ; blow off high 7 bits
E80A  D0 C8             C       ror     al,1                          ; move it into the high bit
E80C  30 46 01          C       xor     f_head,al             ; head param. (xor clears carry)
E80F  79 0C             C       jns     f_cv_ret              ; media & drive agree
E811  A8 80             C       test    al,80h                        ; Is drive 80 track.
E813  75 08             C       jnz     f_cv_ret              ; It was.
E815  C6 06 0041 R 01   C       mov     diskette_status,cmd_error
E81A  E9 ECA1 R         C       jmp     f_io_ret
E81D                    C  f_cv_ret:
E81D  58                C       pop     ax
E81E  C3                C       ret
                        C
                        C  f_check_valid endp
                        C
E81F                    C  code ends

                        C  include kb1.asm
                        C
                        C  ;=================================================================
                        C  ;    Filename:kb1.src
                        C  ;
                        C  ;    This module includes INT 16h.
                        C  ;
                        C  ;=================================================================
                        C
E81F                    C  code segment  public 'ROM'
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C  ;=================================================================
                        C  ;    INT 16h -- i8041A Keyboard Software Interrupt Request Routine
                        C  ;=================================================================
                        C  ;
                        C  ;    Input:  ah       = function number (00h <= ah <= 03h)
                        C  ;    Output: ah       = (ah - 2) if ah >= 2
                        C  ;
                        C  ;    Trash:  None. (bx & ds if ROM stack)
                        C  ;
                        C  ;    Note:   The stack never gets deeper than 6 words!!!
                        C  ;
                        C  ;                                          High Address
```

```
                        C  ;                    |------------------| <-- sp (at entry & exit)
                        C  ;                    | return fsw flags |
                        C  ;                    |------------------|
                        C  ;                    | return cs segment|
                        C  ;                    |------------------|
                        C  ;                    | return ip offset |
                        C  ;                    |------------------| <-- sp after kb trap
                        C  ;                    |        ds        |
                        C  ;                    |------------------|
                        C  ;                    |        bx        |
                        C  ;                    |------------------|
                        C  ;                    |    1 near call   |
                        C  ;                    |------------------| <-- sp (at its deepest!!)
                        C  ;                                         Low Address
                        C  ;
                        C  ;=============================================================================
                        C
                        C       XORG    0E82Eh
E82E                    C1      org     0E82Eh
                        C
E82E                    C  k_io proc    near
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E82E  FB                C       sti                                     ; enable interrupts!!
E82F  1E                C       push    ds                              ; save ds
E830  2E: 8E 1E E574 R  C       mov     ds,word ptr cs:[set_ds_word]    ; avoid potential stack problems
E835  53                C       push    bx                              ; save bx
                        C
                        C       assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
E836  80 FC 01          C       cmp     ah,1                            ; ah <= 1 ?
E839  72 0A             C       jb      k_read                          ; jump if ah = 0
E83B  74 25             C       je      k_look                          ; jump if ah = 1
E83D  80 FC 02          C       cmp     ah,2                            ; ah=2 ?
E840  74 29             C       je      k_stat
                        C
E842  5B                C  k_ret:       pop     bx                              ; restore registers
E843  1F                C       pop     ds
E844  CF                C       iret
                        C
                        C  k_io endp
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Wait for key and extract if from the keyboard buffer.
                        C  ;
                        C  ;    Output:  ah = raw scan code
                        C  ;             al = ASCII translated key
                        C  ;                     or
                        C  ;             ah =   translated key
                        C  ;             al = 00h
                        C  ;
                        C  ;    Trash:   None.
                        C  ;--------------------------------------------------------------------
                        C
E845                    C  k_read       proc    near
```

```
                        C       assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
E845  FB                C       sti                                     ; enable interrupts (again)
E846  E8 E856 R         C       call    k_see                           ; is there a character present?
                        C                                               ; interrupts come back disabled!
                        C
E849  75 02             C       jnz     k_1                             ; jump if input in buffer
E84B  EB F8             C       jmp     k_read
                        C
E84D                    C  k_1:
E84D  E8 E870 R         C       call    k_adv_ptr               ; move pointer to next position
E850  89 1E 001A R      C       mov     word ptr ds:[buffer_head],bx    ; store value in variable
E854  EB EC             C       jmp     short k_ret
                        C
                        C
E856  FA                C  k_see:       cli                                     ; disable interrupts!!
E857  8B 1E 001A R      C       mov     bx,word ptr ds:[buffer_head]    ; get pointer to head of buffer
E85B  3B 1E 001C R      C       cmp     bx,word ptr ds:[buffer_tail]    ; if equal, then nothing there
E85F  8B 07             C       mov     ax,word ptr ds:[bx]             ; get scan code and ascii code
E861  C3                C       ret
                        C
                        C  k_read       endp
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Checks for key in keyboard buffer, but does not extract it.
                        C  ;
                        C  ;    Output:  if key is in buffer, then:
                        C  ;                     zf = 0 (nz = reset)
                        C  ;                     ah = raw scan code
                        C  ;                     al = ASCII translated key
                        C  ;                             or
                        C  ;                     ah =   translated key
                        C  ;                     al = 00h
                        C  ;             else:
                        C  ;                     zf = 1 (z = set)
                        C  ;                     ax = 16th previous key
                        C  ;
                        C  ;    Trash:   ax is trashed if keyboard buffer is empty.
                        C  ;--------------------------------------------------------------------
                        C
E862                    C  k_look       proc    far     ; must be far!!!!
                        C       assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
E862  E8 E856 R         C       call    k_see                           ; is there a character present?
                        C                                               ; interrupts come back disabled!
E865  FB                C       sti                                     ; must return interrupts enabled
E866  5B                C       pop     bx                              ; restore registers
E867  1F                C       pop     ds                              ; blow away flags returning:
E868  CA 0002           C       ret     2                               ; zf & ax = k_see output, & sti
                        C
                        C  k_look       endp
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Returns keyboard shift state kb_flag in al.
                        C  ;
```

```
                        C  ;    Output:  ah = 0.
                        C  ;             al = kb_flag
                        C  ;    Trash:   None.
                        C  ;--------------------------------------------------------------------
                        C
E86B                    C  k_stat        proc    near
                        C       assume   cs:code, ds:data, es:nothing, ss:nothing
                        C
E86B  A0 0017 R         C       mov      al,byte ptr ds:[kb_flag]   ; get the shift status flags
E86E  EB D2             C       jmp      short k_ret
                        C
                        C  k_stat        endp
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Advances kb_buffer ring buffer pointer.
                        C  ;
                        C  ;    Input:   bx
                        C  ;    Output:  bx
                        C  ;    Trash:   None.
                        C  ;--------------------------------------------------------------------
                        C
E870                    C  k_adv_ptr     proc    near
                        C                assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
E870  43                C       inc      bx                                 ; move to next word in list
E871  43                C       inc      bx
E872  3B 1E 0082 R      C       cmp      bx,word ptr ds:[buffer_end]; end of buffer ?
E876  75 04             C       jne      k_adv_end                  ; no, continue
E878  8B 1E 0080 R      C       mov      bx,word ptr ds:[buffer_start]      ; yes, buffer to beginning
E87C                    C  k_adv_end:
E87C  C3                C       ret
                        C
                        C  k_adv_ptr     endp
                        C
E87D                    C  code ends


                        C  include comm2.asm
                        C
                        C  ;====================================================================
                        C  ;    Filename:com2.src
                        C  ;
                        C  ;====================================================================
                        C
E87D                    C  code segment  public 'ROM'
                        C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;       Read Status of RS-232 Interface.  (rs_stat)
                        C  ;
                        C  ;    Input:   dx =     if dh =  0, then address of Z8530 channel (scc_ctl_x).
                        C  ;                      if dh <> 0, then address of 8250 data port (com_data_x).
                        C  ;    Output:  ax =     RS-232 (INS8250-compatible) channel status.
                        C  ;    Trash:   None.
                        C  ;
```

```
                        C  ;    Assumes: com_lstat_x = com_data_x + 5 = dx + 5 (line status)
                        C  ;                     com_mstat_x = com_data_x + 6 = dx + 6 (modem status)
                        C  ;--------------------------------------------------------------------------
                        C
E87D                    C  rs_stat      proc    near                    ; ah = 03h
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E87D  0A F6             C       or      dh,dh                   ; are we a SCC Z8530 chip?
E87F  74 0B             C       jz      scc_stat        ; if so, take jump
                        C
                        C
E881                    C  com_stat:                            ; INS8250 read status routine.
                        C
                        C  ; Get Line Status.
                        C
E881  52                C       push    dx                      ; save dx = com_data_x
E882  83 C2 05          C       add     dx,5                    ; dx = com_lstat_x
E885  EC                C       in      al,dx                   ; get line status
E886  8A E0             C       mov     ah,al                   ; line status comes back in high byte
                        C
                        C  ; Get Modem Status.
                        C
E888  42                C       inc     dx                      ; dx = com_mstat_x
E889  EC                C       in      al,dx                   ; get modem status in low byte
E88A  5A                C       pop     dx                      ; restore dx = com_data_x
E88B  C3                C       ret
                        C
                        C
E88C                    C  scc_stat:                            ; SCC Z8530 read status routine.
                        C
                        C  ; Get Channel Status.
                        C
E88C  33 C0             C       xor     ax,ax                   ; al = selects 0 ; ah = no errors
E88E  EE                C       out     dx,al                   ; dx = scc_ctl_x
E88F  E8 E8E6 R         C       call    rs_dly
E892  EC                C       in      al,dx                   ; get channel status
                        C
E893  A8 04             C       test    al,scc_txd              ; test for transmit ready
E895  74 03             C       jz      scc_no_txd
                        C
E897  80 CC 20          C       or      ah,com_txd              ; if so, set INS8250-compatible bit.
E89A                    C  scc_no_txd:
                        C
E89A  A8 01             C       test    al,scc_rxd              ; test for receive ready
E89C  74 03             C       jz      scc_no_rxd
                        C
E89E  80 CC 01          C       or      ah,com_rxd              ; if so, set INS8250-compatible bit.
E8A1                    C  scc_no_rxd:
                        C
                        C  ; Get Error Status.
                        C
E8A1  B0 01             C       mov     al,1                    ; get errors
E8A3  EE                C       out     dx,al                   ; dx = scc_ctl_x
E8A4  E8 E8E6 R         C       call    rs_dly
E8A7  EC                C       in      al,dx                   ; get error status
```

```
                        C
E8A8  A8 40             C       test    al,scc_fe           ; test for framing error
E8AA  74 03             C       jz      scc_no_fe
                        C
E8AC  80 CC 08          C       or      ah,com_fe           ; if so, set INS8250-compatible bit.
E8AF                    C  scc_no_fe:
                        C
E8AF  A8 10             C       test    al,scc_pe           ; test for parity error
E8B1  74 03             C       jz      scc_no_pe
                        C
E8B3  80 CC 04          C       or      ah,com_pe           ; if so, set INS8250-compatible bit.
E8B6                    C  scc_no_pe:
                        C
E8B6  A8 20             C       test    al,scc_oe           ; test for overrun error
E8B8  74 03             C       jz      scc_no_oe
                        C
E8BA  80 CC 02          C       or      ah,com_oe           ; if so, set INS8250-compatible bit.
E8BD                    C  scc_no_oe:
                        C
                        C  ; Set Modem Status.
                        C
E8BD  B0 30             C       mov     al,(com_dsr+com_cts)    ; al = DSR and CTS
E8BF  C3                C       ret
                        C
                        C  rs_stat      endp
                        C
                        C  ;--------------------------------------------------------------------------
                        C  ;       Wait for Status of RS-232 Interface.  (rs_ws)
                        C  ;
                        C  ;    Input:  ah =    RS-232 channel status for which to wait
                        C  ;            cx =    RS-232 time-out
                        C  ;            dx =    if dh =  0, then address of Z8530 channel (scc_ctl_x).
                        C  ;                    if dh <> 0, then address of 8250 data port to poll.
                        C  ;
                        C  ;    Output: AH =    Status.
                        C  ;            ZF =    set,   if status matches.
                        C  ;                    reset, if time-out.
                        C  ;
                        C  ;    Trash:  al & bx destroyed.
                        C  ;
                        C  ;    Assumes: com_lstat_x = com_data_x + 5 = dx + 5 (line status)
                        C  ;                    com_mstat_x = com_data_x + 6 = dx + 6 (modem status)
                        C  ;--------------------------------------------------------------------------
                        C
E8C0                    C  rs_wsproc    near
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E8C0  51                C       push    cx                      ; save time-out
E8C1  D1 E1             C       shl     cx,1                    ; double timeout
E8C3  33 DB             C       xor     bx,bx                   ; clear bx
E8C5  87 CB             C       xchg    cx,bx                   ; BL now has rs232_time_out.
                        C
E8C7                    C  rs_ws_lp:
E8C7  0A F6             C       or      dh,dh                   ; are we an INS8250 chip?
E8C9  75 06             C       jnz     rs_ws_com       ; if so, take jump
```

```
                        C
E8CB  32 C0             C       xor     al,al                   ; al = selects 0 on SCC Z8530
E8CD  EE                C       out     dx,al                   ; dx = scc_ctl_x
E8CE  E8 E8E6 R         C       call    rs_dly
                        C
E8D1                    C  rs_ws_com:
E8D1  EC                C       in      al,dx                   ; get channel status
E8D2  8A F8             C       mov     bh,al                   ; save status in BH.
E8D4  22 C4             C       and     al,ah                   ; mask bits we're waiting for
E8D6  3A C4             C       cmp     al,ah                   ; are they all on?
                        C
E8D8  74 08             C       jz      rs_ws_exit              ; if so, exit with zf set
                        C
E8DA  E2 EB             C       loop    rs_ws_lp        ; inner loop
E8DC  FE CB             C       dec     bl
E8DE  75 E7             C       jnz     rs_ws_lp        ; outer loop
                        C
E8E0  0A E4             C       or      ah,ah                   ; time-out, exit with zf reset
E8E2                    C  rs_ws_exit:
E8E2  8A E7             C       mov     ah,bh                   ; move status to AH.
E8E4  59                C       pop     cx                      ; restore time-out
E8E5  C3                C       ret
                        C
                        C  rs_wsendp
                        C
                        C
E8E6                    C  rs_dly       proc    near
                        C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E8E6  9C                C       pushf
E8E7  51                C       push    cx
E8E8  B9 0008           C       mov     cx,8
E8EB  E2 FE             C  rs_lp:       loop    rs_lp
E8ED  59                C       pop     cx
E8EE  9D                C       popf
E8EF  C3                C       ret
                        C
                        C  rs_dly       endp
                        C
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;       INS8250 Put Byte (com_pb) & SCC Z8530 Put Byte (scc_pb)
                        C  ;
                        C  ;    Transmit Character to RS-232 Interface.
                        C  ;
                        C  ;    Input:  al =    character to transmit
                        C  ;            cx =    RS-232 time-out
                        C  ;            dx =    if dh =  0, then address of Z8530 channel (scc_ctl_x).
                        C  ;                    if dh <> 0, then address of 8250 data port (com_data_x).
                        C  ;    Output: ah =    line status
                        C  ;    Trash:  bx destroyed.
                        C  ;
                        C  ;    Assumes: com_mctl_x  = com_data_x + 4 = dx + 4
```

```
                        C  ;                    com_lstat_x = com_data_x + 5 = dx + 5
                        C  ;                    com_mstat_x = com_data_x + 6 = dx + 6
                        C  ;
                        C  ;                    scc_data_x = scc_ctl_x + 1 = dx + 1
                        C  ;--------------------------------------------------------------------
                        C
E8F0                    C  com_pb       proc    near                    ; ah = 01h
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E8F0  52                C       push    dx                      ; save dx = com_data_x
E8F1  50                C       push    ax                      ; save character to output in al
                        C
E8F2  B0 03             C       mov     al,(com_rts+com_dtr)    ; signal RTS & DTR
E8F4  83 C2 04          C       add     dx,4                    ; dx = com_mctl_x
E8F7  EE                C       out     dx,al                   ; send to modem control register
                        C
E8F8  B4 30             C       mov     ah,(com_dsr+com_cts)    ; wait for DSR & CTS
E8FA  42                C       inc     dx
E8FB  42                C       inc     dx                      ; dx = com_mstat_x
E8FC  E8 E8C0 R         C       call    rs_ws                   ; wait for modem status register
E8FF  75 13             C       jnz     rs_pbe                  ; if time-out, take jump
                        C
E901  B4 20             C       mov     ah,com_txd              ; wait for transmit ready
E903  4A                C       dec     dx                      ; dx = com_lstat_x
E904  E8 E8C0 R         C       call    rs_ws                   ; wait for line status register
E907  75 0B             C       jnz     rs_pbe                  ; if time-out, take jump
                        C
E909  58                C       pop     ax                      ; restore character to output in al
E90A  5A                C       pop     dx                      ; restore dx = com_data_x
E90B  8A D8             C       mov     bl,al                   ; save character input/output in bl
E90D  E8 E87D R         C       call    rs_stat                 ; get return status
E910  8A C3             C       mov     al,bl                   ; restore character input/output in al
E912  EE                C       out     dx,al                   ; else, output the character
                        C
                        C                                       ; exit for put and get byte
E913                    C  rs_pb_gb:                            ; if SCC Z8530, dx = scc_ctl_x
                        C
E913  C3                C       ret
                        C
E914                    C  rs_pbe:                                      ; exit for put byte error
E914  5B                C       pop     bx                      ; restore character to output in bl
E915  5A                C       pop     dx                      ; if SCC Z8530, restore dx = scc_ctl_x
                        C                                       ; else INS8250, restore dx = com_data_x
                        C
                        C  ;;   call    rs_stat                 ; get return status
E916  8A C3             C       mov     al,bl                   ; restore character to output in al
E918  80 CC 80          C       or      ah,com_te       ; indicate timeout error
E91B  C3                C       ret
                        C
                        C  com_pb       endp
                        C
                        C
E91C                    C  scc_pb       proc    near                    ; ah = 01h
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
```

```
E91C  52              C       push    dx                      ; save dx = scc_ctl_x
E91D  50              C       push    ax                      ; save character to output in al
                      C
E91E  B4 04           C       mov     ah,scc_txd              ; wait for transmit ready
E920  E8 E8C0 R       C       call    rs_ws
E923  75 EF           C       jnz     rs_pbe                  ; if time-out, take jump
                      C
E925  58              C       pop     ax                      ; restore character to output in al
E926  42              C       inc     dx                      ; dx = scc_data_x
E927  EE              C       out     dx,al                   ; else, output the character
                      C
E928  5A              C       pop     dx                      ; restore dx = scc_ctl_x
E929  EB E8           C       jmp     short rs_pb_gb
                      C
                      C scc_pb        endp
                      C
                      C ;-----------------------------------------------------------------------
                      C ;
                      C ;       INS8250 Get Byte (com_gb) & SCC Z8530 Get Byte (scc_gb)
                      C ;
                      C ;    Receive Character to RS-232 Interface.
                      C ;
                      C ;    Input:   cx =     RS-232 time-out
                      C ;             dx =     if dh =  0, then address of Z8530 channel (scc_ctl_x).
                      C ;                      if dh <> 0, then address of 8250 data port (com_data_x).
                      C ;    Output:  al =     character received
                      C ;             ah =     line status
                      C ;    Trash:   bx destroyed.
                      C ;
                      C ;    Assumes: com_mctl_x  = com_data_x + 4 = dx + 4
                      C ;                     com_lstat_x = com_data_x + 5 = dx + 5
                      C ;                     com_mstat_x = com_data_x + 6 = dx + 6
                      C ;
                      C ;                     scc_data_x = scc_ctl_x + 1 = dx + 1
                      C ;-----------------------------------------------------------------------
                      C
E92B                  C com_gb        proc    near                    ; ah = 02h
                      C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                      C
E92B  52              C       push    dx                      ; save dx = com_data_x
E92C  B0 01           C       mov     al,com_dtr              ; signal DTR
E92E  83 C2 04        C       add     dx,4                    ; dx = com_mctl_x
E931  EE              C       out     dx,al                   ; send to modem control register
                      C
E932  B4 20           C       mov     ah,com_dsr              ; wait for DSR
E934  42              C       inc     dx
E935  42              C       inc     dx                      ; dx = com_mstat_x
E936  E8 E8C0 R       C       call    rs_ws                   ; wait for modem status register
E939  75 0E           C       jnz     rs_gbe                  ; if time-out, take jump
                      C
E93B  B4 01           C       mov     ah,com_rxd              ; wait for receive ready
E93D  4A              C       dec     dx                      ; dx = com_lstat_x
E93E  E8 E8C0 R       C       call    rs_ws                   ; wait for line status register
E941  75 06           C       jnz     rs_gbe                  ; if time-out, take jump
                      C
```

```
E943  80 E4 0E          C       and     ah,0Eh                  ; Only interested on low nibble.
E946  5A                C       pop     dx                      ; restore dx = com_data_x
E947  EC                C       in      al,dx                   ; else get character
E948  C3                C       ret
                        C
                        C
E949                    C  rs_gbe:                                      ; exit for get byte error
E949  5A                C       pop     dx                      ; if SCC Z8530, restore dx = scc_ctl_x
                        C                                       ; else INS8250, restore dx = com_data_x
E94A  80 CC 80          C       or      ah,com_te       ; indicate timeout error
E94D  C3                C       ret
                        C
                        C  com_gb       endp
                        C
                        C
E94E                    C  scc_gb       proc    near                    ; ah = 02h
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E94E  52                C       push    dx                      ; save dx = scc_ctl_x
                        C
E94F  B4 01             C       mov     ah,scc_rxd              ; wait for receive ready
E951  E8 E8C0 R         C       call    rs_ws
E954  75 F3             C       jnz     rs_gbe                  ; if time-out, take jump
                        C
E956  42                C       inc     dx                      ; dx = scc_data_x
E957  EC                C       in      al,dx                   ; else get character
                        C
E958  5A                C       pop     dx                      ; restore dx = scc_ctl_x
E959  EB B8             C       jmp     short rs_pb_gb
                        C
                        C  scc_gb       endp
E95B                    C  code ends

                        C  include cal_data.asm
                        C
                        C
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;
                        C  ; This data is used by the clock calendar routine to convert from the
                        C  ; values given by the MM58174 chip to human values
                        C  ;
                        C  ; See: rtc.src and calendar.src
                        C  ;
                        C  ;--------------------------------------------------------------------
                        C
                        C
E95B                    C  code segment  public 'ROM'
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
E95B                    C  c_data1      proc
                        C
                        C  ; Days per year.
                        C
E95B  0000              C  c_dy_yr      dw      (0*366)+(0*365)   ; year 0 = leap year + 0
```

```
E95D  016E                C       dw      (1*366)+(0*365)   ; year 1 = leap year + 1
E95F  02DB                C       dw      (1*366)+(1*365)   ; year 2 = leap year + 2
E961  0448                C       dw      (1*366)+(2*365)   ; year 3 = leap year + 3
E963  05B5                C       dw      (1*366)+(3*365)   ; year 4 = leap year + 0
E965  0723                C       dw      (2*366)+(3*365)   ; year 5 = leap year + 1
E967  0890                C       dw      (2*366)+(4*365)   ; year 6 = leap year + 2
E969  09FD                C       dw      (2*366)+(5*365)   ; year 7 = leap year + 3
                          C
                          C  ; Days per month.
                          C
E96B  1F                  C  c_dy_mo      db      31              ; month 0 = Jan
E96C  1C                  C       db      28              ; month 1 = Feb
E96D  1F                  C       db      31              ; month 2 = Mar
E96E  1E                  C       db      30              ; month 3 = Apr
E96F  1F                  C       db      31              ; month 4 = May
E970  1E                  C       db      30              ; month 5 = Jun
E971  1F                  C       db      31              ; month 6 = Jul
E972  1F                  C       db      31              ; month 7 = Aug
E973  1E                  C       db      30              ; month 8 = Sep
E974  1F                  C       db      31              ; month 9 = Oct
E975  1E                  C       db      30              ; month A = Nov
E976  1F                  C       db      31              ; month B = Dec
                          C
                          C  c_data1      endp
E977                      C  code ends
                          C

                          C  include kb2.asm
                          C
                          C  ;=============================================================================
                          C  ;    Filename:kb2.src
                          C  ;
                          C  ;    This module includes INT 09h
                          C  ;
                          C  ;=============================================================================
                          C
E977                      C  code segment  public 'ROM'
                          C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                          C
                          C       XORG     0E987h
E987                      C1      org      0E987h
                          C
                          C
                          C  ;=============================================================================
                          C  ;    INT 09h -- i8041A Keyboard Hardware Interrupt Service Routine
                          C  ;
                          C  ;    Note:    'make'   -> key is depressed -> 00h + key scan code
                          C  ;             'break'  -> key is released  -> 80h + key scan code
                          C  ;=============================================================================
                          C
                          C
E987                      C  k_intproc     near
                          C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                          C
E987  FB                  C       sti                                     ; re-enable interrupts
```

```
E988  FC                  C       cld                                     ; clear direction flag
                          C
E989  50                  C       push    ax
E98A  53                  C       push    bx
E98B  51                  C       push    cx
E98C  52                  C       push    dx
E98D  56                  C       push    si
E98E  57                  C       push    di
E98F  1E                  C       push    ds
E990  2E: 8E 1E E574 R    C       mov     ds,word ptr cs:[set_ds_word]    ; avoid potential stack problems
E995  06                  C       push    es
                          C
                          C               assume  cs:code, ds:data, es:nothing, ss:nothing
                          C
                          C ; Get the scan code.
                          C
E996  E4 60               C       in      al,p_kscan                      ; get scan code from data port
E998  8A E0               C       mov     ah,al                           ; save scan code in ah
                          C
                          C ; Reset the keyboard.
                          C
E99A  BA 0061             C       mov     dx,p_kctrl                      ; get control port address
E99D  EC                  C       in      al,dx                           ; get the status
E99E  8A D8               C       mov     bl,al                           ; save the status in bl
                          C
E9A0  0C 80               C       or      al,080h                         ; set bit7 -- reset
E9A2  EE                  C       out     dx,al                           ; reset keyboard
                          C
E9A3  8A C3               C       mov     al,bl                           ; retrieve original status
E9A5  EE                  C       out     dx,al                           ; send it back to keyboard
                          C
                          C ; Retrieve the scan code in both al & ah.
                          C
E9A6  8A C4               C       mov     al,ah                           ; scan code in both registers
                          C
                          C ; Test for overrun scan code from keyboard = 0FFh.
                          C ; Note:  20 key scan codes can be buffered up by the   keyboard.
                          C
E9A8  3C FF               C       cmp     al,0FFh                         ; an overrun scan code?
E9AA  75 06               C       jnz     k_ok                            ; if not, continue
                          C
E9AC  E8 EBD7 R           C       call    k_beep                          ; beep the speaker
E9AF  E9 EA5C R           C       jmp     k_nop
E9B2                      C k_ok:
                          C
                          C ;---------------------------------------------------------------------
                          C ;   ah      = scan code (make/break)   al = scan code (make/break)
                          C ;   bx      = ?
                          C ;   cx      = ?
                          C ;   dx      = ?
                          C ;   es:di   = ?
                          C ;---------------------------------------------------------------------
                          C
E9B2  24 7F               C       and     al,07Fh                         ; al = scan code make
                          C
```

```
E9B4  0E                C       push    cs
E9B5  07                C       pop     es                              ;segment for p_kscan table
E9B6  BF CC67 R         C       mov     di,offset kb_data_table         ; es:di points to p_kscan table
                        C
E9B9  33 DB             C       xor     bx,bx                           ; clear bh
E9BB  33 C9             C       xor     cx,cx                           ; clear cl
E9BD  8A 2E 0017 R      C       mov     ch,byte ptr ds:[kb_flag]
                        C
E9C1  8A D8             C       mov     bl,al                           ; bx = scan code make
E9C3  D1 E3             C       shl     bx,1                            ; bx = 2*(scan code make)
E9C5  D1 E3             C       shl     bx,1                            ; bx = 4*(scan code make)
                        C
                        C ;--------------------------------------------------------------------
                        C ;     ah      = scan code (make/break)   al = scan code (make)
                        C ;     bx      = 4*(scan code make) = 4*al
                        C ;     ch      = kb_flag                  cl = 0
                        C ;     dx      = ?
                        C ;     es:di   = p_kscan base
                        C ;--------------------------------------------------------------------
                        C
E9C7  4B                C       dec     bx                              ; bx = 4*(scan code make)-1
                        C
E9C8  F6 C5 08          C       test    ch,alt_shift                    ; alt state?
E9CB  75 32             C       jnz     k_ix                            ; if so, has highest priority
                        C
E9CD  4B                C       dec     bx                              ; bx = 4*(scan code make)-2
                        C
E9CE  F6 C5 04          C       test    ch,cntrl_shift                  ; control state?
E9D1  75 2C             C       jnz     k_ix                            ; if so, next highest priority
                        C
E9D3  4B                C       dec     bx                              ; bx = 4*(scan code make)-3
                        C
                        C ; Handle CapLk Case.
                        C
E9D4  3C 37             C       cmp     al,55                           ; 0 <= scan code <= (7*8)-1
E9D6  77 0C             C       ja      k_no_cap                ; test NumLk case.
                        C
E9D8  F6 C5 40          C       test    ch,caps_lock_mode               ; caplock state?
E9DB  74 1C             C       jz      k_no_lock               ; if not, test shift states
                        C
E9DD  E8 EB98 R         C       call    k_bit                           ; get kb_cap_flags bit in cf
E9E0  73 17             C       jnb     k_no_lock               ; (jnc) swap if cf set
                        C
E9E2  EB 0D             C       jmp     short k_lock                    ; honor caps lock.
                        C
E9E4                    C k_no_cap:
                        C
                        C ; Handle NumLk Case.
                        C
E9E4  3C 47             C       cmp     al,71                           ; scan code >= 71?
E9E6  72 11             C       jb      k_no_lock
E9E8  3C 53             C       cmp     al,83                           ; scan code <= 83?
E9EA  77 0D             C       ja      k_no_lock
E9EC  F6 C5 20          C       test    ch,num_lock_mode                ; numlock state?
E9EF  74 08             C       jz      k_no_lock               ; if not, test shift states
```

```
                        C
E9F1                    C k_lock:                                       ; either caps or num lock.
E9F1  F6 C5 03          C       test    ch,(left_shift+right_shift); if so, and shift state
E9F4  74 09             C       jz      k_ix                            ; also true
                        C
E9F6  4B                C       dec     bx                              ; bx = 4*(scan code make)-4
E9F7  EB 06             C       jmp     short k_ix                      ; (base case)
                        C
E9F9                    C k_no_lock:                                    ; neither caps or num lock.
E9F9  F6 C5 03          C       test    ch,(left_shift+right_shift); (shift state)
E9FC  75 01             C       jnz     k_ix
E9FE  4B                C       dec     bx                              ; bx = 4*(scan code make)-4
                        C ;     jmp     short k_ix                      ; (base case) fall through
                        C
                        C
E9FF                    C k_ix:                                         ; bx = index into p_kscan table
E9FF  D1 E3             C       shl     bx,1                            ; bx = p_kscan word index
EA01  03 FB             C       add     di,bx                           ; add p_kscan base & index
                        C
EA03  26: 8B 15         C       mov     dx,word ptr es:[di]             ; get data word from table
                        C
EA06  33 DB             C       xor     bx,bx                           ; clear bh
EA08  8A DA             C       mov     bl,dl                           ; move translated   key to bx
EA0A  F6 06 0018 R 01   C       test    byte ptr ds:[kb_flag_1].dlx_kb  ; are we a deluxe keyboard
EA0F  74 02             C       jz      k_xlat                          ;   key
                        C
EA11  8A DE             C       mov     bl,dh                       ; move translated deluxe
                        C                                               ;   key to key
EA13                    C k_xlat:                                           ; bx has translated byte (bh=0)
                        C
                        C ;---------------------------------------------------------------------
                        C ;     ah      = scan code (make/break)   al = scan code (make)
                        C ;     bx      = deluxe keyboard translated byte (bh = 0)
                        C ;     ch      = kb_flag                  cl = 0
                        C ;     dx      = es:[di] (could be deluxe key code)
                        C ;     es:di   = p_kscan base + p_kscan scan code word index
                        C ;---------------------------------------------------------------------
                        C
                        C ; Registers are all loaded up.  Start going through the cases.
                        C
EA13  0A D2             C       or      dl,dl                           ; deluxe scan codes are
EA15  74 1C             C       jz      k_no_case               ; NEVER special cases!!!!
                        C
                        C ; Test for special cases.
                        C
EA17  80 FB C0          C       cmp     bl,0C0h                         ; xlated byte special case?
EA1A  72 17             C       jb      k_no_case               ; if not, handle unspecial case
EA1C  80 FB D8          C       cmp     bl,0D8h                         ; 0C0h <= xlated byte <= 0D8h
EA1F  77 12             C       ja      k_no_case               ; if so, do the special function
                        C
                        C ; Test for 'break' of special case.
                        C
EA21  80 FB C8          C       cmp     bl,0C8h                         ; is xlated byte a shift key?
EA24  72 04             C       jb      k_jmp                           ; if so, do 'break' of shift key
                        C                                               ; 0C0h <= xlated byte < 0C8h
```

```
EA26  0A E4              C       or      ah,ah
EA28  78 2E              C       js      k_none                          ; nothing for 'break' of others.
                         C
EA2A                     C  k_jmp:                                               ; jump to special case routine.
EA2A  8B F3              C       mov     si,bx                           ; if so, si gets special index
EA2C  D1 E6              C       shl     si,1                            ; make it a word index
EA2E  2E: FF A4 EA90 R   C       jmp     cs:[si+((offset k_case)-(2*0C0h))]
                         C
EA33                     C  k_no_case:
                         C
                         C  ; Test for 'break' of non special case.
                         C
EA33  0A E4              C       or      ah,ah
EA35  78 21              C       js      k_none                          ; do nothing if 'break' of key.
                         C
                         C  ; Test for 'unpause' case.
                         C
EA37  F6 06 0018 R 08    C       test    byte ptr ds:[kb_flag_1],pause_mode  ; are we in hold state?
EA3C  74 0B              C       jz      k_no_hold                       ; if not, continue.
EA3E  3C 45              C       cmp     al,num_lock_key                         ; don't clear hold state
EA40  74 16              C       je      k_none                                  ; on num_lock_key (Pause).
EA42  80 26 0018 R F7    C       and     byte ptr ds:[kb_flag_1],not pause_mode  ; reset hold state bit &
EA47  EB 0F              C       jmp     short k_none                    ; ignore the key.
                         C
EA49                     C  k_no_hold:
                         C
                         C  ; Test for deluxe scan code.
                         C
EA49  0A D2              C       or      dl,dl
EA4B  75 04              C       jnz     k_no_xcode
                         C
EA4D  8B C2              C       mov     ax,dx                           ; move deluxe key to scan code
EA4F  EB 02              C       jmp     short k_buf                     ; put ax into the buffer
                         C
EA51                     C  k_no_xcode:
EA51  8A C3              C       mov     al,bl                           ; move translated key to key
                         C
EA53                     C  k_buf:                                               ; put ax into kb_buffer.
                         C                                               ; try to put ax into kb_buffer.
EA53  E8 EBBC R          C       call    k_try                           ; zf set (z)   if buffer is full
EA56  74 04              C       jz      k_nop                           ; zf reset (nz) if buffer got ax
                         C                                               ; Signal keybrd input available
                         C
EA58                     C  k_none:                                              ; scan code translate to nothing
                         C
                         C  ; Send specific end of interrupt (SEOI) to pic 'command' port.
                         C
EA58  FA                 C       cli                                     ;prevent interrupt til stack clear
EA59  E8 EBB7 R          C       call    k_eoi                           ; send specific end of interrupt
                         C
EA5C                     C  k_nop:                                               ; restore registers without issuing SEOI
EA5C  07                 C       pop     es
EA5D  1F                 C       pop     ds
EA5E  5F                 C       pop     di
EA5F  5E                 C       pop     si
```

```
EA60  5A                  C       pop     dx
EA61  59                  C       pop     cx
EA62  5B                  C       pop     bx
EA63  58                  C       pop     ax
EA64  CF                  C       iret
                          C
                          C ;--------------------------------------------------------------------
                          C ;   Test for system reset sequence.  'make' only.
                          C ;--------------------------------------------------------------------
                          C
EA65  80 E5 0C            C k_res:        and     ch,(alt_shift+cntrl_shift)
EA68  80 FD 0C            C       cmp     ch,(alt_shift+cntrl_shift)          ; is it CTL ALT shift?
      75 EB               C       jne     k_none
                          C
                          C ; CTL ALT DEL system reset.
                          C
EA6D  C7 06 0072 R 1234   C       mov     word ptr ds:[reset_flag],01234h             ; set flag for warm boot
EA73  E9 DB8F R           C       jmp     diagnostics_1                               ; re-boot
                          C
                          C ;--------------------------------------------------------------------
                          C ;   Pause waiting for another key.  'make' only.
                          C ;--------------------------------------------------------------------
                          C
EA76                      C k_pause:
EA76  80 0E 0018 R 08     C       or      byte ptr ds:[kb_flag_1],pause_mode  ; set the pause bit.
                          C
EA7B  E8 EBB7 R           C       call    k_eoi                               ; send specific end of interrupt
EA7E  FB                  C       sti                                         ; defensive programming measure
                          C
                          C ; Note: Video not disabled during vertical retrace.
                          C
EA7F  80 3E 0049 R 07     C       cmp     byte ptr ds:[v_mode],7      ; never on a monochrome card
EA84  74 0B               C       je      k_hold
                          C
EA86  8B 16 0063 R        C       mov     dx,word ptr ds:[v_base6845]; get 6845 pointer register
EA8A  83 C2 04            C       add     dx,4                                ; get 6845 mode control register
EA8D  A0 0065 R           C       mov     al,byte ptr ds:[v_3x8]      ; get the video mode last sent
EA90  EE                  C       out     dx,al                               ; enable video
                          C
EA91  F6 06 0018 R 08     C k_hold:       test    byte ptr ds:[kb_flag_1],pause_mode  ; test the pause bit.
EA96  75 F9               C       jnz     k_hold                              ; loop until pause bit cleared
EA98  EB C2               C       jmp     short k_nop
                          C
                          C ;--------------------------------------------------------------------
                          C ;   Print Screen sequence.  'make' only.
                          C ;--------------------------------------------------------------------
                          C
EA9A  E8 EBB7 R           C k_prt:        call    k_eoi   ; send specific end of interrupt
                          C       sti             ;allow more keys to be pressed
                          C                                                   ; while printing screen.
EA9D  CD 05               C       INT     5h                                  ; issue print screen interrupt
EA9F  EB BB               C       jmp     short k_nop
                          C
                          C ;--------------------------------------------------------------------
                          C ;   Deluxe code put NUL into kb_buffer.  'make' only.
```

```
                        C  ;--------------------------------------------------------------------
                        C
EAA1  B8 0300           C  k_nul:       mov     ax,0300h                ; NUL=X03
EAA4  EB AD             C       jmp     short k_buf                     ; put ax into the buffer
                        C
                        C  ;--------------------------------------------------------------------
                        C  ; Four state shifts. 'make' & 'break' in kb_flag_1 plus history in kb_flag.
                        C  ;--------------------------------------------------------------------
                        C
EAA6  B0 80             C  k_ins:       mov     al,insert_shift                 ; INS toggle lock
EAA8  E8 EAD8 R         C       call    k_4tog                          ; toggle 4 state
                        C
EAAB  0A E4             C       or      ah,ah                           ; is INS toggle 'make' ?
EAAD  78 A9             C       js      k_none                          ; if not, exit
EAAF  B8 5200           C       mov     ax,(insert_key*100h)+00h        ; else, ax gets deluxe INS key
EAB2  EB 9F             C       jmp     k_buf                           ; put ax into the buffer
                        C
                        C
EAB4  B0 40             C  k_cap:       mov     al,caps_lock_shift      ; CAPS LOCK toggle lock
EAB6  E8 EAD8 R         C       call    k_4tog                          ; toggle 4 state
EAB9  B1 01             C       mov     cl,00000001b                    ; CAPS LOCK LED is bit0.
EABB  E8 EB76 R         C       call    k_LED_cap               ; send LED information
EABE  EB 98             C  k_non1:      jmp     short k_none
                        C
                        C
EAC0  B0 20             C  k_num:       mov     al,num_lock_shift       ; NUM LOCK toggle lock
EAC2  E8 EAD8 R         C       call    k_4tog                          ; toggle 4 state
EAC5  B1 02             C       mov     cl,00000010b                    ; NUM LOCK LED is bit1.
EAC7  E8 EB6C R         C       call    k_LED_num               ; send LED information
EACA  EB F2             C       jmp     short k_non1
                        C
                        C
EACC  B0 10             C  k_scr:       mov     al,scrl_lock_shift      ; SCROLL LOCK toggle lock
EACE  E8 EAD8 R         C       call    k_4tog                          ; toggle 4 state
EAD1  B1 04             C       mov     cl,00000100b                    ; SCROLL LOCK LED is bit1.
EAD3  E8 EB6C R         C       call    k_LED_num               ; send LED information
EAD6  EB E6             C       jmp     short k_non1
                        C
                        C
EAD8  0A E4             C  k_4tog:      or      ah,ah                           ; is toggle shift 'make' ?
EADA  78 0F             C       js      k_4res                          ; if 'break' reset kb_flag_1
                        C
EADC  84 06 0018 R      C       test    byte ptr ds:[kb_flag_1],al ; if 'make', test bit.
EAE0  75 08             C       jnz     k_4ret                          ; return if already pressed
                        C
EAE2  08 06 0018 R      C       or      byte ptr ds:[kb_flag_1],al ; set the bit.
EAE6  30 06 0017 R      C       xor     byte ptr ds:[kb_flag],al   ; toggle kb_flag history.
EAEA  C3                C  k_4ret:      ret
                        C
EAEB  F6 D0             C  k_4res:      not     al
EAED  20 06 0018 R      C       and     byte ptr ds:[kb_flag_1],al ; if 'break', reset bit only.
EAF1  C3                C       ret
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Two state shifts. 'make' & 'break' in kb_flag only.
```

```
                        C  ;--------------------------------------------------------------------
                        C
EAF2  B0 08             C  k_alt:       mov     al,alt_shift                    ; ALT set/reset kb_flag
EAF4  E8 EB13 R         C       call    k_2tog                          ; toggle 2 state
                        C
EAF7  33 C0             C       xor     ax,ax                           ; scan code of ah = 0.
EAF9  86 06 0019 R      C       xchg    al,byte ptr ds:[alt_input] ; alt_input gets 0.
EAFD  0A C0             C       or      al,al                           ; was alt_input 0?
EAFF  74 BD             C       je      k_non1                          ; if so, do nothing.
EB01  E9 EA53 R         C       jmp     k_buf                           ; else, put it into buffer.
                        C
                        C
EB04  B0 04             C  k_ctl:       mov     al,cntrl_shift                  ; CTL set/reset kb_flag
EB06  EB 06             C       jmp     short k_2ret                    ; toggle 2 state and return
                        C
                        C
EB08  B0 02             C  k_lsh:       mov     al,left_shift                   ; LEFT SHIFT set/reset kb_flag
EB0A  EB 02             C       jmp     short k_2ret                    ; toggle 2 state and return
                        C
                        C
EB0C  B0 01             C  k_rsh:       mov     al,right_shift                  ; RIGHT SHIFT set/reset kb_flag
                        C  ;    jmp     short k_2ret                    ; fall through
                        C
                        C
EB0E                    C  k_2ret:
EB0E  E8 EB13 R         C       call    k_2tog                          ; toggle 2 state
EB11  EB AB             C       jmp     short k_non1
                        C
EB13  0A E4             C  k_2tog:      or      ah,ah                           ; is set/reset shift 'make' ?
EB15  78 05             C       js      k_2res                          ; if 'break', reset bit only.
                        C
EB17  08 06 0017 R      C       or      byte ptr ds:[kb_flag],al   ; if 'make', set bit only.
EB1B  C3                C       ret
                        C
EB1C  F6 D0             C  k_2res:      not     al
EB1E  20 06 0017 R      C       and     byte ptr ds:[kb_flag],al   ; if 'break', reset only.
EB22  C3                C       ret
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Alternate Numeric Keypad.  'make' only.
                        C  ;--------------------------------------------------------------------
                        C
EB23  41                C  k_alt9:      inc     cx                              ; Alternate Numeric Keypad9
EB24  41                C  k_alt8:      inc     cx                              ; Alternate Numeric Keypad8
EB25  41                C  k_alt7:      inc     cx                              ; Alternate Numeric Keypad7
EB26  41                C  k_alt6:      inc     cx                              ; Alternate Numeric Keypad6
EB27  41                C  k_alt5:      inc     cx                              ; Alternate Numeric Keypad5
EB28  41                C  k_alt4:      inc     cx                              ; Alternate Numeric Keypad4
EB29  41                C  k_alt3:      inc     cx                              ; Alternate Numeric Keypad3
EB2A  41                C  k_alt2:      inc     cx                              ; Alternate Numeric Keypad2
EB2B  41                C  k_alt1:      inc     cx                              ; Alternate Numeric Keypad1
EB2C                    C  k_alt0:                                              ; Alternate Numeric Keypad0
                        C
EB2C  B0 0A             C       mov     al,10
EB2E  F6 26 0019 R      C       mul     byte ptr ds:[alt_input]         ; alt_input = (10*alt_input)+cl
```

```
EB32  02 C1              C       add     al,cl
EB34  A2 0019 R          C       mov     byte ptr ds:[alt_input],al
EB37  EB 85              C       jmp     short k_non1
                         C
                         C  ;--------------------------------------------------------------------
                         C  ;    Double Zero on Keypad.  'make' only.
                         C  ;--------------------------------------------------------------------
                         C
EB39  B0 30              C  k_00:mov     al,'0'                          ; try to put ax into kb_buffer.
EB3B  E8 EBBC R          C       call    k_try                           ; zf set (z)   if buffer is full
EB3E  74 03              C       jz      k_nop1                          ; zf reset (nz) if buffer got ax
EB40  E9 EA53 R          C       jmp     k_buf                           ; put it into buffer, again.
                         C
EB43  E9 EA5C R          C  k_nop1:      jmp     k_nop
                         C
                         C  ;--------------------------------------------------------------------
                         C  ;    Break key sequence.  'make' only.
                         C  ;--------------------------------------------------------------------
                         C
EB46                     C  k_brk:
EB46  F6 06 0018 R 08    C       test    byte ptr ds:[kb_flag_1].pause_mode  ; are we in hold state?
EB4B  74 08              C       jz      k_do_brk                        ; if not, continue.
EB4D  80 26 0018 R F7    C       and     byte ptr ds:[kb_flag_1],not pause_mode      ; reset hold state bit &
EB52  E9 EA58 R          C       jmp     k_none                                      ; ignore the key
EB55                     C  k_do_brk:
EB55  BB 001E R          C       mov     bx,ds:(offset kb_buffer)
EB58  89 1E 001A R       C       mov     word ptr ds:[buffer_head],bx    ; reset buffer to empty
EB5C  89 1E 001C R       C       mov     word ptr ds:[buffer_tail],bx
EB60  C6 06 0071 R 80    C       mov     byte ptr ds:[bios_break],80h    ; turn on bios_break bit
EB65  CD 1B              C       INT     1Bh                             ; break interrupt vector
EB67  33 C0              C       xor     ax,ax                           ; ax gets deluxe 00h
EB69  E9 EA53 R          C       jmp     k_buf                           ; put ax into the buffer
                         C
                         C  k_intendp
                         C
                         C  ;--------------------------------------------------------------------
                         C  ;    Puts keyboard LED's in correct state after CAPS/NUM/SCROLL LOCK.
                         C  ;
                         C  ;    Input:  ah = scan code (make or break)
                         C  ;            al = kb_flag bit for CAPS/NUM/SCROLL LOCK (caps_lock_shift,
                         C  ;            num_lock_shift, scroll_lock_shift)
                         C  ;            cl = 00000001b for CAPS LOCK LED, 00000010b for NUM LOCK LED,
                         C  ;                 or 00000100 for SCROLL LOCK LED
                         C  ;    Output: None.
                         C  ;
                         C  ;    Trash:  al & cl destroyed.
                         C  ;--------------------------------------------------------------------
                         C
EB6C                     C  k_LED_num    proc    near
                         C               assume  cs:code, ds:data, es:nothing, ss:nothing
                         C
EB6C  F6 06 0018 R 01    C       test   byte ptr ds:[kb_flag_1],dlx_kb  ; are we a deluxe keyboard
EB71  74 03              C       jz      k_LED_cap                ; if   kb, LED is num_lock
EB73  80 F1 80           C       xor    cl,10000000b                  ; if deluxe kb, LED is
                         C                                              ;   ~num_lock
```

```
                        C
EB76                    C  k_LED_cap:
EB76  0A E4             C       or      ah,ah                           ; is CAPS/NUM LOCK 'make' ?
EB78  78 1D             C       js      k_LED_ret               ; if not, exit
                        C
EB7A  84 06 0017 R      C       test    byte ptr ds:[kb_flag],al ; is CAPS/NUM LOCK kb_flag set ?
EB7E  74 03             C       jz      k_LED_cmd               ; if not, LED data is ok.
EB80  80 F1 80          C       xor     cl,10000000b                    ; else, flip sense of LED data.
                        C
EB83                    C  k_LED_cmd:                                   ; polling loop to send command.
EB83  E4 64             C       in      al,kb_status                    ; get 8041 status
EB85  A8 02             C       test    al,00000010b                    ; test input buffer bit
EB87  75 FA             C       jnz     k_LED_cmd               ; if not ok to write cmd, loop.
                        C
EB89  B0 13             C       mov     al,013h                         ; keyboard 'LED' command.
EB8B  E6 60             C       out     p_kscan,al                      ; send keyboard 'LED' command.
                        C
EB8D                    C  k_LED_dat:                                   ; polling loop to send data.
EB8D  E4 64             C       in      al,kb_status                    ; get 8041 status
EB8F  A8 02             C       test    al,00000010b                    ; test input buffer bit
EB91  75 FA             C       jnz     k_LED_dat               ; if not ok to write data, loop.
                        C
EB93  8A C1             C       mov     al,cl                           ; retrieve keyboard 'LED' data.
EB95  E6 60             C       out     p_kscan,al                      ; send keyboard 'LED' data.
EB97                    C  k_LED_ret:
EB97  C3                C       ret
                        C
                        C  k_LED_num    endp
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Get kb_cap_flags bit into the carry flag (cf).
                        C  ;
                        C  ;    Input:  al = scan code (make)
                        C  ;            cl = 0
                        C  ;            es:di    = p_kscan base
                        C  ;    Output: cf set if kb_cap_flags bit
                        C  ;
                        C  ;    Trash:  si destroyed.
                        C  ;--------------------------------------------------------------------
                        C
EB98                    C  k_bitproc    near
                        C       assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
EB98  8B F3             C       mov     si,bx                           ; save bx
                        C
EB9A  33 DB             C       xor     bx,bx                           ; clear bh
EB9C  8A D8             C       mov     bl,al                           ; bx = scan code (make) index
                        C                                               ; bx = 00000000 00xxxyyy
                        C
EB9E  B1 03             C       mov     cl,3                            ; rotate right bx 3
EBA0  D3 CB             C       ror     bx,cl                           ; bx = yyy00000 00000xxx
                        C
EBA2  B1 03             C       mov     cl,3                            ; rotate left bh 3
EBA4  D2 C7             C       rol     bh,cl                           ; bx = 00000yyy 00000xxx
                        C                                               ; bh = remainder = (0-7)
```

```
                        C                                               ; bl = quotient  = (0-6)
                        C
EBA6  8A CF             C       mov     cl,bh                           ; cl = remainder = (0-7)
EBA8  32 FF             C       xor     bh,bh                           ; bx = quotient  = (0-6)
                        C
                        C
EBAA  26: 8A 59 F9      C       mov     bl,byte ptr es:[di+bx-7]    ; bl = proper cap_flags byte
                        C
EBAE  D2 D3             C       rcl     bl,cl                           ; rotate (0-7) times into bit7
EBB0  32 C9             C       xor     cl,cl                           ; cl = 0
EBB2  D0 D3             C       rcl     bl,1                            ; rotate into cf bit7
                        C
EBB4  8B DE             C       mov     bx,si                           ; restore bx
EBB6  C3                C       ret
                        C
                        C  k_bitendp
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Input:  None.
                        C  ;    Output: None.
                        C  ;
                        C  ;    Trash:  al & dx destroyed.
                        C  ;--------------------------------------------------------------------
                        C
EBB7                    C  k_eoiproc    near
                        C
                        C  ; Send specific end of interrupt (SEOI) to pic 'command' port.
                        C
EBB7  B0 61             C       mov     al,pic_seoi_1                   ; specific end of interrupt
EBB9  E6 20             C       out     pic_0,al
EBBB  C3                C       ret
                        C
                        C  k_eoiendp
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Try to put ax into the kb_buffer.
                        C  ;
                        C  ;    Input:  ax      = word to put in kb_buffer.
                        C  ;    Output: zf set  (z)  if buffer is full. (ax trashed)
                        C  ;            zf reset (nz) if buffer got ax.  (ax saved)
                        C  ;
                        C  ;    Trash:  bx, cx, dx, & si destroyed (in general).
                        C  ;--------------------------------------------------------------------
                        C
EBBC                    C  k_tryproc    near
                        C       assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
EBBC  FA                C       cli
EBBD  8B 1E 001C R      C       mov     bx,word ptr ds:[buffer_tail]    ; get buffer end pointer
EBC1  8B F3             C       mov     si,bx                           ; save the value
EBC3  E8 E870 R         C       call    k_adv_ptr               ; advance the tail
                        C
EBC6  3B 1E 001A R      C       cmp     bx,word ptr ds:[buffer_head]    ; has the buffer wrapped around
EBCA  74 08             C       je      k_try_beep
                        C                                               ; zf is reset (nz)
```

```
EBCC  89 04             C       mov     word ptr ds:[si],ax             ; store the value
EBCE  89 1E 001C R      C       mov     word ptr ds:[buffer_tail],bx    ; move the pointer up
EBD2  FB                C       sti
EBD3  C3                C       ret                                     ;return zf reset (nz)
EBD4                    C  k_try_beep:
EBD4  FB                C       sti
EBD5  EB 00             C       jmp     short k_beep                    ;k_beep will return zf set (zf)
                        C
                        C  k_tryendp
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Input:  None.
                        C  ;    Output: zf set  (z)  always.
                        C  ;
                        C  ;    Trash:  ax, bl, cx, & dx destroyed.
                        C  ;--------------------------------------------------------------------
                        C
EBD7                    C  k_beep       proc    near
                        C       assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
                        C  ; Mask keyboard interrupt
EBD7  FA                C       cli                                     ;stop interrupts
EBD8  E8 EBB7 R         C       call    k_eoi                           ; send specific end of interrupt
EBDB  F6 06 0018 R 04   C       test    byte ptr ds:[kb_flag_1],4  ; is beep already occurring?
EBE0  75 2B             C       jnz     skip_beep
EBE2  80 0E 0018 R 04   C       or      byte ptr ds:[kb_flag_1],4  ;set beep in progress bit.
EBE7  FB                C       sti                                     ;allow interrupts while beeping
                        C
                        C  ; interrupts enabled now (by k_eoi)
                        C
EBE8  BA 0061           C       mov     dx,p_kctrl                      ; get kb control port address
EBEB  EC                C       in      al,dx                           ; get control data
EBEC  8A E0             C       mov     ah,al                           ; save control data
                        C
EBEE  B3 80             C       mov     bl,80h                          ; outer loop counter
                        C
EBF0  24 FC             C  k_lp:and     al,0FCh                         ; turn off speaker data
EBF2  EE                C       out     dx,al
                        C
EBF3  B9 0048           C       mov     cx,48h                          ; set up count
EBF6  E2 FE             C       loop    $                               ; delay awhile
                        C
EBF8  0C 02             C       or      al,02h                          ; turn on speaker
EBFA  EE                C       out     dx,al
                        C
EBFB  B9 0048           C       mov     cx,48h                          ; set up count
EBFE  E2 FE             C       loop    $                               ; delay awhile
                        C
EC00  FE CB             C       dec     bl                              ; decrement outer loop counter
EC02  75 EC             C       jnz     k_lp
                        C                                               ; zf is set (z)
EC04  8A C4             C       mov     al,ah                           ; restore control data
EC06  EE                C       out     dx,al
                        C
                        C  ; Unmask keyboard interrupt
```

```
                              C
EC07  FA                      C       cli                                     ;disable interrupts
                              C                                               ;so we can clear stack
                              C                                               ;before next keypress.
EC08  80 26 0018 R FB         C       and     byte ptr ds:[kb_flag_1],0FBh    ;clear the beep in
                              C                                               ;progress bit.
EC0D                          C  skip_beep:
EC0D  0B C9                   C       or      cx,cx                           ; set zf again
EC0F  C3                      C       ret                                     ; return zf set (z)
                              C
                              C  k_beep             endp
                              C
                              C
EC10                          C  k_data1      proc
                              C
EC10  EAA6 R                  C  k_case       dw      k_ins           ; kbins  (0C0h)        ^
EC12  EAB4 R                  C       dw      k_cap           ; kbcap                 |
EC14  EAC0 R                  C       dw      k_num           ; kbnum                 |
EC16  EACC R                  C       dw      k_scr           ; kbscr                 |
EC18  EAF2 R                  C       dw      k_alt           ; kbalt          'make' & 'break'
EC1A  EB04 R                  C       dw      k_ctl           ; kbctl                 |
EC1C  EB08 R                  C       dw      k_lsh           ; kblsh                 |
EC1E  EB0C R                  C       dw      k_rsh           ; kbrsh                 v
                              C
EC20  EA65 R                  C       dw      k_res           ; kbres  (0C8h)         ^
EC22  EB46 R                  C       dw      k_brk           ; kbbrk                 |
EC24  EA76 R                  C       dw      k_pause         ; pause                 |
EC26  EA9A R                  C       dw      k_prt           ; kbprt                 |
EC28  EAA1 R                  C       dw      k_nul           ; kbnul                 |
EC2A  EA58 R                  C       dw      k_none          ; NONE                  |
                              C                               ;                       |
EC2C  EB23 R                  C       dw      k_alt9          ; kdec9                 |
EC2E  EB24 R                  C       dw      k_alt8          ; kdec8                 |
EC30  EB25 R                  C       dw      k_alt7          ; kdec7            'make' only
EC32  EB26 R                  C       dw      k_alt6          ; kdec6                 |
EC34  EB27 R                  C       dw      k_alt5          ; kdec5                 |
EC36  EB28 R                  C       dw      k_alt4          ; kdec4                 |
EC38  EB29 R                  C       dw      k_alt3          ; kdec3                 |
EC3A  EB2A R                  C       dw      k_alt2          ; kdec2                 |
EC3C  EB2B R                  C       dw      k_alt1          ; kdec1                 |
EC3E  EB2C R                  C       dw      k_alt0          ; kdec0                 |
                              C                               ;                       |
EC40  EB39 R                  C       dw      k_00            ; kdb10  (0D8h)         v
                              C
                              C  k_data1      endp
                              C
                              C  code ends


                              C  include fdu1.asm
                              C
                              C
                              C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; CODE ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                              C
EC42                          C  code segment  public   'ROM'
```

```
                        C       assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
                        C
                        C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                        C
                        C       XORG    0EC59h
EC59                    C1      org     0EC59h
                        C
EC59                    C  fd_ioproc    far
                        C
EC59  FB                C       sti                             ; enable interupts
EC5A  55                C       push    bp
EC5B  06                C       push    es
EC5C  1E                C       push    ds
EC5D  56                C       push    si
EC5E  57                C       push    di
EC5F  52                C       push    dx                      ; head & driv
EC60  51                C       push    cx                      ; cyl. & se
EC61  53                C       push    bx                      ; buffer offset
EC62  50                C       push    ax                      ; command &secs
EC63  52                C       push    dx                      ; this one gets modified(96 TPI)
EC64  8B EC             C       mov     bp,sp                   ; BP preserves SP throughout
                        C
                        C  ; test command code & use jump table to jump to appropriate routine
                        C
EC66  2E: 8E 1E E574 R  C       mov     ds,word ptr cs:[set_ds_word]    ; DS = data_seg (40h)
EC6B  80 26 003F R 0F   C       and     motor_status,0Fh        ; preserve motor on bits.
EC70  80 FC 00          C       cmp     ah,0                    ; Is it a reset command?
EC73  74 14             C       jz      diskette_io2            ; Yes, so ignore drive param.
EC75  80 FA 01          C       cmp     dl,1                    ; max. drives
EC78  77 05             C       ja      f_io1                   ; drive out of range.
EC7A  80 FC 05          C       cmp     ah,5                    ; max. command.
EC7D  76 07             C       jbe     diskette_io1            ; command in range
EC7F                    C  f_io1:
EC7F  C6 06 0041 R 01   C       mov     diskette_status,cmd_error  ; 01h
EC84  EB 1B             C       jmp     short f_io_ret          ; quick return
EC86                    C  diskette_io1:
EC86  E8 E805 R         C       call    f_check_valid           ; Will not return if error.
EC89                    C  diskette_io2:
EC89  FC                C       cld                             ; Autoincrement for strings.
EC8A  32 FF             C       xor     bh,bh                   ; clear high byte
EC8C  8A DC             C       mov     bl,ah                   ; move selection into low byte
EC8E  D1 E3             C       shl     bx,1                    ; multiply by 2
EC90  2E: FF A7 EC95 R  C       jmp     cs:[f_table,bx]
                        C
                        C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                        C
EC95                    C  f_table      label   word
                        C
EC95  ECC6 R            C       dw      f_reset                 ; AH = 0 (reset)
EC97  ECA1 R            C       dw      f_io_ret                ; AH = 1 (status)
EC99  ED4E R            C       dw      f_rdata                 ; AH = 2 (read)
EC9B  ED22 R            C       dw      f_wdata                 ; AH = 3 (write)
EC9D  ED4E R            C       dw      f_rdata                 ; AH = 4 (verify)
EC9F  ED22 R            C       dw      f_wdata                 ; AH = 5 (format)
```

```
                  C
                  C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                  C
ECA1              C  f_io_ret:
                  C  ; f_nec_reset is located in the module called fdu4.src.
ECA1  E8 E7C9 R   C       call     f_nec_reset                          ; reset if necessary.
ECA4  BB 0002     C       mov      bx,2                                 ; motor_wait parameter.
ECA7  E8 EDEF R   C       call     f_get_var                  ; Returns 0 in AH.
ECAA  A2 0040 R   C       mov      motor_count,al                       ; motor shut off value.
                  C
ECAD  32 C0       C       xor      al,al                                ; zero AL.
ECAF  8A 26 0041 R C      mov      ah,diskette_status
                  C
ECB3  80 FC 01    C       cmp      ah,1
ECB6  F5          C       cmc
                  C
ECB7  8B E5       C       mov      sp,bp                                ; for safety sake.
ECB9  5B          C       pop      bx                                   ; discard DX.
ECBA  5B          C       pop      bx                                   ; discard AX.
ECBB  5B          C       pop      bx
ECBC  59          C       pop      cx
ECBD  5A          C       pop      dx
ECBE  5F          C       pop      di
ECBF  5E          C       pop      si
ECC0  1F          C       pop      ds
ECC1  07          C       pop      es
ECC2  5D          C       pop      bp
                  C
ECC3  CA 0002     C       ret      2
                  C
                  C  fd_ioendp
                  C
                  C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                  C  ;
                  C  ;       Reset and reprogram the FDC without turning the motors off.
                  C  ;
                  C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                  C
ECC6              C  f_reset        proc     near
                  C
ECC6  FA          C       cli                                           ; disable interrupts
                  C
ECC7  B0 66       C       mov      al,pic_seoi_6                        ; specific end of interrupt
ECC9  BA 0020     C       mov      dx,pic_0                   ; to pic 'command' port.
ECCC  EE          C       out      dx,al
                  C
ECCD  42          C       inc      dx                                   ; pic 'data' port.
ECCE  EC          C       in       al,dx                                ; read mask.
ECCF  24 BF       C       and      al,10111111b                         ; enable IR6.
ECD1  EE          C       out      dx,al
                  C
                  C  ; Develop mask for motor control port.
                  C
ECD2  A0 003F R   C       mov      al,motor_status                 ; which motor is running?
ECD5  24 0F       C       and      al,0Fh                          ; blow off high 4 bits.
```

```
ECD7  74 0E              C       jz      f_r2                    ; no motors running.
ECD9  8A E0              C       mov     ah,al
ECDB  B1 04              C       mov     cl,4
ECDD  D2 E0              C       shl     al,cl                   ; move to high nib.
                         C
                         C  ; A motor is on.  Find out which one.
                         C
ECDF                     C  f_r1:
ECDF  D0 EC              C       shr     ah,1                    ; determine drive select
ECE1  72 04              C       jc      f_r2                    ; from motor enable bit.
ECE3  FE C0              C       inc     al
ECE5  EB F8              C       jmp     short f_r1
ECE7                     C  f_r2:
                         C
                         C  ; Reset signal has to be maintained for at least 14 clocks.
                         C
ECE7  C6 06 0041 R 00    C       mov     diskette_status,0
ECEC  0C 08              C       or      al,8                    ; set bit 3.
ECEE  BA 03F2            C       mov     dx,f_motor_port
ECF1  EE                 C       out     dx,al                   ; send reset signal.
ECF2  0C 04              C       or      al,4                    ; set bit 2.
ECF4  C6 06 003E R 00    C       mov     seek_status,0
ECF9  EE                 C       out     dx,al                   ; clear reset.
ECFA  FB                 C       sti
ECFB  E8 EF82 R          C       call    f_sis                   ; sense int. status
                         C
ECFE  B4 03              C       mov     ah,f_specify_cmd
ED00  E8 EDFC R          C       call    f_put_byte
ED03  BB 0000            C       mov     bx,0                    ; 1st specify byte.
ED06  E8 EDEF R          C       call    f_get_var
ED09  8A E0              C       mov     ah,al
ED0B  E8 EDFC R          C       call    f_put_byte
ED0E  BB 0001            C       mov     bx,1                    ; 2nd specify byte.
ED11  E8 EDEF R          C       call    f_get_var
ED14  8A E0              C       mov     ah,al
ED16  E8 EDFC R          C       call    f_put_byte
ED19  EB 86              C       jmp     f_io_ret
                         C
                         C  f_reset      endp
                         C
                         C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         C  ;
                         C  ; This routine is called by the timer_int routine when motor_count = 0
                         C  ;
                         C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         C
ED1B                     C  stop_disk    proc     near
                         C
ED1B  B0 0C              C       mov     al,0Ch
ED1D  BA 03F2            C       mov     dx,f_motor_port
ED20  EE                 C       out     dx,al
ED21  C3                 C       ret
                         C  stop_disk    endp
                         C
                         C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
                                C
ED22                            C  f_wdata      proc     near
                                C
                                C
ED22  E8 E7E0 R                 C      call     f_motor_on                 ; sets carry if motor was off.
ED25  73 1E                     C      jnc      f_wd1                      ; motor was on, skip delay.
                                C
ED27  E4 67                     C      in       al,sys_conf_b              ; read switch.
ED29  F6 D0                     C      not      al                         ; toggle the sense.
ED2B  24 02                     C      and      al,2                       ; isolate slow motor bit.
ED2D  8A C8                     C      mov      cl,al
ED2F  BA 007D                   C      mov      dx,125                     ; 125 ms delay to start with.
ED32  D3 E2                     C      shl      dx,cl                      ; 125 x 4
ED34  BB 000A                   C      mov      bx,10                      ; motor start delay parameter.
ED37  E8 EDEF R                 C      call     f_get_var          ; returns param. in AL.
ED3A  32 E4                     C      xor     ah,ah                   ; for good measure.
ED3C  F7 E2                     C      mul      dx                         ; AX has total delay
ED3E  8B C8                     C      mov      cx,ax
                                C
ED40                            C  f_wd_loop:
ED40  E8 EF37 R                 C      call     f_wait_one_ms
ED43  E2 FB                     C      loop     f_wd_loop
                                C
ED45                            C  f_wd1:
ED45  80 0E 003F R 80           C      or       motor_status,080h  ; set high bit, indicate write.
ED4A  B0 4A                     C      mov      al,04Ah                    ; DMA mode byte: channel 2,
                                C                                          ; single mode, read transfer
ED4C  EB 29                     C      jmp      short f_rw_common
                                C  f_wdata      endp
                                C
                                C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                                C
ED4E                            C  f_rdata      proc     near
                                C
                                C
ED4E  80 26 003F R 7F           C      and      motor_status,07Fh  ; clear high bit, indicate read
ED53  E8 E7E0 R                 C      call     f_motor_on
ED56  73 11                     C      jnc      f_rd1                      ; motor was on, no delay.
                                C
                                C  ; slow motor delay loop
                                C
ED58  E4 67                     C      in       al,sys_conf_b              ; read configuration switch(7W)
ED5A  A8 02                     C      test     al,2                       ; bit 1 = slow drive bit
ED5C  B9 00FA                   C      mov      cx,250                     ;
ED5F  75 03                     C      jnz      f_rd_loop          ; 1 = 500 ms motors.
ED61  B9 01F4                   C      mov      cx,500                     ; approx. 500 ms delay for
                                C
ED64                            C  f_rd_loop:
ED64  E8 EF37 R                 C      call     f_wait_one_ms
ED67  E2 FB                     C      loop     f_rd_loop
                                C
ED69                            C  f_rd1:
ED69  B0 46                     C      mov      al,046h                    ; DMA mode byte: channel 2,
                                C                                          ; single mode, write transfer.
ED6B  80 7E 03 04               C      cmp      byte ptr f_command,4       ; Is it a verify command?
```

```
ED6F  75 06            C       jne     f_rw_common              ; No, must have been a read.
ED71  B0 42            C       mov     al,042h                  ; DMA mode byte: channel 2,
                       C                                        ; single mode, verify transfer.
ED73  33 DB            C       xor     bx,bx                    ; Fool the DMAC into thinking
ED75  8E C3            C       mov     es,bx                    ; there is a full segment to
                       C                                        ; play with.
                       C  f_rdata      endp
                       C
                       C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                       C  ;
                       C  ;       Common  (f_rw_common)
                       C  ;
                       C  ;    INPUT:           AL       dma mode byte.
                       C  ;
                       C  ;    OUTPUT:
                       C  ;
                       C  ;    DESTROYS:
                       C  ;
                       C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                       C
ED77                   C  f_rw_common  proc     near
                       C
                       C
ED77  C6 06 0040 R FF  C       mov     motor_count,0FFh        ; long wait.
ED7C  E8 EE63 R        C       call    f_set_dma               ; pass mode byte on through.
                       C
                       C  ; Clear out status from previous operation.
                       C
ED7F  06               C       push    es
ED80  1E               C       push    ds
ED81  07               C       pop     es
ED82  32 C0            C       xor     al,al
ED84  B9 0007          C       mov     cx,7
ED87  BF 0042 R        C       mov     di,offset nec_status
ED8A  F3/ AA           C       rep     stosb
ED8C  07               C       pop     es
                       C
ED8D  E8 EEBD R        C       call    f_seek                   ; On return, f_drive has
                       C                                        ; head bit or'd in.
                       C
ED90  8A 56 03         C       mov     dl,byte ptr f_command    ; get command
ED93  B4 C5            C       mov     ah,f_write_cmd
ED95  80 FA 03         C       cmp     dl,3                     ; Is it a write command?
ED98  74 09            C       je      f_rw1                    ; yes
ED9A  B4 4D            C       mov     ah,f_format_cmd
ED9C  80 FA 05         C       cmp     dl,5                     ; Is it a format command?
ED9F  74 02            C       je      f_rw1                    ; yes
EDA1  B4 E6            C       mov     ah,f_read_cmd            ; must be read or verify.
EDA3                   C  f_rw1:
EDA3  E8 EDFC R        C       call    f_put_byte               ; send command.
EDA6  8A 66 00         C       mov     ah,f_drive               ; has head and drive bits.
EDA9  E8 EDFC R        C       call    f_put_byte
EDAC  80 7E 03 05      C       cmp     byte ptr f_command,5     ; was it a format command?
EDB0  74 15            C       je      f_rw_skip               ; yes, skip next 3 params.
EDB2  8A 66 07         C       mov     ah,f_cyl
```

```
EDB5  E8 EDFC R          C        call     f_put_byte
EDB8  8A 66 01           C        mov      ah,f_head
EDBB  80 E4 7F           C        and     ah,07Fh                         ; blow off bit 7.
EDBE  E8 EDFC R          C        call     f_put_byte
EDC1  8A 66 06           C        mov      ah,f_secnum
EDC4  E8 EDFC R          C        call     f_put_byte
                         C
                         C  ; Get bytes 3,4,5,6 from table.
                         C  ; If we are formatting then we need bytes 3,4,7,8 from table.
                         C
EDC7                     C  f_rw_skip:
EDC7  B9 0004            C        mov      cx,4
EDCA  BB 0003            C        mov      bx,3                           ; no. bytes per sector.
EDCD                     C  f_rw2:
EDCD  E8 EDEF R          C        call     f_get_var
EDD0  8A E0              C        mov      ah,al
EDD2  E8 EDFC R          C        call     f_put_byte
EDD5  43                 C        inc      bx
EDD6  83 FB 05           C        cmp      bx,5                           ; time to check for format?
EDD9  75 09              C        jne      f_rw3                          ; No.
EDDB  80 7E 03 05        C        cmp      byte ptr f_command,5           ; was it a format command?
EDDF  75 03              C        jne      f_rw3                          ; no.
EDE1  BB 0007            C        mov      bx,7                           ; 7th parameter in table.
EDE4                     C  f_rw3:
EDE4  E2 E7              C        loop     f_rw2
                         C
EDE6  E8 EF8D R          C        call     f_wait_for_nec
EDE9  E8 EE19 R          C        call     f_get_byte                     ; get the results.
                         C
EDEC                     C  f_rw_ret:
EDEC  E9 ECA1 R          C        jmp      f_io_ret
                         C
                         C  f_rw_common     endp
                         C
                         C  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
                         C  ;
                         C  ;       Get specified byte from fdu parameter table  (f_get_var)
                         C  ;
                         C  ;    INPUT:            BX       parameter number (0 - 10)
                         C  ;
                         C  ;    OUTPUT:           AL       The requested byte.
                         C  ;
                         C  ;    DESTROYS:AH
                         C  ;
                         C  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
                         C
EDEF                     C  f_get_var     proc      near
                         C
EDEF  1E                 C        push     ds
EDF0  33 C0              C        xor      ax,ax
EDF2  8E D8              C        mov      ds,ax                          ; segment 0
                         C
                         C        assume   ds:abs0                        ; tell assembler seg 0:
                         C
EDF4  C5 36 0078 R       C        lds      si,dword ptr [int1Elocn]   ; DS:SI points to table
```

```
EDF8  8A 00              C        mov     al,[bx+si]
EDFA  1F                 C        pop     ds
                         C
                         C        assume  ds:data                         ; tell assembler seg 40:
EDFB  C3                 C        ret
                         C
                         C  f_get_var     endp
                         C
                         C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         C  ;
                         C  ;        Send a byte to the NEC controller  (f_put_byte)
                         C  ;
                         C  ;     INPUT:           AH       byte to output.
                         C  ;
                         C  ;     OUTPUT:
                         C  ;
                         C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         C
EDFC                     C  f_put_byte    proc     near
                         C
EDFC  E8 EF6B R          C        call    f_nec_rdy               ; returns MSR byte in AL.
EDFF  A8 40              C        test    al,40h                          ; direction bit.
EE01  75 05              C        jnz     f_pb_erret                      ; wrong direction.
EE03  42                 C        inc     dx                              ; NEC data port 3F5h
EE04  8A C4              C        mov     al,ah
EE06  EE                 C        out     dx,al
                         C
EE07                     C  f_pb_ret:
EE07  C3                 C        ret
                         C
EE08                     C  f_pb_erret:
EE08  C6 06 0041 R 20    C        mov     diskette_status,fdc_error
EE0D  E9 ECA1 R          C        jmp     f_io_ret
                         C
                         C  f_put_byte    endp
                         C
                         C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         C  ;
                         C  ;        Read the results bytes from the NEC controller (fdu_data1)
                         C  ;
                         C  ;     INPUT:           none
                         C  ;
                         C  ;     OUTPUT:
                         C  ;
                         C  ;
                         C  ;     DESTROYS:DX, SI
                         C  ;
                         C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         C
EE10                     C  fdu_data1     proc
                         C
EE10                     C  f_gb_table    label    byte
EE10  04                 C        db      4                               ; record not found.
EE11  00                 C        db      0                               ; dummy
EE12  10                 C        db      10h                             ; crc error.
```

```
EE13  08                  C       db      8                               ; dma error.
EE14  00                  C       db      0                               ; dummy.
EE15  04                  C       db      4                               ; sector not found.
EE16  03                  C       db      3                               ; write protect.
EE17  02                  C       db      2                               ; address mark error.
EE18  20                  C       db      20h                             ; fdc error.
                          C
                          C  fdu_data1    endp
                          C
EE19                      C  f_get_byte   proc    near
                          C
EE19  06                  C       push    es
EE1A  1E                  C       push    ds
EE1B  07                  C       pop     es
EE1C  BF 0042 R           C       mov     di,offset nec_status            ; ES:DI is now ready for stosb.
EE1F  8B F7               C       mov     si,di                           ; offset of nec_status in SI.
EE21  B9 0007             C       mov     cx,7                            ; maximum no. of bytes.
EE24                      C  f_gb_loop:
EE24  E8 EF6B R           C       call    f_nec_rdy               ; Returns MSR byte in AL.
                          C                                               ; will not return if error.
EE27  A8 10               C       test    al,10h                          ; busy bit.
EE29  74 36               C       jz      f_gb_ret                ; done.
EE2B  A8 40               C       test    al,40h                          ; direction bit.
EE2D  74 05               C       jz      f_gb_out                ; wrong direction.
EE2F  42                  C       inc     dx                              ; point to nec data port (3F5h)
EE30  EC                  C       in      al,dx
EE31  AA                  C       stosb
EE32  E2 F0               C       loop    f_gb_loop
EE34                      C  f_gb_out:
EE34  80 7E 03 00         C       cmp     byte ptr f_command,0            ; was it reset command?
EE38  74 27               C       je      f_gb_ret                ; yes.
EE3A  AC                  C       lodsb                                   ; get ST0 in AL.
EE3B  A8 20               C       test    al,20h                          ; seek end?
EE3D  75 22               C       jnz     f_gb_ret                ; yes.
EE3F  A8 C0               C       test    al,0C0h
EE41  74 1A               C       jz      f_gb_jmp
                          C
EE43  C6 06 0041 R 20     C       mov     diskette_status,fdc_error
EE48  AC                  C       lodsb                                   ; get ST1 in AL.
EE49  B9 0008             C       mov     cx,8
EE4C  33 DB               C       xor     bx,bx
EE4E                      C  f_gb_loop1:
EE4E  D0 C0               C       rol     al,1
EE50  72 03               C       jc      f_gb_decode
EE52  43                  C       inc     bx
EE53  E2 F9               C       loop    f_gb_loop1
EE55                      C  f_gb_decode:
EE55  2E: 8A 87 EE10 R    C       mov     al,cs:f_gb_table[bx]
EE5A  A2 0041 R           C       mov     diskette_status,al
EE5D                      C  f_gb_jmp:
EE5D  07                  C       pop     es
EE5E  E9 ECA1 R           C       jmp     f_io_ret
                          C
EE61                      C  f_gb_ret:
EE61  07                  C       pop     es
```

```
EE62  C3               C       ret
                       C
                       C  f_get_byte    endp
                       C
                       C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                       C  ;
                       C  ;        Test for boundary crossing case then program the
                       C  ;        DMA controller (f_set_dma).
                       C  ;
                       C  ;     INPUT:          AL       mode byte
                       C  ;                     ES       segment of user buffer.
                       C  ;
                       C  ;     OUTPUT:
                       C  ;
                       C  ;     DESTROYS:AX, BX, CX, DX
                       C  ;
                       C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                       C
EE63                   C  f_set_dma     proc      near
                       C
EE63  C6 06 0041 R 09  C       mov      diskette_status,dma_seg_error
EE68  E6 0C            C       out      dma_ff_clr,al                  ; set to known state.
EE6A  E6 0B            C       out      dma_mode,al                    ; send mode byte.
EE6C  8B 5E 04         C       mov      bx,word ptr f_bufoff           ; offset of user buffer.
EE6F  8C C0            C       mov      ax,es                          ; offset of user buffer.
EE71  B1 04            C       mov      cl,4                           ; shift count
EE73  D3 C0            C       rol      ax,cl                          ; move high nib around.
EE75  8A E8            C       mov      ch,al                          ; save high nib.
EE77  80 E5 0F         C       and      ch,0Fh                         ; isolate high nib
EE7A  24 F0            C       and      al,0F0h                        ; prepare for offset calculation
EE7C  03 D8            C       add      bx,ax                          ; 20 bit calculation.
                       C       jnc      f_sd1
EE80  FE C5            C       inc      ch                             ; high nib
EE82                   C  f_sd1:
EE82  8A C5            C       mov      al,ch
EE84  E6 81            C       out      dma_segm_2,al                  ; high nib. to latch
EE86  B0 06            C       mov      al,6
EE88  E6 0A            C       out      dma_mask_bit,al                ; disable channel 2.
EE8A  8A C3            C       mov      al,bl
EE8C  E6 04            C       out      dma_addr_2,al                  ; low byte of address.
EE8E  8A C7            C       mov      al,bh
EE90  E6 04            C       out      dma_addr_2,al                  ; high byte of address.
EE92  8B D3            C       mov      dx,bx                          ; save buffer offset.
EE94  BB 0003          C       mov      bx,3                           ; bytes/sector param
EE97  E8 EDEF R        C       call     f_get_var                 ; get param from table
EE9A  8A C8            C       mov      cl,al                          ; save for shift count
EE9C  8A 66 02         C       mov      ah,f_numsecs                   ; get #sectors.
EE9F  32 C0            C       xor      al,al                          ; AX = #sectors x 256
EEA1  D1 E8            C       shr      ax,1                           ; AX = #sectors x 128
EEA3  D3 E0            C       shl      ax,cl                          ; multiply by bytes/sector.
EEA5  48               C       dec      ax                             ; DMAC counts zero.
EEA6  03 D0            C       add      dx,ax                          ; add count to offset
EEA8  73 03            C       jnc      f_sd2
EEAA  E9 ECA1 R        C       jmp     f_io_ret                   ; exit boundary crossing.
EEAD                   C  f_sd2:
```

```
EEAD  E6 05               C        out     dma_count_2,al               ; low byte of count.
EEAF  8A C4               C        mov     al,ah
EEB1  E6 05               C        out     dma_count_2,al               ; high byte of count.
EEB3  B0 02               C        mov     al,2
EEB5  E6 0A               C        out     dma_mask_bit,al              ; enable channel 2
EEB7  C6 06 0041 R 00     C        mov     diskette_status,0
EEBC                      C  f_sd_ret:
EEBC  C3                  C        ret
                          C  f_set_dma    endp
                          C
                          C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                          C  ;
                          C  ;      Seek  (f_seek)
                          C  ;
                          C  ;   INPUT:
                          C  ;
                          C  ;   OUTPUT:
                          C  ;
                          C  ;   DESTROYS:
                          C  ;
                          C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                          C
EEBD                      C  f_seek        proc     near
                          C
EEBD  8A 4E 00            C        mov     cl,f_drive                   ; get drive# as shift count.
EEC0  B0 01               C        mov     al,1
EEC2  D2 E0               C        shl    al,cl                    ; mask for recal.
EEC4  84 06 003E R        C        test    seek_status,al
EEC8  75 28               C        jnz     f_s1                         ; no recal. required.
EECA  08 06 003E R        C        or      seek_status,al               ; set the corresponding bit.
                          C
                          C  ; Two recalibrate commands required in the case of 96 TPI drives.
                          C
soft (R) Macro Assembler  Version 4.00              4/3/86 16:59:25

   1-182

EECE  B9 0002             C        mov     cx,2                         ; loop count for 96 TPI
EED1                      C  f_s_recal:
EED1  B4 07               C        mov     ah,f_recal_cmd               ; recal. command
EED3  51                  C        push    cx                           ; save it.
EED4  E8 EDFC R           C        call    f_put_byte
EED7  8A 66 00            C        mov     ah,f_drive                   ; drive bit.
EEDA  E8 EDFC R           C        call    f_put_byte                   ; end of command phase.
EEDD  E8 EF82 R           C        call    f_sis                        ; Sense Interrupt Status
EEE0  59                  C        pop     cx                           ; restore it.
                          C
                          C  ; possibly test bit 4 as well (equipment check...track 0 not reached)
EEE1  F6 06 0042 R C0     C        test    nec_status,0C0h
EEE6  74 0A               C        jz     f_s1                     ; equipment OK so restore.
EEE8  E2 E7               C        loop    f_s_recal
EEEA  C6 06 0041 R 40     C        mov     diskette_status,seek_error
EEEF  E9 ECA1 R           C        jmp     f_io_ret
                          C
EEF2                      C  f_s1:
```

```
EEF2  B4 0F              C        mov     ah,f_seek_cmd
EEF4  E8 EDFC R          C        call    f_put_byte
                         C  ; prepare second byte of seek command.
EEF7  8A 46 01           C        mov    al,f_head
EEFA  24 01              C        and     al,1                       ; blow off high 7 bits.
EEFC  D0 E0              C        shl     al,1                       ; move head to bit 2.
EEFE  D0 E0              C        shl     al,1
EF00  0A 46 00           C        or      al,f_drive                 ; combine drive bit with head
EF03  88 46 00           C        mov     f_drive,al                 ; and save it.
EF06  8A E0              C        mov     ah,al
EF08  E8 EDFC R          C        call    f_put_byte
EF0B  8A 66 07           C        mov     ah,f_cyl
EF0E  F6 46 01 80        C        test    byte ptr f_head,80h        ; test 80 track bit.
EF12  74 02              C        jz      f_cont
EF14  D0 E4              C        shl     ah,1                       ; cyl x 2
EF16                     C  f_cont:
EF16  E8 EDFC R          C        call    f_put_byte                 ; end of seek command.
EF19  E8 EF82 R          C        call    f_sis                      ; Sense Interrupt Status
                         C
EF1C  F6 06 0042 R C0    C        test    nec_status,0C0h            ; test for seek end.
                         C
EF21  75 13              C        jnz     f_s_ret
EF23  BB 0009            C        mov     bx,9                       ; head settle parameter.
EF26  E8 EDEF R          C        call    f_get_var
EF29  0A C0              C        or      al,al
EF2B  74 09              C        jz      f_s_ret                    ; head settle = 0.
EF2D  8A C8              C        mov     cl,al                      ; use settle parm. al loop index
EF2F  32 ED              C        xor     ch,ch
EF31                     C  f_head_settle:
EF31  E8 EF37 R          C        call    f_wait_one_ms
EF34  E2 FB              C        loop    f_head_settle
EF36                     C  f_s_ret:
EF36  C3                 C        ret
                         C
                         C  f_seek          endp
                         C
                         C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         C  ;
                         C  ;     One millisecond delay loop (approximately)
                         C  ;
                         C  ;               The CALL is 19 clocks.
                         C  ;               The callers LOOP statement is 17 clocks.
                         C  ;               No flags affected.
                         C  ;
                         C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         C
EF37                     C  f_wait_one_ms proc     near
EF37  51                 C        push    cx                         ; 11 clocks
EF38  B9 0176            C        mov     cx,374                     ; 4 clocks
EF3B  E2 FE              C  w_one:        loop      w_one                    ; (17 x CX) + 5 clocks
EF3D  59                 C        pop     cx                         ; 8 clocks
EF3E  C3                 C        ret                                ; 16 clocks
                         C  f_wait_one_ms endp
                         C
```

```
                          C  include fdu2.asm
                          C
                          C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                          C  ;
                          C  ;
                          C  ;    INPUT:          none
                          C  ;
                          C  ;    OUTPUT:         MSB of seek_status is set if NEC interrupts.
                          C  ;
                          C  ;    DESTROYS:nothing
                          C  ;
                          C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                          C
EF57                      C             ORG     0EF57h
                          C
EF57                      C  fd_int     proc    near
                          C
                          C
EF57  FB                  C       sti                                   ; enable interupts
EF58  50                  C       push      ax
EF59  1E                  C       push      ds
EF5A  B0 66               C       mov       al,pic_seoi_6               ; specific end of interupt
EF5C  E6 20               C       out       pic_0,al               ; send to 8259
EF5E  2E: 8E 1E E574 R    C       mov       ds,word ptr cs:[set_ds_word]  ; set DS to segment 40h
EF63  80 0E 003E R 80     C       or        seek_status,80h             ; interupt indicator
EF68  1F                  C       pop       ds
EF69  58                  C       pop       ax
EF6A  CF                  C       iret
                          C  fd_int     endp
                          C
                          C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                          C  ;
                          C  ;       NEC ready (f_nec_rdy)
                          C  ;
                          C  ;    INPUT:          none
                          C  ;
                          C  ;    OUTPUT:         AL              Main Status Register byte.
                          C  ;                    DX              Points to port 3F4h
                          C  ;
                          C  ;    DESTROYS:
                          C  ;
                          C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                          C
EF6B                      C  f_nec_rdy  proc    near
                          C
EF6B  51                  C       push      cx
EF6C  33 C9               C       xor       cx,cx
EF6E                      C  f_nr1:
EF6E  BA 03F4             C       mov       dx,f_nec_status             ; Main status register(3F4h)
EF71  EC                  C       in        al,dx
EF72  A8 80               C       test     al,080h                 ; RQM
EF74  75 0A               C       jnz       f_nr_ret               ; NEC is ready.
EF76  E2 F6               C       loop      f_nr1
                          C
EF78  C6 06 0041 R 80     C       mov       diskette_status,time_out
```

```
EF7D  E9 ECA1 R          C       jmp     f_io_ret                    ; Took too long to respond.
EF80                     C  f_nr_ret:
EF80  59                 C       pop     cx
EF81  C3                 C       ret
                         C
                         C  f_nec_rdy     endp
                         C
                         C  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
                         C  :
                         C  :       Sense Interrupt Status  (f_sis).
                         C  :
                         C  :    INPUT:            none
                         C  :
                         C  :    OUTPUT:
                         C  :
                         C  :    DESTROYS:
                         C  :
                         C  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
                         C
EF82                     C  f_sisproc     near
                         C
EF82  E8 EF8D R          C       call    f_wait_for_nec              ; no return on error.
EF85  B4 08              C       mov     ah.f_snsint_cmd             ; end of command phase.
EF87  E8 EDFC R          C       call    f_put_byte                  ; no return on error.
EF8A  E9 EE19 R          C       jmp     f_get_byte
                         C
                         C  f_sisendp
                         C
                         C  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
                         C  :
                         C  :    Wait for NEC to interupt on completion of execution phase.
                         C  :       or time out if NEC never interupts (f_wait_for_nec).
                         C  :
                         C  :    INPUTS:           none
                         C  :
                         C  :    OUTPUTS:
                         C  :
                         C  ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
                         C
EF8D                     C  f_wait_for_necproc     near
                         C
EF8D  FB                 C       sti                                 ; enable interupts
EF8E  51                 C       push    cx
EF8F  B9 03E8            C       mov     cx.1000                     ; wait a while
                         C
EF92                     C  w_nec:
EF92  F6 06 003E R 80    C       test    seek_status.80h             ; test MSB (int_flag)
EF97  75 0D              C       jnz     w_nec_ret
EF99  E8 EF37 R          C       call    f_wait_one_ms
EF9C  E2 F4              C       loop    w_nec
                         C
EF9E  C6 06 0041 R 80    C       mov     diskette_status.time_out
EFA3  E9 ECA1 R          C       jmp     f_io_ret
EFA6                     C  w_nec_ret:
EFA6  80 26 003E R 7F    C       and     seek_status.07Fh        ; clear MSB.
```

```
EFAB  59                C       pop     cx
EFAC  C3                C       ret
                        C
                        C  f_wait_for_necendp
                        C

                        C  include fdu3.asm
                        C
                        C
                        C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                        C  ; fd_parms
                        C  ;    This is the set of parameters required for diskette operation.
                        C  ;    They are pointed to by interrupt vector 1Eh (0:78h). To modify
                        C  ;    the parameters, build another parameter block and point disk_pointer
                        C  ;    to it.
                        C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                        C
                        C       XORG    0EFC7h
EFC7                    C1      org     0EFC7h
                        C
EFC7                    C  fdu_data2    proc
                        C
EFC7                    C  fd_parms     label   byte
                        C
EFC7  CF                C       db      (f_srt_48 shl 4) + f_hut   ; SRT + HUT
EFC8  02                C       db      (f_hlt shl 1) + f_ndma            ; HLT + DMA mode
EFC9  25                C       db      f_motor_wait                      ; motor off delay
EFCA  02                C       db      2                                 ; 512 bytes per sector
EFCB  08                C       db      8                                 ; EOT last sector on cyl
EFCC  2A                C       db      02Ah                              ; gap length
EFCD  FF                C       db      0FFh                              ; DTL
EFCE  50                C       db      050h                              ; gap length for format
EFCF  F6                C       db      0F6h                              ; fill byte for format
EFD0  19                C       db      25                                ; head settle time (ms)
EFD1  04                C       db      4                                 ; motor start time
                        C
                        C  fdu_data2    endp
                        C
EFD2                    C  code ends

                        C  include prt.asm
                        C
                        C  ;======================================================================
                        C  ;    Filename:prt.src
                        C  ;
                        C  ;    This module includes INT 17h.
                        C  ;
                        C  ;======================================================================
                        C
EFD2                    C  code segment  public 'ROM'
                        C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C  ;======================================================================
                        C  ;    INT 17h  -- Printer Software Interrupt Request Routine
                        C  ;
```

```
                        C  ;    Input:   ah = 0   print character
                        C  ;             al =     character to print
                        C  ;             dx =     printer port number (0,1,2,3)
                        C  ;    Output:  ah =     status of printer: bit #0 set if time out
                        C  ;             al =     character to print (preserved)
                        C  ;
                        C  ;    Input:   ah = 1   initialize the printer port
                        C  ;             dx =     printer port number (0,1,2,3)
                        C  ;    Output:  ah =     status of printer
                        C  ;             al       preserved
                        C  ;
                        C  ;    Input:   ah = 2   read the printer status
                        C  ;             dx =     printer port number (0,1,2,3)
                        C  ;    Output:  ah =     status of printer
                        C  ;
                        C  ;    Trash:   ah =     (ah - 2) if ah > 2
                        C  ;             al       preserved
                        C  ;
                        C  ;    Assumes: prt_stat_x = prt_data_x + 1 = dx + 1
                        C  ;                     prt_cmd_x  = prt_data_x + 2 = dx + 2
                        C  ;
                        C  ; Printer Status Byte from prt_stat_x:
                        C  ;
                        C  ;    bit      #0 =     time-out on printing character (p_out).
                        C  ;                      set by software; no hardware significance.
                        C  ;    bits  #1-2 =      no significance (always cleared).
                        C  ;    bit      #3 =     hardware: I/O error.
                        C  ;    bit      #4 =     hardware: selected.
                        C  ;    bit      #5 =     hardware: out of paper.
                        C  ;    bit      #6 =     hardware: acknowledge.
                        C  ;    bit      #7 =     hardware: not busy.
                        C  ;
                        C  ; "Good" Statuses are:         90h & 10h if a printer is connected.
                        C  ;                      30h if printer is disconnected.
                        C  ;===========================================================================
                        C
                        C       XORG    0EFD2h
EFD2                    C1      org     0EFD2h
                        C
EFD2                    C  p_io proc    near
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
EFD2  FB                C       sti                                         ; enable interrupts
                        C
EFD3  83 FA 04          C       cmp     dx,4                                ; 4 printers allowed max
EFD6  73 32             C       jae     p_nop
                        C
                        C               assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
EFD8  51                C       push    cx                                  ; save registers
EFD9  52                C       push    dx
EFDA  57                C       push    di
EFDB  86 C4             C       xchg    al,ah                               ; reverse al & ah
                        C                                                   ; ah saves al throughout
EFDD  1E                C       push    ds                                  ; save ds
```

```
EFDE  2E: 8E 1E E574 R      C       mov     ds,word ptr cs:[set_ds_word]                    ; satisfy assumptions
                            C
EFE3  8B FA                 C       mov     di,dx                                           ; get port number (0-3)
EFE5  33 C9                 C       xor     cx,cx                                           ; clear ch
EFE7  8A 8D 0078 R          C       mov     cl,byte ptr ds:[di+printer_t_out]   ; get printer time-out
EFEB  D1 E1                 C       sal     cx,1                                            ; double it 4 faster cpu
EFED  03 FF                 C       add     di,di                                           ; make word index
EFEF  8B 95 0008 R          C       mov     dx,word ptr ds:[di+printer_addr]    ; get address of printer
                            C                                                               ; data port
EFF3  1F                    C       pop     ds                                              ; restore ds
                            C
                            C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
                            C
EFF4  0B D2                 C       or      dx,dx                                           ; is a printer there?
EFF6  74 0D                 C       jz      p_ret
                            C
EFF8  33 FF                 C       xor     di,di                                           ; clear di
                            C
EFFA  0A C0                 C       or      al,al                                           ; al = 0?
EFFC  74 0D                 C       jz      p_out                                           ; dx = prt_data_x
                            C
EFFE  42                    C       inc     dx                                              ; dx = prt_stat_x
                            C
EFFF  2C 02                 C       sub     al,2                                            ; al = 1 < 2?
F001  72 24                 C       jb      p_init                                          ; dx = prt_stat_x
                            C                                                               ; al = 2?
F003  74 2F                 C       je      p_stat                                          ; dx = prt_stat_x
                            C
F005  86 C4                 C p_ret:        xchg    al,ah                                           ; reverse al & ah back
                            C                                                               ; ah saved al throughout
F007  5F                    C       pop     di                                              ; restore registers
F008  5A                    C       pop     dx
F009  59                    C       pop     cx
F00A  CF                    C p_nop:        iret
                            C
                            C ;-----------------------------------------------------------------------
                            C ;   Print Character to Parallel Printer Interface.
                            C ;
                            C ;   Input:   ah =     character to print
                            C ;            cx =     printer time-out
                            C ;            dx =     address of printer data port (prt_data_x).
                            C ;            di =     0
                            C ;   Output:  ah =     character to print
                            C ;            al =     status of printer: bit #0 set if time out
                            C ;            dx =     address of printer status port (prt_stat_x).
                            C ;   Trash:   cx & di destroyed.
                            C ;
                            C ;   Assumes: prt_stat_x = prt_data_x + 1 = dx + 1
                            C ;                      prt_cmd_x  = prt_data_x + 2 = dx + 2
                            C ;-----------------------------------------------------------------------
                            C
                            C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                            C
F00B  8A C4                 C p_out:        mov     al,ah                                   ; get character to print
F00D  EE                    C       out     dx,al                                   ; output character
```

```
                        C
F00E  42                C       inc     dx                              ; dx = prt_stat_x
                        C
F00F  EC                C  p_lp:in      al,dx                           ; get printer status
F010  24 F8             C       and     al,0F8h                         ; clear bogus printer bits
F012  34 49             C       xor     al,049h                         ; flip acknowledge & I/O err bit
                        C                                               ; & set printer time-out bit #0
F014  78 07             C       js      p_ok                            ; wait for not busy bit #7 set
                        C
                        C  ;    mov     ax,90FEh                ; wait for printer not busy
                        C  ;    INT     15h                             ; another task ?
                        C
F016  4F                C       dec     di                              ; inner loop counter
F017  75 F6             C       jnz     p_lp                            ; inner loop
F019  E2 F4             C       loop    p_lp                            ; outer loop
F01B  EB E8             C       jmp     short p_ret                     ; return status with time-out
                        C
F01D  42                C  p_ok:inc     dx                              ; dx = prt_cmd_x
F01E  B0 0D             C       mov     al,0Dh                          ; set strobe high (al = 0Dh)
F020  EE                C       out     dx,al
F021  90                C       nop
F022  48                C       dec     ax                              ; set strobe low (al = 0Ch)
F023  EE                C       out     dx,al
F024  4A                C       dec     dx                              ; dx = prt_stat_x
                        C
F025  EB 0D             C       jmp     short p_stat
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Initialize Parallel Printer Interface.
                        C  ;
                        C  ;    Input:  ah =    byte to return in al.
                        C  ;            dx =    address of printer status port (prt_stat_x).
                        C  ;    Output: al =    status of printer
                        C  ;            dx =    address of printer status port (prt_stat_x).
                        C  ;    Trash:  cx =    0 destroyed.
                        C  ;
                        C  ;    Assumes: prt_stat_x = prt_data_x + 1 = dx + 1
                        C  ;             prt_cmd_x  = prt_data_x + 2 = dx + 2
                        C  ;--------------------------------------------------------------------
                        C
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
F027  B0 08             C  p_init: mov    al,08h                          ; request init (hold line low)
F029  42                C       inc     dx                              ; dx = prt_cmd_x
F02A  EE                C       out     dx,al
                        C
F02B  B5 05             C       mov     ch,05h                          ; delay awhile (cx = 05??h)
F02D  E2 FE             C       loop    $
                        C
F02F  B0 0C             C       mov     al,0Ch                          ; disable interupts, manual lf
F031  EE                C       out     dx,al                           ; (init done - line set high)
F032  90                C       nop
                        C
F033  4A                C       dec     dx                              ; dx = prt_stat_x
                        C  ;    jmp     short p_stat                    ; fall through
```

```
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Read Status of Parallel Printer Interface.
                        C  ;
                        C  ;    Input:   dx =     address of printer status port (prt_stat_x).
                        C  ;
                        C  ;    Output:  al =     status of printer
                        C  ;             dx =     address of printer status port (prt_stat_x).
                        C  ;
                        C  ;    Assumes: prt_stat_x = prt_data_x + 1 = dx + 1
                        C  ;--------------------------------------------------------------------
                        C
                        C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
F034  EC                C  p_stat: in    al,dx                          ; get printer status
F035  24 F8             C       and      al,0F8h                        ; clear bogus printer bits
F037  34 48             C       xor      al,048h                        ; flip acknowledge & I/O err bit
F039  EB CA             C       jmp     short p_ret                   ; exit
                        C
                        C  p_io endp
                        C
F03B                    C  code ends

                        C  include vid.asm
                        C
                        C  ;====================================================================
                        C  ;    Filename:vid.src
                        C  ;
                        C  ;    This module includes INT 10h, the display routines.
                        C  ;
                        C  ;====================================================================
                        C
                        C  ; These constants must be defined (amount to scroll/clear during vert retrace):
= 00E0                  C       V_KSCROLL1       equ      224      ; 314 chars to move during vert. retrace
= 0162                  C       V_KSCROLL2       equ      354      ; 354 chars to move during vert. retrace
                        C
F03B                    C  code segment  public 'ROM'
                        C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;  Call table, for invoking INT 10h functions, based on value in AH
                        C  ;--------------------------------------------------------------------
                        C
                        C       XORG     0F045h
F045                    C1      org      0F045h
                        C
F045                    C  v_data1       proc     near
                        C
F045  F150 R            C  v_tbldw       v_set_mode                ; ah = 00h
F047  F23D R            C       dw       v_curs_type               ; ah = 01h
F049  F24B R            C       dw       v_curs_pos                ; ah = 02h
F04B  F269 R            C       dw       v_r_curs_pos              ; ah = 03h
F04D  F633 R            C       dw       grf_light_pen             ; ah = 04h       (see lp.src)
F04F  F280 R            C       dw       v_page                    ; ah = 05h
F051  F2D3 R            C       dw       v_scrl_up         ; ah = 06h
```

```
F053  F417 R              C       dw      v_scrl_dn       ; ah = 07h
F055  F438 R              C       dw      v_rac                   ; ah = 08h
F057  F462 R              C       dw      v_wac                   ; ah = 09h
F059  F497 R              C       dw      v_wc                    ; ah = 0Ah
F05B  F4D2 R              C       dw      v_col                   ; ah = 0Bh
F05D  D619 R              C       dw      grf_write_dot           ; ah = 0Ch      (see graph.src)
F05F  D5FF R              C       dw      grf_read_dot            ; ah = 0Dh      (see graph.src)
F061  F4FE R              C       dw      v_terminal              ; ah = 0Eh
F063  F587 R              C       dw      v_stat                  ; ah = 0Fh
                          C
                          C  v_data1      endp
                          C
                          C  ;--------------------------------------------------------------------
                          C  ;    INT 10h --  Video Interrupt Service Routine.
                          C  ;--------------------------------------------------------------------
                          C  ;    -- Set CPU flags.
                          C  ;    -- Segment registers properly loaded.
                          C  ;    -- CALLs routines with:    -- al, bx, cx, dx intact
                          C  ;                               -- ah = v_mode
                          C  ;                               -- si = 2 * (function that was in ah)
                          C  ;                               -- di = bits #4 & 5 of switch_bits
                          C  ;                               -- bp = value of ax to be returned
                          C  ;
                          C  ;    Input:  ah      = function number (00h <= ah <= 0Fh)
                          C  ;
                          C  ;    Trash:  None. (bp, si, di, ds, & es if ROM stack)
                          C  ;--------------------------------------------------------------------
                          C
                          C       XORG    0F065h
F065                      C1      org     0F065h
                          C
F065                      C  v_io proc    near
                          C
                          C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
                          C
F065  FB                  C       sti                                     ; enable interrupts
F066  80 FC 0F            C       cmp     ah,0Fh                          ; input out of range?
F069  77 03               C       ja      v_nop
F06B  E9 F0FC R           C       jmp     v_io_2                          ;if not, GOTO v_io Part 2
                          C
F06E                      C  v_nop:
F06E  CF                  C       iret
                          C
                          C  ; The remainder of v_io follows the video parameters
                          C
                          C  v_io endp
                          C
                          C  ;--------------------------------------------------------------------
                          C
                          C       XORG    0F0A4h
F0A4                      C1      org     0F0A4h
                          C
F0A4                      C  v_data2      proc    near
                          C
F0A4                      C  v_parms      label   byte
```

```
                        C
                        C  ; 6845 Parameters except register 3h (Horizontal Synch Width).
                        C
F0A4  38 28 2D 0A       C  v_md_40              db      38h,28h,2Dh,0Ah   ; text   40 x 25
F0A8  1F 06 19 1C       C                db     1Fh,06h,19h,1Ch   ; mode 0 -> monochrome
F0AC  02 07 06 07       C                db     02h,07h,06h,07h   ; mode 1 -> color
F0B0  00 00 00 00       C                db     00h,00h,00h,00h
                        C
F0B4  71 50 5A 0C       C  v_md_80              db      71h,50h,5Ah,0Ch   ; text   80 x 25
F0B8  1F 06 19 1C       C                db     1Fh,06h,19h,1Ch   ; mode 2 -> monochrome
F0BC  02 07 06 07       C                db     02h,07h,06h,07h   ; mode 3 -> color
F0C0  00 00 00 00       C                db     00h,00h,00h,00h
                        C
F0C4  38 28 2D 0A       C  v_md_graph    db     38h,28h,2Dh,0Ah   ; graphics
F0C8  7F 06 64 70       C                db     7Fh,06h,64h,70h   ; mode 4 -> 320 x 200 color
F0CC  02 01 06 07       C                db     02h,01h,06h,07h   ; mode 5 -> 320 x 200 monochrome
F0D0  00 00 00 00       C                db     00h,00h,00h,00h   ; mode 6 -> 640 x 200 monochrome
                        C
F0D4  61 50 52 0F       C  v_md_mono     db     61h,50h,52h,0Fh   ; monochrome card 80 x 25
F0D8  19 06 19 19       C                db     19h,06h,19h,19h   ; mode 7 -> monochrome card
F0DC  02 0D 0B 0C       C                db    02h,0Dh,0Bh,0Ch ;
F0E0  00 00 00 00       C                db     00h,00h,00h,00h
                        C
F0E4  0800              C  v_md_len      dw     2048              ; 40x25          modes 0 & 1
F0E6  1000              C                dw     4096              ; 80x25          modes 2 & 3
F0E8  4000              C                dw     16384             ; graphics       modes 4 & 5
F0EA  4000              C                dw     16384             ;                mode  6
                        C  ;only in 1050 dw     32768             ;                modes 64 & 72
                        C
F0EC  28                C  v_md_wid      db     40                ; 0 -> text       40x 25  monochrome
F0ED  28                C                db     40                ; 1 -> text       40x 25  color
F0EE  50                C                db     80                ; 2 -> text       80x 25  monochrome
F0EF  50                C                db     80                ; 3 -> text       80x 25  color
F0F0  28                C                db     40                ; 4 -> graphics  320x200  color
F0F1  28                C                db     40                ; 5 -> graphics  320x200  monochrome
F0F2  50                C                db     80                ; 6 -> graphics  640x200  monochrome
F0F3  50                C                db     80                ; 7 -> text       80x 25  monochrome card
F0F4  2C                C  v_md_enable   db     2Ch               ; 0 -> text       40x 25  monochrome
F0F5  28                C                db     28h               ; 1 -> text       40x 25  color
F0F6  2D                C                db     2Dh               ; 2 -> text       80x 25  monochrome
F0F7  29                C                db     29h               ; 3 -> text       80x 25  color
F0F8  2A                C                db     2Ah               ; 4 -> graphics  320x200  color
F0F9  2E                C                db     2Eh               ; 5 -> graphics  320x200  monochrome
F0FA  1E                C                db     1Eh               ; 6 -> graphics  640x200  monochrome
F0FB  29                C                db     29h               ; 7 -> text       80x 25  monochrome card
                        C
                        C  v_data2       endp
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;
                        C  ; This is the second part of v_io, moved here because there wasn't room
                        C  ; enough, ahead of the video parameters, for the entire thing.
                        C  ;
                        C  ;--------------------------------------------------------------------
                        C
```

```
F0FC                    C  v_io_2       proc    near
                        C               assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
F0FC  06                C       push    es                              ; save 'trashable' registers
F0FD  1E                C       push    ds
F0FE  2E: 8E 1E E574 R  C       mov     ds,word ptr cs:[set_ds_word]    ; avoid potential stack problems
F103  57                C       push    di
F104  56                C       push    si
F105  55                C       push    bp
                        C
                        C               assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
F106  50                C       push    ax                              ; save ax
                        C
                        C  ; Test for absence of video controller or presence of EGA card
                        C
F107  E4 67             C       in      al,sys_conf_b                   ; read switch at port 67H
F109  A8 30             C       test    al,030H                         ; test for Sw. 5 & 6 = 00
F10B  75 13             C       jnz     v_io_ok                         ; if not 00, OK to do vidio
F10D  33 C0             C       xor     ax,ax                           ; zeros, to address segment 0
F10F  8E C0             C       mov     es,ax                           ; now es addresss abs0 segment
F111  26: A1 0040 R     C       mov     ax,word ptr es:int10locn    ; get INT10H vector value
F115  3D F065 R         C       cmp     ax,offset v_io                  ; has INT 10 pointer changed ?
F118  58                C       pop     ax                              ;  retrieve function code in AH
F119  50                C       push    ax                              ;  save AX  again
F11A  75 04             C       jne     v_io_ok                         ; if so, we assume EGA present,
F11C  58                C       pop     ax                              ; else, no video: restore AX
F11D  EB 2B 90          C       jmp     v_io_ret                ; and go to exit
                        C
F120                    C  v_io_ok:
F120  8A C4             C       mov     al,ah                           ; al = function number
F122  02 C4             C       add     al,ah                           ; * 2
F124  98                C       cbw
F125  50                C       push    ax                              ; save for CALL later.
                        C
F126  BF B800           C       mov     di,para_graph                   ; get screen segment ..
F129  A0 0010 R         C       mov     al,byte ptr ds:[switch_bits]    ; .. check switch bits ..
F12C  A8 20             C       test    al,20h
F12E  74 0A             C       jz      v_colour                ; .. if either is 0, it's
F130  A8 10             C       test    al,10h                          ; .. color display board
F132  74 06             C       jz      v_colour
F134  81 C7 F800        C       add     di,para_mono-para_graph         ; .. it's a monochrome board
F138  EB 00             C       jmp short v_colour
                        C
F13A                    C  v_colour:
F13A  8E C7             C       mov     es,di                           ; set es = video ram
                        C
F13C  5F                C       pop     di                              ; Get index to call table.
                        C
F13D  58                C       pop     ax                              ; restore ax
F13E  8A 26 0049 R      C       mov     ah,byte ptr ds:[v_mode]         ; get display driver mode
F142  8B E8             C       mov     bp,ax                           ; bp saves ax throughout
                        C                                               ; (return ah = v_mode
                        C                                               ;  unless specific ret. value)
                        C
```

```
F144  FC                  C       cld                                     ; String ops move UP, mostly.
F145  2E: FF 95 F045 R    C       call    cs:[di+(offset v_tbl)]          ; perform display function
                          C
F14A                      C  v_io_ret:
F14A  5D                  C       pop     bp                              ; restore 'trashed' registers
F14B  5E                  C       pop     si                              ; (destroyed if ROM stack)
F14C  5F                  C       pop     di
F14D  1F                  C       pop     ds
F14E  07                  C       pop     es
                          C
F14F  CF                  C       iret
                          C
                          C  v_io_2       endp
                          C  ;--------------------------------------------------------------------
                          C  ;    Set Mode & Clear Screen    ah = 00h
                          C  ;
                          C  ;    Input:  al      = mode  = 0 -> text       40x 25  monochrome
                          C  ;                            = 1 -> text       40x 25  color
                          C  ;                            = 2 -> text       80x 25  monochrome
                          C  ;                            = 3 -> text       80x 25  color
                          C  ;                            = 4 -> graphics   320x200 color
                          C  ;                            = 5 -> graphics   320x200 monochrome
                          C  ;                            = 6 -> graphics   640x200 monochrome
                          C  ;                            = 7 -> text       80x 25  monochrome card
                          C  ;                            = 64 -> graphics 640x400 monochrome
                          C  ;                            = 72 -> graphics 640x400 monochrome tinytext
                          C  ;
                          C  ;    Output: ah      = 00h
                          C  ;            al      = v_colorpal
                          C  ;
                          C  ;    Assume:  contents of v_base6845     = pointer register              (3D4h)
                          C  ;            (contents of v_base6845)+1  = data register         (3D5h)
                          C  ;            (contents of v_base6845)+4  = mode control register (3D8h)
                          C  ;            (contents of v_base6845)+5  = overscan register             (3D9h)
                          C  ;            (contents of v_base6845)+6  = status register       (3DAh)
                          C  ;            (contents of v_base6845)+10 = mode control register #2(3DEh)
                          C  ;
                          C  ;    Trash:  si & di destroyed.
                          C  ;--------------------------------------------------------------------
                          C
F150                      C  v_set_mode   proc    near
                          C
                          C               assume  cs:code, ds:data, es:v_ram, ss:nothing
                          C
F150  52                  C       push    dx                              ; save dx
F151  51                  C       push    cx                              ; save cx
                          C
                          C  ; Get Color/Monochrome dependent stuff.
                          C
F152  32 E4               C       xor     ah,ah                           ; AH = mode ctrl., color board
F154  BA 03D4             C       mov     dx,color_pointer          ; color 6845 pointer register.
                          C
F157  F6 06 0010 R 10     C       test    byte ptr ds:[switch_bits],10h   ; monochrome board?
F15C  74 0D               C       jz      v_set_mode_color          ; if not, skip monochrome stuff.
F15E  F6 06 0010 R 20     C       test    byte ptr ds:[switch_bits],20h
```

```
F163  74 06              C       jz      v_set_mode_color
                         C                                               ; It's a monochrome board, so..
F165  B8 0107            C       mov     ax,0107h                ; (AH,AL) = (overwrite mode
                         C                                               ; for monochrome, mono. mode)
F168  83 C2 E0           C       add     dx,v_pointer-color_pointer; monochrome pointer register.
                         C
F16B                     C  v_set_mode_color:
                         C
                         C  ; Save CRT Mode & 6845 address, and reset display monitor with mode control.
                         C
F16B  A2 0049 R          C       mov     byte ptr ds:[v_mode],al     ; save mode.
F16E  89 16 0063 R       C       mov     word ptr ds:[v_base6845],dx; save 6845 pointer register.
F172  86 E0              C       xchg    ah,al                           ; AH=mode, AL=mode ctrl.
F174  83 C2 04           C       add     dx,4                            ; get 6845 mode control register
F177  EE                 C       out     dx,al                           ; reset display
F178  83 EA 04           C       sub     dx,4                            ; restore 6845 address.
F17B  8A C4              C       mov     al,ah                           ; restore mode in al
                         C
                         C  ; Get pointer to display parameters.
                         C
F17D  1E                 C       push    ds                              ; save ds = data_seg
                         C
                         C               assume  cs:code, ds:abs0, es:v_ram, ss:nothing
                         C
F17E  33 F6              C       xor     si,si
F180  8E DE              C       mov     ds,si                           ; satisfy assumptions
F182  C5 36 0074 R       C       lds     si,dword ptr ds:[int1Dlocn]; display parameter pointer
                         C
                         C  ; ds:si in effect points to cs:v_parms.
                         C
                         C               assume  cs:code, ds:code, es:v_ram, ss:nothing
                         C
                         C  ; Determine which set of parameters to use from mode.
                         C
F186  B9 0010            C       mov     cx,16                           ; count of parameters
F189  3C 02              C       cmp     al,2                            ; 40x25 mode?  (0 & 1?)
F18B  72 0E              C       jb      v_set_mode_lp                   ; if so, we're done
F18D  03 F1              C       add     si,cx                           ; next set of parameters
F18F  3C 04              C       cmp     al,4                            ; 80x25 mode?  (2 & 3?)
F191  72 08              C       jb      v_set_mode_lp                   ; if so, we're done
F193  03 F1              C       add     si,cx                           ; next set of parameters
F195  3C 07              C       cmp     al,7                            ; graphics mode?  (4,5,6,64,72?)
F197  75 02              C       jne     v_set_mode_lp                   ; if so, we're done
F199  03 F1              C       add     si,cx                           ; else, monochrome card (7)
                         C
                         C  ; Loop through 6845 initialization table outputting register number and data
                         C
F19B                     C  v_set_mode_lp:
                         C
F19B  B0 10              C       mov     al,16                           ; get 6845 register number =
                         C                                               ;  = (16 - cl) (unscrambled)
F19D  2A C1              C       sub     al,cl
F19F  EE                 C       out     dx,al                           ; output to register port
F1A0  42                 C       inc     dx                              ; point to 6845 data register
F1A1  AC                 C       lodsb                                   ; get parm value:  al gets ds:si
```

```
F1A2  EE              C       out     dx,al                           ; output to data port
F1A3  4A              C       dec     dx                              ; point back to pointer register
F1A4  E2 F5           C       loop    v_set_mode_lp                   ; next register
                      C                                               ; dx = pointer register
                      C
                      C               assume  cs:code, ds:data, es:v_ram, ss:nothing
                      C
F1A6  1F              C       pop     ds                              ; restore ds = data_seg
                      C
F1A7  8A C4           C       mov     al,ah                           ; save mode in ah & al
F1A9  32 E4           C       xor     ah,ah                           ; ax = mode
F1AB  8B F0           C       mov     si,ax                           ; si = mode hence...
                      C
                      C ; Clear the screen.
                      C
F1AD  B9 2000         C       mov     cx,2000h                ; assume 8k words to clear
                      C
F1B0  3C 04           C       cmp     al,4                            ; 40x25 or 80x25 text modes 0-3?
F1B2  72 0E           C       jb      v_md_clr_8k                     ; if so, clear 8k words.
F1B4  3C 07           C       cmp     al,7                            ; monochrome card mode 7?
F1B6  74 08           C       je      v_md_clr_2k                     ; if so, clear 2k words.
F1B8  72 02           C       jb      v_md_clr_graphics       ; graphics mode 4-6 clear 8k wds
F1BA  D1 E1           C       shl     cx,1                            ;   mode 64 & 72 clear 16k wds
F1BC                  C v_md_clr_graphics:
F1BC  33 C0           C       xor     ax,ax                           ; graphics mode
F1BE  EB 05           C       jmp     short v_md_clr                  ; clear screen with zeroes
                      C
F1C0                  C v_md_clr_2k:
F1C0  B5 08           C       mov     ch,08h                          ; cx = 0800h
F1C2                  C v_md_clr_8k:
F1C2  B8 0720         C       mov     ax,(7*100h)+(' ')       ; clear with attribute & space
F1C5                  C v_md_clr:
F1C5  33 FF           C       xor     di,di
F1C7  F3/ AB          C       rep     stosw                           ; es:di gets ax
                      C
                      C ; Set mode control register #2
                      C
                      C ; Handle underline on shades-of-gray monitor.
                      C                                               ; dx = pointer register
F1C9  83 C2 06        C       add     dx,6                            ; get 6845 status register
F1CC  EC              C       in      al,dx                           ; get CRT status
F1CD  24 10           C       and     al,010h                         ; isolate color/shades bit #4
F1CF  D0 E0           C       shl     al,1                            ; move it to underline bit #6
F1D1  D0 E0           C       shl     al,1
                      C
                      C ; Handle double scan line modes 64 & 72.
                      C                                               ; dx = 6845 status register
F1D3  83 FE 40        C       cmp     si,64                           ; modes 0 through 7?
F1D6  72 04           C       jb      v_md_dbl                ; if so, single scan line mode
F1D8  40              C       inc     ax                              ; else mode 64 or 72, set bit #0
F1D9  BE 0006         C       mov     si,6                            ; modes 64 & 72 look like mode 6
                      C                                               ;    from now on
                      C
F1DC                  C v_md_dbl:                                     ; double scan line mode.
F1DC  83 C2 04        C       add     dx,4                            ; get mode control register #2
```

```
F1DF  EE                  C       out     dx,al
                          C
                          C  ; Enable display monitor with mode control.
                          C
F1E0  2E: 8A 84 F0F4 R    C       mov     al,byte ptr cs:[si+v_md_enable]
F1E5  A2 0065 R           C       mov     byte ptr ds:[v_3x8],al      ; save the value for later
                          C                                           ; dx = mode control register #2
F1E8  83 EA 06            C       sub     dx,6                        ; get 6845 mode control register
F1EB  EE                  C       out     dx,al                       ; enable display
                          C
                          C  ; Determine width & length of screen.
                          C
F1EC  33 C0               C       xor     ax,ax                       ; clear ah
F1EE  2E: 8A 84 F0EC R    C       mov     al,byte ptr cs:[si+v_md_wid]
F1F3  A3 004A R           C       mov     word ptr ds:[v_width],ax
                          C
F1F6  81 E6 000E          C       and     si,0Eh                      ; make word index divided by 2
F1FA  2E: 8B 84 F0E4 R    C       mov     ax,word ptr cs:[si+v_md_len]
F1FF  A3 004C R           C       mov     word ptr ds:[v_height],ax
                          C
                          C  ; Set up overscan register & v_colorpal.
                          C
F202  B0 30               C       mov     al,030h                     ; v_colorpal for modes 0-5 & 7
F204  8A 26 0049 R        C       mov     ah,byte ptr ds:[v_mode]     ; retrieve v_mode
F208  80 FC 06            C       cmp     ah,6                        ; modes 0-5 ?
F20B  72 0D               C       jb      v_ovr_ok                    ; if so, we're ok.
                          C
F20D  74 09               C       je      v_ovr_not_ok                ; 640x200 graphics mode 6 ?
                          C                                           ; if so, change v_colorpal
                          C
F20F  80 FC 40            C       cmp     ah,64                       ; 640x400 graphics mode 64, 72 ?
F212  72 06               C       jb      v_ovr_ok                    ; if not, we're ok.
                          C
                          C                                           ; if so we set v_height wrong.
F214  D1 26 004C R        C       shl     word ptr ds:[v_height],1    ; double v_height from 16k to 32k
                          C
F218                      C  v_ovr_not_ok:
F218  B0 3F               C       mov     al,03Fh                     ; v_colorpal for modes 6,64,72
                          C
F21A                      C  v_ovr_ok:                                ; dx = 6845 mode control register
F21A  A2 0066 R           C       mov     byte ptr ds:[v_colorpal],al; save the value for later
F21D  42                  C       inc     dx                          ; get 6845 overscan register
F21E  EE                  C       out     dx,al
                          C
                          C  ; Clear all cursor positions.
                          C
                          C               assume  cs:code, ds:data, es:data, ss:nothing
                          C
F21F  1E                  C       push    ds                          ; set es = firmware data area
F220  07                  C       pop     es
                          C
F221  BF 0050 R           C       mov     di,ds:(offset v_curpos)     ; get address [defined by DB]
F224  B9 0008             C       mov     cx,8
F227  33 C0               C       xor     ax,ax                       ; ax = 0
F229  F3/ AB              C       rep     stosw                       ; es:di gets ax = 0
```

```
                        C
                        C  ; Clear other firmware data variables (ax = 0).
                        C
F22B  A3 004E R         C       mov     word ptr ds:[v_top],ax      ; set starting offset to 0
F22E  A2 0062 R         C       mov     byte ptr ds:[v_apage],al    ; set current active page to 0
F231  C7 06 0060 R 0607 C       mov     word ptr ds:[v_cursize],0607h       ; set cursor mode
                        C
                        C  ; Clean up.
                        C
F237  A0 0066 R         C       mov     al,byte ptr ds:[v_colorpal]
F23A  59                C       pop     cx                                  ; restore cx
F23B  5A                C       pop     dx                                  ; restore dx
F23C  C3                C       ret
                        C
                        C  v_set_mode   endp
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;    Set Cursor Value           ah = 01h
                        C  ;
                        C  ;    Input:  ch      = bits #0-4 = starting line for cursor
                        C  ;            cl      = bits #0-4 = ending line for cursor
                        C  ;    Output: ah      = 10
                        C  ;            al      = cl
                        C  ;
                        C  ;    Trash:  si & di destroyed. (si = dx; di = cx)
                        C  ;---------------------------------------------------------------------
                        C
F23D                    C  v_curs_type  proc    near
                        C       assume  cs:code, ds:data, es:v_ram, ss:nothing
                        C
F23D  89 0E 0060 R      C       mov     word ptr ds:[v_cursize],cx ; save the cursor value
                        C
F241  8B F9             C       mov     di,cx                               ; di saves cx
                        C
F243  B4 0A             C       mov     ah,10                               ; 6845 cursor set register = 10
F245  E8 F2B6 R         C       call    v_6845                              ; set cursor; ah 6845 gets cx
                        C                                                   ;  si = dx destroyed
                        C                                                   ;  ah = preserved = 10; al = cl
                        C                                                   ;  di restores cx
F248  8A C1             C       mov     al,cl                               ; restore al with original cl
F24A  C3                C       ret
                        C  v_curs_type  endp
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;    Set Cursor Position             ah = 02h
                        C  ;
                        C  ;    Input:  bh      = page number (0-7)
                        C  ;            (dh,dl) = (row,col) of current cursor from (0,0)
                        C  ;    Output: if bh = v_apage,  ah = 14
                        C  ;                                    al = low byte of cursor position
                        C  ;            else,                   ah = v_mode
                        C  ;                                    al = preserved
                        C  ;
                        C  ;    Trash:  si & di destroyed. (si = dx, di = cx)
                        C  ;---------------------------------------------------------------------
```

```
                        C
F24B                    C  v_curs_pos   proc    near
                        C
                        C       assume  cs:code, ds:data, es:v_ram, ss:nothing
                        C
F24B  8B F9             C       mov     di,cx                           ; save cx
                        C
F24D  8A CF             C       mov     cl,bh
F24F  8B F1             C       mov     si,cx
F251  81 E6 0007        C       and     si,7                            ; mask to 8 pages
F255  03 F6             C       add     si,si                           ; *2 => word index
                        C
F257  89 94 0050 R      C       mov     word ptr ds:[si+v_curpos],dx    ; save the cursor position
                        C
F25B  3A 3E 0062 R      C       cmp     bh,byte ptr ds:[v_apage]    ; if active page, put the cursor
F25F  74 03             C       je      v_set_curs                  ; on the screen.
                        C                                           ; not active page, so just
F261  8B CF             C       mov     cx,di                       ;  restore cx
F263  C3                C       ret                                 ;  and exit
                        C
F264                    C  v_set_curs:                              ; active page, so set cursor..
F264  8B C2             C       mov     ax,dx                       ; ax gets cursor position
F266  EB 42 90          C       jmp     v_set_cur_pos               ; set cursor; ah 6845 gets cx
                        C                                           ;  si = dx destroyed
                        C                                           ;  ah = preserved = 14
                        C                                           ;  al = low byte of cursor posn
                        C
                        C  v_curs_pos   endp
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Read Cursor                         ah = 03h
                        C  ;
                        C  ;    Input:  bh      = page number (0-7)
                        C  ;    Output: (dh,dl) = (row,col) of current cursor from (0,0)
                        C  ;            (ch,cl) = current cursor mode setting
                        C  ;            ax      = dx
                        C  ;
                        C  ;    Trash:  None.
                        C  ;--------------------------------------------------------------------
                        C
F269                    C  v_r_curs_pos proc    near
                        C
                        C               assume  cs:code, ds:data, es:v_ram, ss:nothing
                        C
F269  8B C2             C       mov     ax,dx
F26B  8B CB             C       mov     cx,bx                           ; save bx
F26D  8A DF             C       mov     bl,bh
F26F  81 E3 0007        C       and     bx,07h                          ; page number mod 8
F273  D1 E3             C       shl     bx,1                            ; page number mod 8 word index
F275  8B 97 0050 R      C       mov     dx,word ptr ds:[bx+v_curpos]
F279  8B D9             C       mov     bx,cx                           ; restore bx
F27B  8B 0E 0060 R      C       mov     cx,word ptr ds:[v_cursize]
F27F  C3                C       ret
                        C
                        C  v_r_curs_pos endp
```

```
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Read Light Pen (see graph.src)      ah = 04h
                        C  ;--------------------------------------------------------------------
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    Set Active Display Page             ah = 05h
                        C  ;
                        C  ;    Input:   al       = new page number (0-7 for modes 0-1; 0-3 for 2-3)
                        C  ;    Output:  6845 is reset to display the new active page
                        C  ;             ah       = 14
                        C  ;             al       = low byte of cursor position
                        C  ;
                        C  ;    Trash:   bp, si & di destroyed. (si = dx, di = cx)
                        C  ;--------------------------------------------------------------------
                        C
F280                    C  v_page       proc    near
                        C
                        C       assume  cs:code, ds:data, es:v_ram, ss:nothing
                        C
F280  25 0007           C       and     ax,07h                          ; page number mod 8
F283  8B E8             C       mov     bp,ax                           ; save ax = page number mod 8
                        C
F285  A2 0062 R         C       mov     byte ptr ds:[v_apage],al    ; save active page number (0-7)
F288  74 08             C       jz      v_page_0                    ; page number = 0?
                        C
F28A  8B FA             C       mov     di,dx                           ; save dx
F28C  F7 26 004C R      C       mul     word ptr ds:[v_height]          ; dx:ax = (page number)*v_height
F290  8B D7             C       mov     dx,di                           ; restore dx
                        C
F292                    C  v_page_0:
F292  A3 004E R         C       mov     word ptr ds:[v_top],ax          ; save starting address of page
F295  D1 F8             C       sar     ax,1                            ; divide by 2 for byte count
                        C
                        C
F297  8B F9             C       mov     di,cx                           ; save cx
                        C
F299  8B C8             C       mov     cx,ax                           ; cx gets offset of active page
F29B  B4 0C             C       mov     ah,12                           ; 6845 cursor set address = 12
F29D  E8 F2B6 R         C       call    v_6845                          ; set cursor; ah 6845 gets cx
                        C                                               ;  si = dx destroyed
                        C                                               ;  ah = preserved = 12; al = cl
                        C                                               ;  di restores cx
                        C
F2A0  8B C5             C       mov     ax,bp                           ; restore ax = page number mod 8
F2A2  D1 E0             C       shl     ax,1                            ; page number mod 8 word index
F2A4  8B F0             C       mov     si,ax
F2A6  8B 84 0050 R      C       mov     ax,word ptr ds:[si+v_curpos]    ; get page's cursor position
                        C
F2AA                    C  v_set_cur_pos:                               ;(ah,al) = (row,col) cursor pos.
                        C                                               ; di     = value to return in cx
F2AA  E8 F5E3 R         C       call    v_posn                          ; (ah,al) -> ax offset; si trash
F2AD  03 06 004E R      C       add     ax,word ptr ds:[v_top]          ; add offset of active page
F2B1  D1 F8             C       sar     ax,1                            ; divide by 2 for byte count
                        C
```

```
F2B3  B5 0E              C       mov     ch,14                           ; 6845 cursor pos register = 14
F2B5  91                 C       xchg    cx,ax
                         C
                         C
                         C  ; Now:       ah      = 6845 register selection
                         C  ;    ch      = first data byte to (ah) 6845 internal register
                         C  ;    cl      = second data byte to (ah+1) 6845 internal register
                         C  ;    di      = value to return in cx
03                          C
                         C
F2B6                     C  v_6845:              ; program 6845 cursor:
                         C
F2B6  8B F2              C       mov     si,dx                           ; save dx
F2B8  E8 F2C7 R          C       call    v_out_byte                      ; output to 6845
F2BB  FE C4              C       inc     ah                              ; next 6845 register
F2BD  8A E9              C       mov     ch,cl                           ; get the register input
F2BF  E8 F2C7 R          C       call    v_out_byte                      ; output to 6845
F2C2  8B D6              C       mov     dx,si                           ; restore dx value
F2C4  8B CF              C       mov     cx,di                           ; set up return value
F2C6  C3                 C       ret
                         C  v_page       endp
                         C
                         C  ;--------------------------------------------------------------------------
                         C  ;    Output two byte to the selected 6845 registers
                         C  ;
                         C  ;    Input:  ah      = 6845 register selection
                         C  ;            ch      = first data byte to (ah) 6845 internal register
                         C  ;            cl      = second data byte to (ah+1) 6845 internal register
                         C  ;            di      = value to return in cx
                         C  ;
                         C  ;    Assume: contents of v_base6845      = pointer register
                         C  ;          (contents of v_base6845)+1      = data register
                         C  ;
                         C  ;    Output: ah      = 6845 register selection
                         C  ;            al      = second data byte
                         C  ;            cx      = di
                         C  ;--------------------------------------------------------------------------
                         C
F2C7                     C  v_out_byte   proc    near
                         C
                         C       assume  cs:code, ds:data, es:v_ram, ss:nothing
                         C
F2C7  8B 16 0063 R       C       mov     dx,word ptr ds:[v_base6845]; get 6845 pointer register
F2CB  8A C4              C       mov     al,ah                           ; get the register address
F2CD  EE                 C       out     dx,al                           ; select the data register
                         C
F2CE  42                 C       inc     dx                              ; next register
F2CF  8A C5              C       mov     al,ch                           ; get second data byte
F2D1  EE                 C       out     dx,al                           ; output a data byte to 6845
F2D2  C3                 C       ret
                         C
                         C  v_out_byte   endp
                         C
                         C  ;--------------------------------------------------------------------
                         C  ;    Scroll Active Page Up               ah = 06h
```

```
                        C  ;
                        C  ;    Input:   if al = 0, then clear entire window with attribute in bh
                        C  ;             else,    al = number of rows to 'scroll' up
                        C  ;                          = number of rows to clear at bottom of window
                        C  ;             bh       = attribute to be used on blank row(s)
                        C  ;             (ch,cl)  = (row,col) of upper left corner of window from (0,0)
                        C  ;             (dh,dl)  = (row,col) of lower right corner of window from (0,0)
                        C  ;    Output:  ah       = attribute to be used on blank row(s)
                        C  ;             if v_mode = 7,    al = 20h = space
                        C  ;             else                   al = v_3x8
                        C  ;
                        C  ;    Assume:  (contents of v_base6845)+6  = status register
                        C  ;
                        C  ;    Trash:   bp, si, & di destroyed. (bx thru dx destroyed if ROM stack)
                        C  ;--------------------------------------------------------------------------
                        C
F2D3                    C  v_scrl_up     proc     near
                        C
                        C                assume   cs:code, ds:data, es:v_ram, ss:nothing
                        C
F2D3  E8 F5F4 R         C       call     v_txt_md          ; all registers preserved
F2D6  72 03             C       jb       v_txt_up
F2D8  E9 D6A3 R         C       jmp      grf_graphics_up           ; jump if graphics
F2DB                    C  v_txt_up:
11074                      C
F2DB  52                C       push     dx                        ; save registers
F2DC  51                C       push     cx
F2DD  53                C       push     bx
                        C
F2DE  8A D8             C       mov      bl,al                     ; save line count
F2E0  8B C1             C       mov      ax,cx                     ; pass upper left coordinates....
F2E2  E8 F596 R         C       call     v_scrl_pos                ; to common scroll positioning routine
F2E5  74 44             C       jz       v_clr                     ; clear rows if nothing to move
                        C
                        C  ; Scroll cl rows up.
                        C
                        C  ;v_mv_up:
F2E7  03 F0             C       add      si,ax                     ; add (bytes/row)*(rows to scroll) to
                        C                                          ; 'from' address for scroll
                        C
                        C  ; Scroll cl rows up/down (based on bp & direction flag DF).
                        C
F2E9                    C  v_mv:
F2E9  8A E1             C       mov      ah,cl                     ; ah <- number of rows to scroll
                        C
F2EB  80 3E 0049 R 07   C       cmp      byte ptr ds:[v_mode],7
F2F0  74 29             C       je       v_mv_fast         ; jump to fast loop
                        C
F2F2  53                C       push     bx                        ; save parameters of " clear"
                        C
                        C  ; If a non-Olivetti Video Controller Board, we'll do a slow scroll
                        C
F2F3  BB F396 R         C       mov      bx,offset v_1             ; set up to move one word per scan line
F2F6  E4 67             C       in       al,sys_conf_b             ; read port 67h (switches at 7w)
F2F8  A8 08             C       test     al,NON_OLI_VID            ; non Olivetti video controller ?
```

```
F2FA  75 19              C       jnz     v_mv2                   ; jump if not Olivetti Video Controller
                         C
                         C  ; The dip-switches say it's Olivett.  are we runing on a Turbo (M24SP) ?
                         C
F2FC  E8 F401 R          C       call    v_8253                  ; latch and read counter 0
F2FF  8A FB              C       mov     bh,bl                   ; bh <- lsb of latched count
F301  B1 02              C       mov     cl,2                    ; execute enough instructions to
F303  E2 FE              C  v_mv1:       loop    v_mv1                   ;- distinguish 8 MHz from 10 MHz
F305  E8 F401 R          C       call    v_8253                  ; latch and read counter 0 again
F308  2A FB              C       sub     bh,bl                   ; compute the difference
F30A  80 FF 23           C       cmp     bh,23h                  ; max val is 20h on 10 MHz 8086
F30D  BB F3A0 R          C       mov     bx,offset v_2           ; set up to move two words per scan line
F310  72 03              C       jb      v_mv2                   ; jump if 10 MHz cpu clock
F312  BB F396 R          C       mov     bx,offset v_1           ; set up to move one word per scan line
F315                     C  v_mv2:
                         C
                         C  ;    Perform the scroll
                         C
F315  E8 F35F R          C       call    v_scroll_or_clear
F318  5B                 C       pop     bx                      ; recover the "clear" parameters
F319  EB 10              C       jmp short v_clr                 ; go clear rows at top/bottom
                         C
                         C
                         C  ;---------------------------------------------------------------------
                         C
F31B                     C  v_mv_fast:                           ; ah has number of rows to move
F31B  1E                 C       push    ds                      ; save data segment
F31C  06                 C       push    es                      ; use video RAM address in es for move
F31D  1F                 C       pop     ds                      ; set up ds to address video RAM
F31E                     C  v_mv_flp:                            ; ah has number of rows to move
F31E  8A CA              C       mov     cl,dl                   ; number of columns to move
F320  F3/ A5             C       rep     movsw                   ; move the row (es:di gets ds:si)
F322  03 F5              C       add     si,bp                   ; add number of bytes to skip in row to
                         C                                       ; 'from' address for scroll
F324  03 FD              C       add     di,bp                   ; add number of bytes to skip in row to
                         C                                       ; 'to' address for scroll
F326  FE CC              C       dec     ah                      ; one less row to be moved
                         C
F328  75 F4              C       jnz     v_mv_flp        ; go back if not finished
                         C
F32A  1F                 C       pop     ds                      ;restore data segment
                         C  ;---------------------------------------------------------------------
                         C
                         C  ;    Set up to clear one or more rows
                         C
F32B                     C  v_clr:
F32B  8A E7              C       mov     ah,bh                   ; ah <- attribute of line to clear
F32D  B0 20              C       mov     al,' '                  ; al <- " space"
F32F  80 3E 0049 R 07    C       cmp     byte ptr ds:[v_mode],7
F334  74 0C              C       je      v_clr_fast              ; fast clear if monochrome board
F336  8B F0              C       mov     si,ax                   ; si <- attr:space
F338  8A E3              C       mov     ah,bl                   ; ah <- number of rows to clear
F33A  BB F381 R          C       mov     bx,offset v_0           ; set up for clear
                         C
                         C  ;    Perform the clear
```

```
                        C
F33D  E8 F35F R         C       call    v_scroll_or_clear
                        C
F340  EB 0A             C       jmp short v_clr_fin
                        C
F342                    C  v_clr_fast:
F342  8A CA             C       mov     cl,dl                   ;number of columns to clear
F344  F3/ AB            C       rep stosw               ;clear the row
F346  03 FD             C       add     di,bp                   ;add number of bytes to skip in row
F348  FE CB             C       dec     bl                      ;decrement row counter
F34A  75 F6             C       jnz     v_clr_fast              ;repeat if not finished
                        C
F34C                    C  v_clr_fin:
F34C  80 3E 0049 R 07   C       cmp     byte ptr ds:[v_mode],7
F351  74 07             C       je      v_scrl_mode_7
F353  A0 0065 R         C       mov     al,byte ptr ds:[v_3x8]  ;
F356  BA 03D8           C       mov     dx,03D8h        ;; video enable register.
F359  EE                C       out     dx,al                   ;; enable video.
F35A                    C  v_scrl_mode_7:
                        C
                        C  ;    Restore the registers and return
                        C
F35A  FC                C       cld                             ; restore the direction flag to UP
F35B  5B                C       pop     bx
F35C  59                C       pop     cx
F35D  5A                C       pop     dx
F35E  C3                C       ret
                        C
                        C  ;    Scroll or clear the requested number of rows
                        C
F35F                    C  v_scroll_or_clear:
                        C
F35F  8A CA             C       mov     cl,dl                   ; cx <- number of cols to move per row
F361  52                C       push    dx                      ; save dl for subsequent "clear" call
F362  8B 16 0063 R      C       mov     dx,[v_base6845]
F366  83 C2 06          C       add     dx,6                    ; dx <- addr(video status register)
F369  1E                C       push    ds                      ; save ds
F36A  06                C       push    es
F36B  1F                C       pop     ds                      ; ds <- v_ram segment
                        C               assume  ds:v_ram
                        C
                        C  ;    Inhibit the timer interrupt
                        C
F36C  FA                C       cli
F36D  E4 21             C       in      al,pic_1        ; 8259 mask register
F36F  50                C       push    ax                      ; save state of pic mask register
F370  0C 01             C       or      al,01h                  ; mask IRQ0
F372  E6 21             C       out     pic_1,al
F374  FB                C       sti
                        C
                        C  ;    Wait for the end of horizontal retrace
                        C
F375  E8 F40C R         C       call    v_sync
                        C
                        C  ;    Loop for each row to scroll or clear
```

```
                        C
F378                    C  v_rows:
F378  51                C       push    cx                      ; save the column counter on the stack
                        C
                        C  ;      Waste some time to be sure retrace is ended
                        C
F379                    C  v_cols:
F379  51                C       push    cx                      ;This section of code is
F37A  B1 02             C       mov     cl,2                    ; necessary for the
F37C  E2 FE             C  v_c2:loop    v_c2                    ;  Hercules Graphics Card
F37E  59                C       pop     cx
                        C
                        C  ;      If clearing rows
                        C
F37F  FF E3             C       jmp     bx                      ; select case (v_0, v_1 or v_2)
                        C
                        C  ;        Loop for all columns of the current row
                        C  ;          Wait for the beginning of horizontal retrace
                        C
F381                    C  v_0:
F381  EC                C       in      al,dx
F382  D0 D8             C       rcr     al,1
F384  73 FB             C       jnc     v_0
                        C
                        C  ;          Clear one word of the current row
                        C
F386  96                C       xchg    ax,si                   ; ax <- attrib:space
F387  AB                C       stosw
                        C
                        C  ;          If there are more columns to clear in the current row
                        C
F388  49                C       dec     cx
F389  74 03             C       jz      v_01
                        C
                        C  ;            Clear a second word
                        C
F38B  AB                C       stosw
F38C  EB 01             C       jmp     short v_02
F38E                    C  v_01:
F38E  41                C       inc     cx                      ; adjust the column counter
F38F                    C  v_02:
F38F  96                C       xchg    ax,si                   ; restore ah = row counter
                        C
                        C  ;        Repeat for all columns of the current row
                        C
F390  E2 E7             C       loop    v_cols
F392  EB 4A             C       jmp     short v_31
                        C
                        C  ;      Else scrolling rows
                        C  ;        Loop for all columns in the current row
                        C  ;          If only one word can be moved at each horizontal retrace
                        C  ;            Wait for the beginning of horizontal retrace
                        C
F394  00E0              C       dw      V_KSCROLL1              ; # of words to move at vert. retrace
F396                    C  v_1:
```

```
F396  EC              C       in      al,dx
F397  D0 D8           C       rcr     al,1
F399  73 FB           C       jnc     v_1
                      C
                      C  ;            Move one word of the current row
                      C
F39B  A5              C       movsw
                      C
                      C  ;            If vertical retrace has started
                      C  ;              If scrolling the full screen width
                      C  ;                Move a fixed number of words or finish the scroll
                      C
F39C  EB 0E           C       jmp     short v_v
                      C
                      C  ;          Else two words can be moved at each horizontal retrace
                      C  ;            If there is only one column remaining in this row
                      C
F39E  0162            C       dw      V_KSCROLL2              ; # of words to move at vert. retrace
F3A0                  C  v_2:
F3A0  E2 03           C       loop    v_21
                      C
                      C  ;              Move the last word of the current row
                      C
F3A2  41              C       inc     cx                      ; adjust the column counter
F3A3  EB F1           C       jmp     v_1
                      C
                      C  ;            Else
                      C  ;              Wait for the beginning of horizontal retrace
                      C
F3A5                  C  v_21:
F3A5  EC              C       in      al,dx
F3A6  D0 D8           C       rcr     al,1
F3A8  73 FB           C       jnc     v_21
                      C
                      C  ;              Move two words of the current row
                      C
F3AA  A5              C       movsw
F3AB  A5              C       movsw
                      C
                      C  ;              If vertical retrace has started
                      C
F3AC                  C  v_v:
F3AC  A8 04           C       test    al,04h                  ; has vertical retrace started ?
F3AE  74 2A           C       jz      v_22                    ; jump if not
                      C
                      C  ;                 If scrolling the full screen width
                      C
F3B0  0B ED           C       or      bp,bp
F3B2  75 26           C       jnz     v_22
                      C
                      C  ;                      Move a fixed number of words or finish the scroll
                      C
F3B4  8B EA           C       mov     bp,dx                   ; save the video status reg. addr
F3B6  5A              C       pop     dx                      ; dx <- number of columns per row
F3B7  52              C       push    dx                      ; save it for the next row pass
```

```
F3B8  49              C       dec     cx                      ; cx = cols remaining to move, this row
F3B9  8A C4           C       mov     al,ah                   ; al <- remaining row count
F3BB  FE C8           C       dec     al                      ; don't want to consider the curr. row
F3BD  F6 E2           C       mul     dl                      ; ax <- cols remaining on subseq. rows
F3BF  03 C1           C       add     ax,cx                   ; ax <- total columns remaining to move
F3C1  2E: 8B 4F FE    C       mov     cx,cs:[bx-2]            ; set up to move V_KSCROLLn words
F3C5  2B C1           C       sub     ax,cx                   ; ax <- cols left after move V_KSCROLLn
                      C                                       ; fewer than V_KSCROLLn remaining ?
F3C7  72 1E           C       jb      v_v2                    ; jump if so, exit the row loop
F3C9  F3/ A5          C       rep movsw               ; move V_KSCROLLn words
F3CB  F6 F2           C       div     dl                      ; al <- full rows remaining to be moved
                      C                                       ; ah <- columns left to move, curr. row
F3CD  8A CC           C       mov     cl,ah                   ; restore column counter in cx
F3CF  41              C       inc     cx                      ; adjust column counter
F3D0  40              C       inc     ax                      ; adjust row counter
F3D1  8A E0           C       mov     ah,al                   ; restore row counter in ah
F3D3  8B D5           C       mov     dx,bp                   ; restore video status reg addr
F3D5  33 ED           C       xor     bp,bp                   ; restore bp register
F3D7  E8 F40C R       C       call    v_sync                  ; wait for the end of hor. retrace
                      C
                      C ;       Repeat for all columns in the current row
                      C
F3DA                  C v_22:
F3DA  E2 9D           C       loop    v_cols
                      C
                      C ;   Repeat while more rows remain to be scrolled
                      C
F3DC                  C v_3:
F3DC  03 F5           C       add     si,bp                   ; advance source pointer to next row
F3DE                  C v_31:
F3DE  03 FD           C       add     di,bp                   ; advance dest. pointer to next row
F3E0  59              C       pop     cx                      ; reinitialize the column counter
F3E1  FE CC           C       dec     ah                      ; decrement the row counter
F3E3  74 09           C       jz      v_4                     ; jump if all rows finished
F3E5  EB 91           C       jmp     v_rows                  ; continue with the next row
                      C
                      C ;   Move any remaining words during the current vertical retrace interval
                      C
F3E7                  C v_v2:
F3E7  03 C8           C       add     cx,ax
F3E9  F3/ A5          C       rep movsw               ; move the remaining words
F3EB  59              C       pop     cx                      ; reinitialize the column counter
F3EC  33 ED           C       xor     bp,bp                   ; restore bp register
                      C
                      C ;   Restore the sate of the timer interrupt and return
                      C
F3EE                  C v_4:
F3EE  FA              C       cli
F3EF  58              C       pop     ax                      ;retrieve state of 8259 pic mask reg.
F3F0  8A E0           C       mov     ah,al
F3F2  80 E4 01        C       and     ah,01h                  ;save only IRQ0 state
F3F5  E4 21           C       in      al,pic_1        ; 8259 mask register
F3F7  24 FE           C       and     al,not 01h              ; unmask IRQ0
F3F9  0A C4           C       or      al,ah                   ;restore original IRQ0 (Timer Interrupt)
F3FB  E6 21           C       out     pic_1,al
```

```
F3FD  FB                C       sti
F3FE  1F                C       pop     ds                      ; restore ds
                        C               assume  ds:data
F3FF  5A                C       pop     dx                      ; restore dl for subsequent "clear" call
F400  C3                C       ret
                        C
                        C ;     Latch and read counter 0 of the 8253
                        C
F401                    C v_8253:
F401  B0 00             C       mov     al,0                    ; "latch counter" command
F403  E6 43             C       out     43h,al                  ; latch 8253 counter 0
F405  E4 40             C       in      al,40h                  ; lsb of latched count
F407  8A D8             C       mov     bl,al                   ; save it for comparison
F409  E4 40             C       in      al,40h                  ; msb of latched count
F40B  C3                C       ret
                        C
                        C ;     Wait for the end of horizontal retrace
                        C
F40C                    C v_sync:
F40C  EC                C       in      al,dx
F40D  D0 D8             C       rcr     al,1
F40F  73 FB             C       jnc     v_sync
F411                    C v_sync2:
F411  EC                C       in      al,dx
F412  D0 D8             C       rcr     al,1
F414  72 FB             C       jc      v_sync2
F416  C3                C       ret
                        C v_scrl_up     endp
                        C
                        C ;-----------------------------------------------------------------------
                        C ;     Scroll Active Page Down             ah = 07h
                        C ;
                        C ;     Input:  if al = 0, then clear entire window with attribute in bh
                        C ;             else,   al = number of rows to 'scroll' down
                        C ;                        = number of rows to clear at top of window
                        C ;             bh      = attribute to be used on blank row(s)
                        C ;             (ch,cl) = (row,col) of upper left corner of window from (0,0)
                        C ;             (dh,dl) = (row,col) of lower right corner of window from (0,0)
                        C ;     Output: ah      = attribute to be used on blank row(s)
                        C ;             if v_mode = 7,    al = 20h = space
                        C ;             else                      al = v_3x8
                        C ;
                        C ;     Assume: (contents of v_base6845)+6  = status register
                        C ;
                        C ;     Trash:  bp, si, & di destroyed. (bx thru dx destroyed if ROM stack)
                        C ;-----------------------------------------------------------------------
                        C
F417                    C v_scrl_dn     proc    near
                        C
                        C               assume  cs:code, ds:data, es:v_ram, ss:nothing
                        C
F417  FD                C       std                             ; NOTE: scroll down everything backwards
                        C
F418  E8 F5F4 R         C       call    v_txt_md        ; all registers preserved
F41B  72 03             C       jb      v_txt_dn
```

```
F41D  E9 D738 R           C       jmp     grf_graphics_down ; jump if graphics
F420                      C  v_txt_dn:
                          C
F420  52                  C       push    dx                      ; save registers
F421  51                  C       push    cx
F422  53                  C       push    bx
                          C
F423  8A D8               C       mov     bl,al                   ; save line count
F425  8B C2               C       mov     ax,dx                   ; pass lower right coordinates....
F427  E8 F596 R           C       call    v_scrl_pos              ; to common scroll positioning routine
F42A  74 07               C       jz      v_clr_top         ; clear rows if nothing to move
                          C
                          C  ; Scroll cl rows down.
                          C
F42C                      C  v_mv_dn:
F42C  2B F0               C       sub     si,ax                   ; subtract (bytes/row)*(rows to scroll)
                          C                                       ; from 'from' address for scroll
F42E  F7 DD               C       neg     bp                      ; negate number of bytes to skip per row
F430  E9 F2E9 R           C       jmp     v_mv                    ; now identical to scroll up!
                          C
                          C  ; Clear bl rows above.
                          C
F433                      C  v_clr_top:
F433  F7 DD               C       neg     bp                      ; negate number of bytes to skip per row
F435  E9 F32B R           C       jmp     v_clr                   ; now identical to v_clr_bot!
                          C
                          C  v_scrl_dn    endp
                          C
                          C  ;---------------------------------------------------------------------
                          C  ;    Read Attribute & Character at Cursorah = 08h
                          C  ;
                          C  ;    Input:  bh      = current active display page (0-7)
                          C  ;    Output: al      = character read
                          C  ;            ah      = attribute of character read
                          C  ;
                          C  ;    Assume: (contents of v_base6845)+6  = status register
                          C  ;
11472                        C  ;       Trash:  si & di destroyed. (si = dx: di = bx)
                          C  ;---------------------------------------------------------------------
                          C
F438                      C  v_racproc    near
                          C
                          C       assume  cs:code, ds:data, es:v_ram, ss:nothing
                          C
F438  E8 F5F4 R           C       call    v_txt_md          ; all registers preserved
F43B  72 03               C       jb      v_txt_rac
F43D  E9 D79B R           C       jmp     grf_graphics_read ; jump if graphics
F440                      C  v_txt_rac:
                          C
F440  8B FB               C       mov     di,bx                   ; save bx
F442  E8 F5C5 R           C       call    v_fpos                  ; ax & si destroyed.
                          C                                       ; bx = offset into current page
                          C
F445  8B F2               C       mov     si,dx                           ; save dx
F447  BA 0006             C       mov     dx,6                            ; get 6845 status reg. offset
```

```
F44A  03 16 0063 R        C       add     dx,word ptr ds:[v_base6845]; add 6845 pointer
                          C
                          C ; Wait for horizontal retrace... we can't read the screen during a trace
                          C ; without disturbing the screen image.
                          C
                          C
F44E                      C v_rac_inline:                         ; make sure we're in a scanline
F44E  EC                  C       in      al,dx                   ; get horiz. retrace blanking status
F44F  D0 D8               C       rcr     al,1                    ; test for horiz. retrace
F451  72 FB               C       jc      v_rac_inline            ; wait for display enable low (cf)
F453  FA                  C       cli                             ; disable ints FIRST
                          C
                          C                                       ; wait for blanking:
F454                      C v_rac_inblank:
F454  EC                  C       in      al,dx                   ; get retrace blanking status
F455  D0 D8               C       rcr     al,1                    ; test display enable (bit #0)
F457  73 FB               C       jnc     v_rac_inblank           ; try again if still in scanline.
                          C
F459  26: 8B 07           C       mov     ax,es:[bx]              ; char. and attr.  now in AX
F45C  FB                  C       sti                             ; enable interrupts immediately
                          C
F45D  8B D6               C       mov     dx,si                   ; restore dx
F45F  8B DF               C       mov     bx,di                   ; restore bx
F461  C3                  C       ret
                          C
                          C v_racendp
                          C
                          C ;--------------------------------------------------------------------
                          C ;   Write Attribute & Character at Cursor        ah = 09h
                          C ;
                          C ;   Input:  al      = character to write
                          C ;           bh      = current active display page (0-7)
                          C ;           bl      = attribute of character to write
                          C ;           cx      = counter of characters to write
                          C ;           bp      = value to return in ax
                          C ;   Output: al      = character to write
                          C ;           ah      = attribute of character to write
                          C ;
                          C ;   Assume: (contents of v_base6845)+6  = status register
                          C ;
                          C ;   Trash:  bp, si & di destroyed. (si = cx; dx if ROM stack)
                          C ;--------------------------------------------------------------------
                          C
F462                      C v_wacproc     near
                          C
                          C       assume  cs:code, ds:data, es:v_ram, ss:nothing
                          C
F462  E8 F5F4 R           C       call    v_txt_md        ; all registers preserved
F465  72 03               C       jb      v_txt_wac
F467  E9 D897 R           C       jmp     grf_graphics_write; jump if graphics
F46A                      C v_txt_wac:
                          C
F46A  8B FB               C       mov     di,bx                   ; save bx
                          C
F46C  E8 F5C5 R           C       call    v_fpos                  ; ax & si destroyed.
```

```
                        C                                       ; bx = offset into current page
F46F  87 DF             C       xchg    bx,di                   ; restore bx; di = transfer offset
                        C
F471  8B C5             C       mov     ax,bp                   ; restore ax
F473  8A E3             C       mov     ah,bl                   ; transfer attribute byte to ah
F475  8B E8             C       mov     bp,ax                   ; save attribute & character in bp
                        C
F477  8B F1             C       mov     si,cx                           ; save cx
F479  52                C       push    dx                              ; save dx
F47A  8B 16 0063 R      C       mov     dx,word ptr ds:[v_base6845]; get 6845 pointer register
F47E  83 C2 06          C       add     dx,6                            ; get 6845 status register
                        C
                        C ; Wait for horizontal retrace blank interval ...
                        C
                        C
F481                    C v_wac_hi:                             ; wait till we're in a scanline..
F481  EC                C       in      al,dx                   ; get CRT status
F482  D0 D8             C       rcr     al,1                    ; test display enable (bit #0)
F484  72 FB             C       jc      v_wac_hi        ; wait for display enable low (cf)
F486  FA                C       cli                             ; disable ints FIRST
                        C
F487                    C v_wac_lo:                             ; now wait till start of blanking..
F487  EC                C       in      al,dx                   ; 08    get CRT status
F488  D0 D8             C       rcr     al,1                    ; 02    test display enable (bit #0)
F48A  73 FB             C       jnc     v_wac_lo                ; 16/04 wait for display enable hi (cf)
                        C
F48C  8B C5             C       mov     ax,bp                   ; 02    restore ax
                        C ;stosw                                ; 11    es:di gets ax (attribute & char)
                        C
                        C ;dec cx                       ; 02
                        C                                       ; we can do 2 words in 10 us
                        C ;jz       v_wac_end           ; 04/16
                        C
F48E  AB                C       stosw                           ; 11    es:di gets ax (attribute & char)
                        C
                        C ; Worst case: (8+2+16)+(8+2+4)+(2+11)+(2+4+11) = 70 cycles = 87.5% of 80 cycles
                        C
F48F  FB                C       sti                             ; enable interrupts immediately
F490  E2 EF             C       loop    v_wac_hi        ; do it cx times
                        C
F492                    C v_wac_end:
F492  FB                C       sti                             ; enable interrupts immediately
                        C
F493  5A                C       pop     dx                      ; restore dx
F494  8B CE             C       mov     cx,si                   ; restore cx
F496  C3                C       ret
                        C
                        C v_wacendp
                        C
                        C ;---------------------------------------------------------------------
                        C ;     Write Character at Cursor Position  ah = 0Ah
                        C ;
                        C ;     Input:  al      = character to write
                        C ;             bh      = current active display page (0-7)
                        C ;             cx      = counter of characters to write
```

```
                        C  ;            bp      = value to return in ax
                        C  ;    Output: al      = character to write
                        C  ;            ah      = top byte of offset of character in page 0
                        C  ;
                        C  ;    Assume: (contents of v_base6845)+6  = status register
                        C  ;
                        C  ;    Trash:  si & di destroyed. (si = cx; dx if ROM stack)
                        C  ;-----------------------------------------------------------------------
                        C
F497                    C  v_wc proc    near
                        C
                        C       assume  cs:code, ds:data, es:v_ram, ss:nothing
                        C
F497  E8 F5F4 R         C       call    v_txt_md        ; all registers preserved
F49A  72 03             C       jb      v_txt_wc
F49C  E9 D897 R         C       jmp     grf_graphics_write; jump if graphics
F49F                    C  v_txt_wc:
                        C
F49F  8B FB             C       mov     di,bx                   ; save bx
                        C
F4A1  E8 F5C5 R         C       call    v_fpos                  ; si destroyed.
                        C                                       ; ax = offset into page 0
                        C                                       ; bx = offset into current page
F4A4  87 DF             C       xchg    bx,di                   ; restore bx; di = transfer offset
                        C
                        C
F4A6  8B F1             C       mov     si,cx                           ; save cx
F4A8  52                C       push    dx                              ; save dx
                        C
F4A9  8B D5             C       mov     dx,bp                           ; retrieve character in dl
F4AB  8A C2             C       mov     al,dl                           ; get char in al (ah = attr.)
F4AD  8B E8             C       mov     bp,ax                           ; bp = (attr. char)
                        C
F4AF  BA 0006           C       mov     dx,6                            ; get 6845 status register
F4B2  03 16 0063 R      C       add     dx,word ptr ds:[v_base6845]; get 6845 pointer register
                        C
                        C  ; Wait for horizontal retrace...
                        C
F4B6                    C  v_wc_next:
                        C
F4B6                    C  v_wc_hi:
F4B6  EC                C       in      al,dx                   ; get CRT status
F4B7  D0 D8             C       rcr     al,1                    ; test display enable (bit #0)
F4B9  72 FB             C       jc      v_wc_hi                 ; wait for display enable low (cf)
F4BB  FA                C       cli                             ; disable ints FIRST
                        C
F4BC                    C  v_wc_lo:
F4BC  EC                C       in      al,dx                   ; 08    get CRT status
F4BD  D0 D8             C       rcr     al,1                    ; 02    test display enable (bit #0)
F4BF  73 FB             C       jnc     v_wc_lo                 ; 16/04 wait for display enable hi (cf)
                        C
F4C1  8B C5             C       mov     ax,bp                   ; 02    restore ax = (attr. char)
F4C3  AA                C       stosb                           ; 11    es:di gets al (character)
                        C
F4C4  49                C       dec     cx                      ; 02
```

```
                        C                                       ;we can do 2 bytes in 10 us
F4C5  74 06             C       jz      v_wc_end                ; 04/16
F4C7  47                C       inc     di                      ; 02    skip past attribute byte
                        C
F4C8  AA                C       stosb                           ; 11    es:di gets al (character)
                        C
                        C  ; Worst case: (8+2+16)+(8+2+4)+(2+11)+(2+4+2+11) = 72 cycles = 90% of 80 cycles
                        C
F4C9  FB                C       sti                             ; enable interrupts immediately
F4CA  47                C       inc     di                      ; skip past attribute byte
F4CB  E2 E9             C       loop    v_wc_next       ; do it cx times
                        C
F4CD                    C  v_wc_end:
F4CD  FB                C       sti                             ; enable interrupts immediately
                        C
F4CE  5A                C       pop     dx                      ; restore dx
F4CF  8B CE             C       mov     cx,si                   ; restore cx
F4D1  C3                C       ret
                        C
                        C  v_wc endp
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;    Set Overscan, Back, & Foreground Colors     ah = 0Bh
                        C  ;
                        C  ;    Input:  bh      = palette color ID to set (0-127)
                        C  ;            bl      = color value to be used with that color ID
                        C  ;    Output: ah      = v_mode
                        C  ;            al      = new v_colorpal
                        C  ;
                        C  ;    Assume: (contents of v_base6845)+5  = overscan register
                        C  ;
                        C  ;    Trash:  si & di destroyed. (si = bx; di = dx)
                        C  ;---------------------------------------------------------------------
                        C
F4D2                    C  v_colproc    near
                        C
                        C       assume  cs:code, ds:data, es:v_ram, ss:nothing
                        C
F4D2  A0 0066 R         C       mov     al,byte ptr ds:[v_colorpal]; get current palette
                        C
F4D5  8B FA             C       mov     di,dx                           ; save dx
F4D7  8B F3             C       mov     si,bx                           ; save bx
                        C
F4D9  0A FF             C       or      bh,bh                           ; palette color ID = 0?
F4DB  74 0A             C       jz      v_col_0                         ; handle color ID 0
                        C
F4DD  24 DF             C       and     al,0DFh                         ; clear palette select bit #5
F4DF  D0 DB             C       rcr     bl,1                            ; test new color (bit #0)
F4E1  73 0B             C       jnb     v_col_1                         ; if bit #0 set, all done
F4E3  0C 20             C       or      al,20h                          ; else set palette select bit #5
F4E5  EB 07             C       jmp     short v_col_1
                        C
F4E7                    C  v_col_0:
F4E7  80 E3 1F          C       and     bl,01Fh                         ; save bits #0-4 of new color
F4EA  24 E0             C       and     al,0E0h                         ; clear bits #0-4 of old color
```

```
F4EC  0A C3              C       or      al,bl                           ; and combine the two.
                         C
F4EE                     C  v_col_1:
F4EE  BA 0005            C       mov     dx,5                            ; get 6845 overscan reg. offset
F4F1  03 16 0063 R       C       add     dx,word ptr ds:[v_base6845]; add 6845 pointer register
F4F5  EE                 C       out     dx,al                           ; output selection
                         C
F4F6  8B DE              C       mov     bx,si                           ; restore bx
F4F8  8B D7              C       mov     dx,di                           ; restore dx
                         C
F4FA  A2 0066 R          C       mov     byte ptr ds:[v_colorpal],al; save the value for later
F4FD  C3                 C       ret
                         C
                         C  v_colendp
                         C
                         C  ;--------------------------------------------------------------------
                         C  ;    Write Dot (see graph.src)  ah = 0Ch
                         C  ;--------------------------------------------------------------------
                         C
                         C  ;--------------------------------------------------------------------
                         C  ;    Read Dot (see graph.src)   ah = 0Dh
                         C  ;--------------------------------------------------------------------
                         C
                         C  ;--------------------------------------------------------------------
                         C  ;    Terminal Emulator to active page    ah = 0Eh
                         C  ;
                         C  ;    Input:  al      = character to write
                         C  ;            bl      = foreground color in graphics mode
                         C  ;            bp      = value to return in ax
                         C  ;    Output: All registers saved.
                         C  ;
                         C  ;    Trash:  si & di destroyed. (si = cx; di = bx; dx if ROM stack)
                         C  ;--------------------------------------------------------------------
                         C
F4FE                     C  v_terminal proc near
                         C
                         C       assume  cs:code, ds:data, es:v_ram, ss:nothing
                         C
F4FE  3C 07              C       cmp     al,BEL                          ; is it bell character?
F500  75 03              C       jne     v_term_nobell
                         C
F502  E9 F606 R          C       jmp     v_bell
                         C
F505                     C  v_term_nobell:
F505  52                 C       push    dx                              ; save dx
F506  8B F1              C       mov     si,cx                           ; save cx
F508  8B FB              C       mov     di,bx                           ; save bx
                         C
                         C  ; Get cursor position in active page.
                         C
F50A  B7 07              C       mov     bh,07h                          ;mask for page number, MOD 8
F50C  22 3E 0062 R       C       and     bh,byte ptr ds:[v_apage]    ; get active page number (0-7)
                         C
F510  B4 03              C       mov     ah,03h                          ; call v_r_curs_pos
F512  CD 10              C       INT     10h                             ; (dh,dl) = (row,col) of cursor
```

```
                        C                                               ; (ch,cl) = cursor mode setting
                        C
F514  8B C5             C       mov     ax,bp                           ; restore ax
F516  B9 0001           C       mov     cx,1                            ; character count for write char
F519  B4 0A             C       mov     ah,0Ah                          ; function code for write char
                        C
                        C  ; Handle special cases: dx has (row,col) of current cursor position.
                        C
F51B  3C 0A             C       cmp     al,LF                           ; is it a line feed?
F51D  74 14             C       je      v_lf
F51F  3C 0D             C       cmp     al,CR                           ; is it a carriage return?
F521  74 60             C       je      v_cr
F523  3C 08             C       cmp     al,BS                           ; is it a backspace?
F525  74 54             C       je      v_bs
                        C
                        C  ; Normal Case: write the character
                        C
F527  CD 10             C       INT     10h                             ; to write the character
                        C
F529  FE C2             C       inc     dl                              ; increment the column
F52B  3A 16 004A R      C       cmp     dl,byte ptr ds:[v_width]    ; column overflow?
F52F  72 13             C       jb      v_set_new_cur                   ; set new cursor position
                        C
F531  32 D2             C       xor     dl,dl                           ; carriage return cursor
                        C
F533  80 3E 0049 R 48   C  v_lf:cmp     byte ptr ds:[v_mode],72     ; is this mode 72 ?
F538  B4 31             C       mov     ah,49                           ; mode 72 has 50 rows
F53A  74 02             C       je      v_lrow                          ; jump if mode 72
F53C  B4 18             C       mov     ah,24                           ; modes 4,5,6,64 have 25 rows
F53E  3A F4             C  v_lrow: cmp   dh,ah                           ; are we at last row yet?
F540  74 07             C       je      v_scrl_tty                      ; if yes,  go scroll the screen
F542  FE C6             C       inc     dh                              ; otherwise, inc to next row
F544                    C  v_set_new_cur:
F544  B4 02             C       mov     ah,02h                          ; call v_curs_pos to set new
F546  EB 29 90          C       jmp     v_term_ret                      ; cursor position
                        C
F549                    C  v_scrl_tty:                              ; (dh,dl) = (row,col) = (24,0) or (49,0)
F549  B4 02             C       mov     ah,02h                          ; call v_curs_pos to set cursor
F54B  CD 10             C       INT     10h                             ; and so that we can read back
                        C                                               ; the proper attribute byte
                        C
F54D  32 E4             C       xor     ah,ah                           ; ah = 0 for graphics
F54F  E8 F5F4 R         C       call    v_txt_md                    ; are we text mode?
F552  73 04             C       jnb     v_scrl_tty_graphics             ; jump if graphics
                        C
F554  B4 08             C       mov     ah,08h                          ; call v_rac
F556  CD 10             C       INT     10h                             ; to get attribute byte in ah
                        C
F558                    C  v_scrl_tty_graphics:
F558  33 C9             C       xor     cx,cx                           ; (ch,cl)= upper left (row,col)
                        C                                               ;        = (0,0)
F55A  8A FC             C       mov     bh,ah                           ; store attribute in bh
F55C  B8 0601           C       mov     ax,0601h                    ; call v_scrl_up to scroll
                        C                                               ; one line with attribute bh
F55F  8A 16 004A R      C       mov     dl,byte ptr ds:[v_width]    ; (dh,dl)= lower right (row,col)
```

```
F563  80 EA 01            C       sub     dl,1                            ; column = v_width-1
F566  80 3E 0049 R 48     C       cmp     byte ptr ds:[v_mode],72    ; is this mode 72 ?
F56B  B6 31               C       mov     dh,49                           ; if yes then row = 49
F56D  74 02               C       je      v_term_ret                      ; jump if mode = 72
F56F  B6 18               C       mov     dh,24                           ; if not mode 72 then row = 24
                          C
F571                      C  v_term_ret:
F571  CD 10               C       INT     10h
F573                      C  v_term_nop:
                          C
                          C  ; Clean up.
                          C
F573  8B C5               C       mov     ax,bp                           ; restore ax
F575  8B DF               C       mov     bx,di                           ; restore bx
F577  8B CE               C       mov     cx,si                           ; restore cx
F579  5A                  C       pop     dx                              ; restore dx
F57A  C3                  C       ret
                          C
                          C
F57B  0A D2               C  v_bs:or      dl,dl                           ; back space -- column = 0 ?
F57D  74 F4               C       jz      v_term_nop                      ; don't change cursor position
F57F  FE CA               C       dec     dl
F581  EB C1               C       jmp     v_set_new_cur
                          C
                          C
F583  32 D2               C  v_cr:xor     dl,dl                           ; carriage return
F585  EB BD               C       jmp     v_set_new_cur
                          C
                          C  v_terminal endp
                          C
                          C  ;--------------------------------------------------------------------
                          C  ;    Read Current Video Status  ah = 0Fh
                          C  ;
                          C  ;    Input:   None.
                          C  ;    Output:  ah       = number of character columns on screen
                          C  ;             al       = display mode currently set
                          C  ;             bh       = current active display page (0-7)
                          C  ;
                          C  ;    Trash:   None.
                          C  ;--------------------------------------------------------------------
                          C
F587                      C  v_stat       proc    near
                          C
                          C       assume  cs:code, ds:data, es:v_ram, ss:nothing
                          C
F587  8A 26 004A R        C       mov     ah,byte ptr ds:[v_width]
F58B  A0 0049 R           C       mov     al,byte ptr ds:[v_mode]
F58E  8A 3E 0062 R        C       mov     bh,byte ptr ds:[v_apage]
F592  80 E7 07            C       and     bh,07h                          ; page number mod 8
F595  C3                  C       ret
                          C
                          C  v_stat       endp
                          C       page
                          C  ;====================================================================
                          C
```

```
C  ;--------------------------------------------------------------------
C  ;    Common scroll positioning and register initialization routine.
C  ;
C  ;    Input:   (ah,al)  = starting (row,col) position from (0,0)
C  ;                      = (ch,cl) for up / (dh,dl) for down
C  ;             bh       = attribute to be used on blank row(s)
C  ;             if bl = 0, then clear entire window with attribute in bh
C  ;             else,    bl = number of rows to 'scroll' up / down
C  ;                         = number of rows to clear at top / bottom of window
C  ;             (ch,cl)  = (row,col) of upper left corner of window from (0,0)
C  ;             (dh,dl)  = (row,col) of lower right corner of window from (0,0)
C  ;
C  ;    es:      +-----------------------------------------------+
C  ;             |    (ch,cl)                                    |
C  ;             |        +-------------------------------+      |
C  ;             |        |                               |      |
C  ;             |        |                               |      |
C  ;             |        |                               |      |
C  ;             |     ^  +-------------------------------+      |
C  ;             |  bl |  |                               |      |
C  ;             |     v  +-------------------------------+      |
C  ;             |                                  (dh,dl)      |
C  ;             +-----------------------------------------------+
C  ;
C  ;    Output:  zf       = state of bl at entry (nz if scroll; z if clear only)
C  ;             ax       = (number of bytes/row) * (number of rows to scroll)
C  ;                      = (2 * bl * v_width)
C  ;             bh       = attribute to be used on blank line(s)
C  ;             if bl = 0 at entry, then clear entire window with bl attribute
C  ;                      bl = dh = window height to clear
C  ;             else,    bl = number of rows to 'scroll' up / down
C  ;                         = number of rows to clear at top / bottom of window
C  ;             ch       = zero
C  ;             cl       = number of rows to move
C  ;                      = delta of upper and lower coordinates-bl = (dh-bl)
C  ;             (dh,dl)  = delta of upper left and lower right coordinates
C  ;                      = window height and width
C  ;             bp       = number of bytes NOT to move per row (2*(v_width -dl))
C  ;             si       = 'from' address for scroll
C  ;             di       = 'to' address for scroll
C  ;             ds       = data_seg (preserved)
C  ;             es       = para_mono
C  ;
C  ;                      <-- bp ->                    <-- bp ->
C  ;    es:      +-----------------------------------------------+
C  ;             | si, di offset if up                           |
C  ;             |        +-------------------------------+ ^ |
C  ;             |        |<------------ dl ------------->| | |
C  ;             |        |                               | | |
C  ;             |        |                               | | dh |
C  ;             |     ^  +-------------------------------+ | |
C  ;             |  bl |  |                               | | |
C  ;             |     v  +-------------------------------+ v |
C  ;             |                          si, di offset if down |
C  ;             +-----------------------------------------------+
```

```
                        C  ;
                        C  ;--------------------------------------------------------------------------
                        C       page
F596                    C  v_scrl_pos   proc    near
                        C
                        C               assume  cs:code, ds:data, es:v_ram, ss:nothing
                        C
F596  E8 F5E3 R         C       call    v_posn                          ; (ah,al) -> ax offset; si trash
                        C
F599  2A F5             C       sub     dh,ch                           ; (dh,dl) gets delta (drow,dcol)
F59B  FE C6             C       inc     dh                              ; dh = number of rows
F59D  2A D1             C       sub     dl,cl
F59F  FE C2             C       inc     dl                              ; dl = number of columns
                        C
F5A1  8B 36 004E R      C       mov     si,word ptr ds:[v_top]      ; get offset of active page
F5A5  03 F0             C       add     si,ax                           ;  add offset in page => 'from'
F5A7  8B FE             C       mov     di,si                           ;  , 'to' addresses.
                        C
                        C
F5A9  33 C9             C       xor     cx,cx                           ; init count register to zero.
F5AB  8A CA             C       mov     cl,dl                           ; cx = number of columns = dl
F5AD  A1 004A R         C       mov     ax,word ptr ds:[v_width]    ; get screen width
F5B0  8B E8             C       mov     bp,ax                           ; bp = v_width
F5B2  2B E9             C       sub     bp,cx                           ; bp = (v_width - dl)
F5B4  D1 E5             C       shl     bp,1                            ; bp = 2 * (v_width - dl)
                        C
F5B6  D0 E0             C       shl     al,1                            ; al = 2 * v_width
F5B8  F6 E3             C       mul     bl                              ; ax = 2*v_width*no. of rows
                        C
F5BA  8A CE             C       mov     cl,dh
F5BC  2A CB             C       sub     cl,bl                           ; cl <= number of rows to move
                        C
F5BE  0A DB             C       or      bl,bl                           ; if rows to scroll,
F5C0  75 02             C       jnz     v_scrl_mv_and_clr ; then, move & clear row, return nz.
                        C
F5C2  8A DE             C       mov     bl,dh                       ; else clear dh rows only, return z.
                        C
F5C4                    C  v_scrl_mv_and_clr:
F5C4  C3                C       ret                                     ; ZF indicates state of BL at entry
                        C
                        C  v_scrl_pos   endp
                        C       page
                        C  ;--------------------------------------------------------------------------
                        C  ;    Calculates video ram buffer offset of a character in text mode
                        C  ;
                        C  ;    Input:  bh      = current active display page (0-7)
                        C  ;    Output: bx      = offset of character in text mode at display page
                        C  ;                    = (page number)*(v_height)+offset of v_curpos(bh)
                        C  ;            ax      = offset of character in text mode from page 0
                        C  ;
                        C  ;    Trash:  si = destroyed.
                        C  ;--------------------------------------------------------------------------
                        C
F5C5                    C  v_fpos       proc    near
                        C
```

```
                        C       assume  cs:code, ds:data, es:v_ram, ss:nothing
                        C
F5C5  8A C7             C       mov     al,bh                           ; al gets page number
F5C7  33 DB             C       xor     bx,bx                           ; bx = 0
F5C9  25 0007           C       and     ax,07h                          ; ax = page number mod 8
F5CC  8B F0             C       mov     si,ax                           ; si keeps page number mod 8
F5CE  74 07             C       jz      v_fpos_0                ; page number = 0?
                        C
F5D0                    C  v_fpos_lp:
F5D0  03 1E 004C R      C       add     bx,word ptr ds:[v_height]  ; optimization: word multipli-
F5D4  48                C       dec     ax                              ; cation by less than 8 without
F5D5  75 F9             C       jnz     v_fpos_lp               ; destroying dx (or cx).
                        C
F5D7                    C  v_fpos_0:                                    ; bx = (page number)*(v_height)
F5D7  D1 E6             C       shl     si,1                            ; page number mod 8 word index
F5D9  8B 84 0050 R      C       mov     ax,word ptr ds:[si+v_curpos]
F5DD  E8 F5E3 R         C       call    v_posn                          ; (ah,al) -> ax offset; si trash
F5E0  03 D8             C       add     bx,ax                           ; bx = (page)*(v_height)+offset
F5E2  C3                C       ret
                        C
                        C  v_fpos       endp
                        C
                        C  ;---------------------------------------------------------------------------
                        C  ;    Calculates video ram buffer offset of a character in text mode
                        C  ;
                        C  ;    Input:  (ah,al)  = (row,col) position
                        C  ;    Output: ax       = offset of character in text mode.
                        C  ;
                        C  ;    Trash:  si destroyed.
                        C  ;---------------------------------------------------------------------------
                        C
F5E3                    C  v_posn               proc    near
                        C
                        C               assume  cs:code, ds:data, es:v_ram, ss:nothing
                        C
F5E3  8B F0             C       mov     si,ax
F5E5  81 E6 00FF        C       and     si,0FFh                 ; si keeps column (al)
F5E9  8A C4             C       mov     al,ah                   ; al gets row (ah)
F5EB  F6 26 004A R      C       mul     byte ptr ds:[v_width]   ; ax gets (row * v_width)
F5EF  03 C6             C       add     ax,si                   ; ax gets (row * v_width)+ column
F5F1  D1 E0             C       shl     ax,1                    ; ax gets 2*((row * v_width)+column)
F5F3  C3                C       ret
                        C
                        C  v_posn               endp
                        C
                        C
                        C  ;---------------------------------------------------------------------------
                        C  ;    Is v_mode text or graphics or black/white card?
                        C  ;
                        C  ;    Input:  None.
                        C  ;    Output: carry flag (cf) set if text.  carry flag cleared if graphics.
                        C  ;            Text Modes:     0 to 3 and 7
                        C  ;            Graphics Modes: 4 to 6, 64, and 72
                        C  ;
                        C  ;    Trash:  None.
```

```
                        C ;--------------------------------------------------------------------------
                        C
F5F4                    C v_txt_md      proc    near
                        C
                        C               assume  cs:code, ds:data, es:v_ram, ss:nothing
                        C
F5F4  80 3E 0049 R 04   C       cmp     byte ptr ds:[v_mode],4
F5F9  72 09             C       jb      v_txt_ok
                        C
F5FB  80 3E 0049 R 07   C       cmp     byte ptr ds:[v_mode],7
F600  74 02             C       je      v_txt_ok
F602  F8                C       clc                                     ; graphics mode (CF = 0)
F603  C3                C       ret                                     ; modes 4 to 6, 64, and 72
                        C
F604                    C v_txt_ok:
F604  F9                C       stc                                     ; text mode (CF = 1)
F605  C3                C       ret                                     ; modes 0 to 3 and 7
                        C
                        C v_txt_md      endp
                        C
                        C ;--------------------------------------------------------------------------
                        C ;   Handle BEL character: Beeps the speaker.
                        C ;
                        C ;   No parameters.
                        C ;
                        C ;--------------------------------------------------------------------------
                        C
F606                    C v_bell proc near
                        C
                        C               assume  cs:code, ds:data, es:v_ram, ss:nothing
                        C
F606  50                C       push    ax
                        C
F607  B0 B6             C       mov     al,t2cmd                ; p_8253_2,lsb 1st,mode 3,no BCD
F609  E6 43             C       out     p_8253_ctrl,al
F60B  B0 00             C       mov     al,00h                  ; p_timer count
F60D  E6 42             C       out     p_8253_2,al             ; least significant byte
F60F  B0 06             C       mov     al,06h
F611  E6 42             C       out     p_8253_2,al             ; most significant byte
                        C
F613  E4 61             C       in      al,p_kctrl              ; get control data
F615  8A E0             C       mov     ah,al                   ; save control status
F617  0C 03             C       or      al,03h                  ; turn speaker on
F619  E6 61             C       out     p_kctrl,al
                        C
F61B  51                C       push    cx
F61C  B9 00C8           C       mov     cx,200                  ;512 msec
F61F                    C bell_wait:
F61F  E8 EF37 R         C       call    f_wait_one_ms           ;wait for 1 ms
F622  E2 FB             C       loop    bell_wait
F624  59                C       pop     cx
                        C
F625  8A C4             C       mov     al,ah                   ; restore control status
F627  E6 61             C       out     p_kctrl,al
                        C
```

```
F629  58                C       pop     ax
F62A  C3                C       ret                                     ; return from v_term
                        C
                        C  v_bell endp
                        C
F62B                    C  code ends

                        C  include lp.asm
                        C
                        C  ;===========================================================================
                        C  ;
                        C  ;    Read Light Pen                      function code = 04h
                        C  ;
                        C  ;    Input:   ah       = video mode
                        C  ;             ds       = 40h
                        C  ;    Output: ah        = 0 light pen switch not down/not triggered
                        C  ;             ah       = 1 implies:
                        C  ;                      (dh,dl) = (row,col) of character light pen
                        C  ;                                position from (0,0)
                        C  ;                      ch       = raster line (0-199)
                        C  ;                      bx       = pixel column (0-319,0-639)
                        C  ;
                        C  ;    Destroys: SI,DX,AL,CL registers destroyed.
                        C  ;
                        C  ;---------------------------------------------------------------------------
                        C
                        C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                        C  ;v_lp_table:  Table used by grf_light_pen to control operation of the routine:
                        C  ;          There is one entry in the table for each mode, (except modes
                        C  ;          64 and 72).  Bit definitions are:
                        C  ;             Bit 7:   1 = graphic mode (modes 4,5 and 6.)
                        C  ;             Bit 6:   1 = mode 6.
                        C  ;             Bits 5-3: unused.
                        C  ;             Bits 2-0: = Adjustment value.  Subtract this value from
                        C  ;                         value return by 6845.
                        C  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                        C
F62B                    C  code segment  public   'ROM'
                        C       assume   cs:code, ds:data, es:nothing, ss:nothing
                        C
F62B                    C  lp_table      proc     near
                        C
F62B                    C  v_lp_table    label    byte
F62B  03                C       db       003h              ;mode 0
F62C  03                C       db       003h              ;mode 1
F62D  05                C       db       005h              ;mode 2
F62E  05                C       db       005h              ;mode 3
F62F  83                C       db       083h              ;mode 4
F630  83                C       db       083h              ;mode 5
F631  C3                C       db       0c3h              ;mode 6
F632  04                C       db       004h              ;mode 7
                        C
                        C  lp_table      endp
                        C
                        C
```

```
F633                    C  grf_light_pen proc     near
                        C
F633  80 FC 07          C       cmp     ah,7
F636  76 05             C       jbe     v_lp_valid_mode
F638  32 E4             C       xor     ah,ah
F63A  E9 F6CA R         C       jmp     v_lp_ret
F63D                    C  v_lp_valid_mode:
                        C  ;convert the mode value into a word, and save it in si, so
                        C  ;we can use it to look up values in the v_lp_table.
                        C
F63D  8A C4             C       mov     al,ah
F63F  32 E4             C       xor     ah,ah
F641  8B F0             C       mov     si,ax    ;save it in an index register
                        C
                        C  ;read the light pen switch value from the status register.
F643  8B 16 0063 R      C       mov     dx,word ptr ds:[v_base6845]
F647  83 C2 06          C       add     dx,6             ;point to status register
F64A  EC                C       in      al,dx
                        C
                        C  ;the light pen switch values are on bits 1 and 2.
                        C  ;  bit 1 = +switch has " triggered" (meaning that there is a value available)
                        C  ;  bit 2 = -switch held down
                        C  ;  We only want to return a value if switch held down AND a trigger has
                        C  ;  occurred.  (02h)
                        C
F64B  24 06             C       and     al,06h           ;isolate bits 1 and 2
F64D  3C 02             C       cmp     al,02h           ; 010b= switch AND trigger
F64F  74 0A             C       je      v_lp_read; go read the light pen
F651  B4 00             C       mov     ah,0             ; 0 to ah, don't change flags
                        C                                ; 0 is switch not triggered return value.
F653  7C 75             C       jl      v_lp_ret ; 000b= switch held down, but no trigger
                        C                                ; return without issuing a reset
                        C  ;Otherwise, value is AL is 100b or 110b, meaning switch is not currently
                        C  ;depressed, so reset the trigger in case its been set.
                        C
F655  42                C       inc     dx               ;point to the reset port
F656  EE                C       out     dx,al            ;the value sent is not important
F657  4A                C       dec     dx               ;leave dx pointing to status register
F658  EB 70 90          C       jmp     v_lp_ret
                        C
F65B                    C  v_lp_read:
                        C  ;read the values in R16 and R17 of the 6845.
                        C  ; R16 is the MSB, R17 is the LSB.
                        C
F65B  83 EA 06          C       sub     dx,6             ;dx now points back to 6845 index register
F65E  B9 0002           C       mov     cx,2             ;we have to read 2 registers
F661                    C  v_lp_read_lp:
F661  B0 0F             C       mov     al,15            ;
F663  02 C1             C       add     al,cl            ;on 1st pass, we'll read 15+2, or LSB
                        C                                ;on 2nd pass, we'll read 15+1, or MSB
                        C
F665  EE                C       out     dx,al            ;send value to index register
F666  EB 00             C       jmp     $+2              ;recovery time
F668  42                C       inc     dx               ;point to data register
F669  EC                C       in      al,dx            ;read the data
```

```
F66A  4A                  C       dec     dx              ;point back to index register for next pass
                          C
F66B  86 E0               C       xchg    ah,al           ;on 1st pass, we save the LSB in AH.
                          C                               ;on 2nd pass, we move the MSB to AH, LSB to AL
F66D  E2 F2               C       loop    v_lp_read_lp    ;go back and read the MSB
                          C  ;At this point:
                          C  ;    AX=raw value from 6845, a WORD offset from start of 6845 address space
                          C  ;    CX=0    this is important, because we will convert control byte to
                          C  ;            a word.
                          C
                          C  ;First, there is a slight problem with the 6845, so we'll
                          C  ;have to subtract an adjustment value so the numbers come out correctly:
                          C
F66F  2E: 8A 8C F62B R    C       mov     cl,cs:[si+v_lp_table]   ;get a copy of the control byte in cl
F674  8B D9               C       mov     bx,cx                   ;CH was 0, so bh is now also 0
                          C                                       ;so we can SUB from AX register.
F676  80 E3 07            C       and     bl,07h                  ;isolate the adjustment bits (0-2)
F679  2B C3               C       sub     ax,bx                   ;perform subtraction, bh is 0.
                          C
                          C  ;Secondly, we have to subtract the v_top, in case we aren't
                          C  ;on Page 0.  We'll have to convert the page offset from a byte
                          C  ;value into a word value before subtraction.
                          C
F67B  8B 16 004E R        C       mov     dx,word ptr ds:[v_top]      ;subtract page offset
F67F  D1 EA               C       shr     dx,1                        ;convert to word offset
F681  2B C2               C       sub     ax,dx                       ;
F683  7F 02               C       jg      v_lp_calc           ;if ax less than dx
F685  33 C0               C       xor     ax,ax                       ;set ax=0
                          C
                          C  ;graphic and alpha modes have to be treated differently from this point, so
                          C  ;we'll use the top bit of the control byte to branch
                          C
                          C  ;At this point:
                          C  ;    AX=adjusted memory value of where light pen is
                          C  ;    CL=Control byte.
                          C  ;    BH=0, CH=0
F687                      C  v_lp_calc:
F687  D0 D1               C       rcl     cl,1                    ;original copy of control byte is
                          C                                       ;still in cl.
F689  72 15               C       jc      v_lp_graf_modes
                          C
F68B  F6 36 004A R        C       div     byte ptr ds:[v_width]   ;divide by number of words on a line
                          C                                       ;(character/attribute pair is a word)
F68F  8A F0               C       mov     dh,al                   ;the quotient, or number of rows, is
                          C                                       ;in al.
F691  8A D4               C       mov     dl,ah                   ;the remainder, or column number is
                          C                                       ;in ah.
F693  8A EE               C       mov     ch,dh                   ;mov number of rows to ch.
F695  B1 03               C       mov     cl,3
F697  D2 E5               C       shl     ch,cl                   ;multiply by 8, since there are
                          C                                       ;8 raster lines per row.  This
                          C                                       ;means user will always get top
                          C                                       ;raster line
F699  8A DA               C       mov     bl,dl                   ;bh is still 0, from earlier.
F69B  D3 E3               C       shl     bx,cl                   ;multiply by 8, since there are
```

```
                        C                                       ;8 pixels across per character
                        C                                       ;This means caller will always get
                        C                                       ;leftmost pixel.
F69D  EB 1D 90          C       jmp     v_lp_read_end           ;go reset trigger and return.
F6A0                    C  v_lp_graf_modes:
                        C  ;At this point:
                        C  ;    AX=Adjusted memory value
                        C  ;    Cl=Control byte, left shifted one
                        C  ;    CH, BH=0
                        C  ; NOTE: in graphics mode, light pen always returns one of the EVEN numbered
                        C  ;    line (memory is top half of address space).
                        C
F6A0  B2 28             C       mov     dl,40                   ;all graphics modes have 40 words
F6A2  F6 F2             C       div     dl                      ;per row
                        C
F6A4  D0 D1             C       rcl     cl,1                    ;is this mode 6?
F6A6  73 02             C       jnc     v_lp_calc_graf          ;if not, jump
F6A8  D0 E4             C       shl     ah,1                    ;otherwise, double the column and pixel
                        C                                       ;counts, since there are twice as many
                        C                                       ;pixels and twice as many rows per word
F6AA                    C  v_lp_calc_graf:
F6AA  8A D4             C       mov     dl,ah                   ;put column number in dl
F6AC  8A DC             C       mov     bl,ah                   ;and in bl, (BH already = 0)
F6AE  B1 03             C       mov     cl,3
F6B0  D3 E3             C       shl     bx,cl                   ; multiply by 8, so x8 for medium
                        C                                       ; x16 for high res.
                        C
F6B2  8A E8             C       mov     ch,al                   ;quotient, or raster value to ch.
F6B4  D0 E5             C       shl     ch,1                    ;multiply by 2 because even rows
                        C                                       ;are all at start of memory
F6B6  8A F0             C       mov     dh,al                   ;mov the raster to dh
F6B8  D0 EE             C       shr     dh,1
F6BA  D0 EE             C       shr     dh,1                    ;divide by 4, since there are 4
                        C                                       ;EVEN numbered rasters per character.
                        C  .list
F6BC                    C  v_lp_read_end:
F6BC  8B C2             C       mov     ax,dx                   ;save dx
F6BE  8B 16 0063 R      C       mov     dx,word ptr ds:[v_base6845]
F6C2  83 C2 07          C       add     dx,7
F6C5  EE                C       out     dx,al
F6C6  8B D0             C       mov     dx,ax                    ;restore dx.
F6C8  B4 01             C       mov     ah,1                     ;indicate succesful read.
F6CA                    C  v_lp_ret:
F6CA  C3                C       ret
                        C
                        C
                        C  grf_light_pen endp
                        C
F6CB                    C  code ends


                        C  include comm3.asm
                        C
                        C
                        C  ;===========================================================================
```

```
                        C  ;    Filename:com3.src
                        C  ;=============================================================================
                        C
F6CB                    C  code segment  public 'ROM'
                        C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C
                        C  ;-----------------------------------------------------------------------
                        C  ;    INS8250 Asynchronous Communication Chip Baud Rate Time Constants
                        C  ;    (baud rate generator signal is 3.6864 MHz put through a
                        C  ;       divide-by-2 circuit).
                        C  ;
                        C  ;                       ((3,686,400 Hz)/2) = Input Freq.
                        C  ;    Time Constant = -------------------------------
                        C  ;                                (16)*(baud rate)
                        C  ;-----------------------------------------------------------------------
                        C
F6CB                    C  scc_init      proc     near                       ; ah = 00h
                        C                assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
F6CB  8A E8             C       mov     ch,al                          ; save input parameters.
                        C
F6CD  32 C0             C       xor      al,al
F6CF  EE                C       out      dx,al                              ; dummy read on scc_ctl_x
F6D0  E8 E8E6 R         C       call     rs_dly
F6D3  EC                C       in       al,dx                              ; insures proper register addr.
F6D4  E8 E8E6 R         C       call     rs_dly
                        C
F6D7  B0 09             C       mov      al,9                               ; select write register #9
F6D9  EE                C       out      dx,al                              ; reset and disable int
F6DA  E8 E8E6 R         C       call     rs_dly
F6DD  B0 C1             C       mov      al,11000001b                       ; hw reset, st=0, mie=dis, nv=1
F6DF  EE                C       out      dx,al
F6E0  E8 E8E6 R         C       call     rs_dly
                        C
F6E3  B0 04             C       mov      al,4                               ; select write register #4
F6E5  EE                C       out      dx,al                              ; transfer parameters
F6E6  E8 E8E6 R         C       call     rs_dly
                        C
F6E9  B0 44             C       mov      al,01000100b                       ; 16x, 8-bit synch, 1 or 2 stop
                        C
F6EB  8A E5             C       mov     ah,ch                          ; get input parameters.
F6ED  80 E4 04          C       and      ah,00000100b                       ; get bit #2
F6F0  D0 E4             C       shl      ah,1                               ; move to bit #3
F6F2  0A C4             C       or       al,ah                              ; 'or' the bit
                        C
F6F4  8A E5             C       mov     ah,ch                          ; get input parameters.
F6F6  80 E4 18          C       and      ah,00011000b                       ; get bits #3 & 4
F6F9  B1 03             C       mov      cl,3
F6FB  D2 EC             C       shr      ah,cl                              ; move to bits #0 & 1
F6FD  0A C4             C       or       al,ah                              ; 'or' the bits
                        C
F6FF  EE                C       out      dx,al                              ; all done!!!
F700  E8 E8E6 R         C       call     rs_dly
                        C
```

```
F703  B0 0B              C       mov     al,11                           ; select write register #11
F705  EE                 C       out     dx,al                           ; clock mode control
F706  E8 E8E6 R          C       call    rs_dly
F709  B0 55              C       mov     al,01010101b                    ; no x,r=br,t=br,txc o,tx ck
F70B  EE                 C       out     dx,al
F70C  E8 E8E6 R          C       call    rs_dly
                         C
F70F  B0 0C              C       mov     al,12                           ; select write register #12
F711  EE                 C       out     dx,al                           ; low byte of baud rate const
F712  E8 E8E6 R          C       call    rs_dly
                         C
F715  8A DD              C       mov     bl,ch                   ; get input parameters.
F717  81 E3 00E0         C       and     bx,11100000b                    ; get bits #5, 6, & 7 (clear bh)
F71B  B1 04              C       mov     cl,4
F71D  D2 EB              C       shr     bl,cl                           ; move to bits #1,2,& 3
                         C                                               ; bx is word index
                         C
                         C  ; NOTE: These values are the SAME as the com_baud EXCEPT for the - 2!!!!
                         C
                         C  ;    mov     ax,word ptr cs:[bx+scc_baud]    ; get 8530 baud count
                         C
F71F  2E: 8B 87 E729 R   C       mov     ax,word ptr cs:[bx+com_baud]    ; get 8250 baud count
F724  48                 C       dec     ax                              ; and subtract 2!!!!!
F725  48                 C       dec     ax
                         C
F726  EE                 C       out     dx,al                           ; output low byte of baud rate
F727  E8 E8E6 R          C       call    rs_dly
                         C
F72A  B0 0D              C       mov     al,13                           ; select write register #13
F72C  EE                 C       out     dx,al                           ; high byte of baud rate const
F72D  E8 E8E6 R          C       call    rs_dly
                         C
F730  8A C4              C       mov     al,ah                           ; output high byte of baud rate
F732  EE                 C       out     dx,al
F733  E8 E8E6 R          C       call    rs_dly
                         C
F736  B0 0E              C       mov     al,14                           ; select write register #14
F738  EE                 C       out     dx,al                           ; baud rate generator enable
F739  E8 E8E6 R          C       call    rs_dly
F73C  B0 03              C       mov     al,00000011b                    ; baud generator enable & source
F73E  EE                 C       out     dx,al
F73F  E8 E8E6 R          C       call    rs_dly
                         C
F742  B0 01              C       mov     al,1                            ; select write register #1
F744  EE                 C       out     dx,al                           ; data xfer mode definition
F745  E8 E8E6 R          C       call    rs_dly
F748  32 C0              C       xor     al,al                           ; rx dis, pty no spec, tx dis
F74A  EE                 C       out     dx,al                           ; ext dis
F74B  E8 E8E6 R          C       call    rs_dly
                         C
F74E  B0 03              C       mov     al,3                            ; select write register #3
F750  EE                 C       out     dx,al                           ; RxD parameters and control
F751  E8 E8E6 R          C       call    rs_dly
                         C
F754  B0 01              C       mov     al,00000001b                    ; RxD enable
```

```
                        C
F756  8A DD             C       mov     bl,ch                           ; get input parameters.
F758  81 E3 0003        C       and     bx,00000011b                        ; get bits #0 & 1 (clear bh)
F75C  2E: 8A A7 F786 R  C       mov     ah,byte ptr cs:[bx+scc_dbit]
F761  B1 06             C       mov     cl,6
F763  D2 E4             C       shl     ah,cl                               ; move data bits to #6 & 7
F765  0A C4             C       or      al,ah                               ; 'or' the data bits
                        C
F767  EE                C       out     dx,al
F768  E8 E8E6 R         C       call    rs_dly
                        C
F76B  B0 05             C       mov     al,5                                ; select write register #5
F76D  EE                C       out     dx,al                               ; TxD parameters and control
F76E  E8 E8E6 R         C       call    rs_dly
                        C
F771  B0 08             C       mov     al,00001000b                        ; TxD enabled. (power-up value)
F773  95                C       xchg    ax,bp                               ; get original function code
F774  0A E4             C       or      ah,ah                               ; is it 0 or FF?
F776  95                C       xchg    ax,bp                               ; restore AX
F777  75 02             C       jnz     scc_pwrup                   ; jump if original AH=0FFh
F779  B0 8A             C       mov     al,10001010b                        ; TxD,DTR,RTS enabled. (normal)
                        C
F77B                    C  scc_pwrup:
F77B  D0 EC             C       shr     ah,1                                ; move bits #6 & 7 to #5 & 6
F77D  0A C4             C       or      al,ah                               ; 'or' the bits
F77F  EE                C       out     dx,al
F780  E8 E8E6 R         C       call    rs_dly
                        C
F783  E9 E88C R         C       jmp     scc_stat                    ; return status
                        C
                        C  ;---------------------------------------------------------------------
                        C  ;    Z8530 Compatible Data-Bit Definitions for Mapping   bits
                        C  ;---------------------------------------------------------------------
                        C
F786  00                C  scc_dbit     db      00b     ;    5 data bits (0) (non- )
F787  02                C               db      10b     ;    6 data bits (1) (non- )
F788  01                C               db      01b     ;    7 data bits (2)
F789  03                C               db      11b     ;    8 data bits (3)
                        C
                        C  scc_init     endp
                        C
F78A                    C  code ends


                           ; include    hdu1.asm

                        C  include mem.asm
                        C
                        C
                        C  ;=====================================================================
                        C  ;    Filename:mem.src
                        C  ;
                        C  ;    This module includes INT 12h, 11h, & 15h.
                        C  ;
                        C  ;=====================================================================
```

```
                        C
F78A                    C  code segment  public 'ROM'
                        C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    INT 12h -- memory size detect
                        C  ;--------------------------------------------------------------------
                        C
                        C       XORG     0F841h
F841                    C1      org      0F841h
                        C
F841                    C  m_size        proc     near
                        C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
F841  FB                C       sti
F842  1E                C       push     ds
F843  B8 0040           C       mov      ax,data_seg
F846  8E D8             C       mov      ds,ax
                        C
                        C       assume   cs:code, ds:data, es:nothing, ss:nothing
                        C
F848  A1 0013 R         C       mov      ax,word ptr ds:[memory_size]
F84B  1F                C       pop      ds
                        C
F84C  CF                C       iret
                        C
                        C  m_size        endp
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    INT 11h  -- equipment check
                        C  ;--------------------------------------------------------------------
                        C
                        C       XORG     0F84Dh
F84D                    C1      org      0F84Dh
                        C
F84D                    C  m_equip       proc     near
                        C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
F84D  FB                C       sti
F84E  1E                C       push     ds
F84F  B8 0040           C       mov      ax,data_seg
F852  8E D8             C       mov      ds,ax
                        C
                        C       assume   cs:code, ds:data, es:nothing, ss:nothing
                        C
F854  A1 0010 R         C       mov      ax,word ptr ds:[switch_bits]
F857  1F                C       pop      ds
                        C
F858  CF                C       iret
                        C
                        C  m_equip       endp
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    INT 15h  -- cassette I/O
                        C  ;--------------------------------------------------------------------
```

```
                        C
                        C       XORG    0F859h
F859                    C1      org     0F859h
                        C
F859                    C  m_cass       proc    far
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
F859  E9 E68D R         C       jmp     int15a          ;call new int 15 code module
                        C
                        C  ; INT 15 module executes a Far RET 2 to pop the flags and return to the
                        C  ; code which performed the INT 15
                        C
                        C  m_cass       endp
                        C
F85C                    C  code ends

                        C  include nmi.asm
                        C
                        C
                        C  ;===========================================================================
                        C  ;    Filename:nmi.src
                        C  ;
                        C  ;    This module includes INT 02h.
                        C  ;
                        C  ;    And ENABLE_PARITY
                        C  ;
                        C  ;===========================================================================
                        C
F85C                    C  code segment  public 'ROM'
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C  ;---------------------------------------------------------------------------
                        C  ;    INT 02h
                        C  ;---------------------------------------------------------------------------
                        C
                        C       XORG    0F85Fh
F85F                    C1      org     0F85Fh
                        C
F85F                    C  n_intproc    near
                        C
F85F  50                C       push    ax
F860  E4 62             C       in      al,ControlC             ; High two bits indicate parity.
F862  24 C0             C       and     al,0C0h                 ; Mask of low 6 bits.
F864  74 13             C       jz      n_out                   ; It wasn't a parity interrupt!
F866  BE E625 R         C       mov     si,offset parity1_m     ; System board message.
F869  D0 C0             C       rol     al,1
F86B  72 03             C       jc      n_1
F86D  BE E644 R         C       mov     si,offset parity2_m     ; Expansion board message.
F870                    C  n_1:
F870  B8 0000           C       mov     ax,0                    ; change Video Mode to Mode 0
F873  CD 10             C       INT     10h                     ; change the mode
                        C
F875  E8 DFFA R         C       call    DRomString              ; display the appropriate parity msg
F878  F4                C       hlt
F879                    C  n_out:
```

```
F879  58                C       pop     ax
F87A  CF                C       iret
                        C
                        C  n_intendp
                        C
F87B                    C  code ends


                        C  include boot2.asm
                        C
                        C  ;===========================================================================
                        C  ;    Filename:boot.src
                        C  ;
                        C  ;    This module includes INT 19h.
                        C  ;
                        C  ;===========================================================================
                        C
F87B                    C  code segment  public 'ROM'
                        C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C  ;===========================================================================
                        C  ;    INT 19h -- Cold boot routine:
                        C  ;
                        C  ;    This code reads Track 0, Side 0, Sector 0 into memory at
                        C  ;    0000:7C00 and iret's into the secondary boot-strap loader
                        C  ;===========================================================================
                        C  ;    Note:    The stack looks like this at exit!!!!
                        C  ;
                        C  ;                                                  High Address
                        C  ;                        |------------------| <-- sp (at entry & exit)
                        C  ;                        | return fsw flags |
                        C  ;                        |------------------|
                        C  ;                        | return cs segment|
                        C  ;                        |------------------|
                        C  ;                        | return ip offset |
                        C  ;                        |------------------| <-- sp after INT 19h trap
                        C  ;                        |   saved si, bp   |
                        C  ;                        |------------------| <-- sp at loading of return address
                        C  ;                                                   Low Address
                        C  ;
                        C  ;    Output:
                        C  ;    (DL)     = Driver Number              [ 00h -> Floppy Drive (A:);
                        C  ;                                            01h -> Floppy Drive (B:);
                        C  ;                                            80h -> Fixed Disk   (C:). ]
                        C  ;
                        C  ;    (DH)     = Head Number                [ These three parameters will
                        C  ;    (CH)     = Cylinder Number      specify the booted partition
                        C  ;    (CL)     = Sector Number                in the case of the fixed disk. ]
                        C  ;
                        C  ;    (AH)     = Successful status = 0.
                        C  ;    (AL)     = Number of Sectors read  [in order to read 512 bytes. ]
                        C  ;
                        C  ;    (ES:BX)  = Address of the transfer [0000:7C00]
                        C  ;    (DS:BX)
                        C  ;
```

```
                        C  ;    (CS:IP)  = Address of entry point  [0000:7C00]
                        C  ;
                        C  ;    (SS:SP)  = Stack Segment and Pointer are left intact from the INT
                        C  ;               19h invocation for multi-tasking environments.
                        C  ;
                        C  ;    (IF)     = The interrupt enable flag is left intact from the INT
                        C  ;               19h invocation for multi-tasking environments.
                        C  ;
                        C  ;    Trash:   bp destoryed. (si, di, & bp preserved.)
                        C  ;=============================================================================
                        C
F87B                    C  bt_int       proc    near
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
F87B  FB                C       sti                                     ; enable interrupts
F87C  55                C       push    bp                              ; save BP & SI
F87D  56                C       push    si
                        C
                        C       assume  ds:data                         ; reset master table ptr to ROM
F87E  2E: 8E 1E E574 R  C       mov     ds,word ptr cs:[set_ds_word]    ; satisfy assumption
                        C  ;mov word ptr ds:[master_tbl_ptr+0000h],cs:(offset mastab)
                        C  ;mov word ptr ds:[master_tbl_ptr+0002h],cs
                        C
F883  BE D9EC R         C       mov     si,cs:(offset bt_m)             ; boot strap message
F886  E8 DFFA R         C       call    DRomString                      ; print banner
                        C
F889  32 DB             C       xor     bl,bl                           ; disable error message blinking
F88B  53                C       push    bx                              ; save blink status
                        C
                        C       assume  ds:abs0, es:abs0
F88C                    C  bt_o:                                        ; boot strap outer loop
F88C  33 C0             C       xor     ax,ax                           ; AX = abs0_seg.
F88E  8E D8             C       mov     ds,ax                           ; satisfy assumptions
F890  8E C0             C       mov     es,ax
                        C
                        C  ; Reset fd_parms table vector.
                        C
F892  C7 06 0078 R EFC7 R C     mov     word ptr ds:[int1Elocn+0],cs:(offset fd_parms)
F898  8C 0E 007A R      C       mov     word ptr ds:[int1Elocn+2],cs
                        C
                        C  ; Initialize retry loop.
                        C
F89C  BD 0003           C       mov     bp,3                            ; retry counter
F89F                    C  bt_i:                                        ; boot retry inner loop
                        C
                        C  ; Initialize the drive.
                        C
F89F  33 C0             C       xor     ax,ax                           ; AX = 0.
F8A1  8B D8             C       mov     bx,ax                           ; BX = 0.
F8A3  8B C8             C       mov     cx,ax                           ; CX = 0.
F8A5  8B D0             C       mov     dx,ax                           ; DX = 0.
F8A7  CD 13             C       INT     13h
F8A9  72 0A             C       jc      bt_nxt                          ; try again, if error
                        C
                        C  ; Read the boot sector.
```

```
                        C
F8AB  B8 0201           C       mov     ax,0201h                ; read one sector
                        C                                               ; bl = 0.
F8AE  B7 7C             C       mov     bh,7Ch                          ; xfer address = ES:BX = 0:7C00
                        C                                               ; cx = 0.
F8B0  41                C       inc     cx                              ; track 0; sector 1
                        C                                               ; dx = 0.
                        C                                               ; head  0; drive  0
                        C
F8B1  06                C       push    es                              ; save return registers
F8B2  CD 13             C       INT     13h                             ; BX,CX,DX,SI,DI,BP, & DS saved
F8B4  07                C       pop     es                              ; restore return registers
                        C
F8B5                    C  bt_nxt:
F8B5  73 31             C       jnc     bt_ok                           ; jump if no error during read
                        C
F8B7  5B                C       pop     bx                              ; get blink status
F8B8  0A DB             C       or      bl,bl                       ; have 3 retries been completed?
F8BA  74 21             C       jz      bt_dec                          ; jump if no
                        C
                        C  ; We can't boot from the floppy.  Has the INT 18h vector been changed ?
                        C  ; If so, assume INT 18h now addresses a boot-from-the-LAN routine
                        C
F8BC  53                C       push    bx                              ;save blink status
F8BD  A1 0060 R         C       mov     ax,word ptr int18locn           ;get INT 18h vector value
F8C0  8D 1E FFD2 R      C       lea     bx,cs:basic_trap        ;offset of basic trap routine
F8C4  3B D8             C       cmp     bx,ax                           ;vector same as initialized ?
F8C6  5B                C       pop     bx                              ; restore blink status
F8C7  74 04             C       je      bt_again                ;if vector same, stay in boot
F8C9  CD 18             C       INT     18h                             ;otherwise, do an INT 18H
F8CB  EB 1B             C       jmp short bt_ok                         ;in case there's an iret
                        C
F8CD                    C  bt_again:
F8CD  0A DB             C       or      bl,bl                           ; set blink state in SF
F8CF  BE DA06 R         C       mov     si,cs:(offset bt_merr)          ; blink error message on
F8D2  78 03             C       js      bt_blnk                         ; blink state from BL
F8D4  BE DA2B R         C       mov     si,cs:(offset bt_spaces)  ; blink error message off
                        C
F8D7                    C  bt_blnk:
F8D7  E8 DFFA R         C       call    DRomString                      ; blink error message
F8DA  80 F3 80          C       xor     bl,10000000b                    ; toggle blink state
                        C
F8DD                    C  bt_dec:
F8DD  53                C       push    bx                              ; resave blink status
F8DE  4D                C       dec     bp                              ; decrement retry count
F8DF  75 BE             C       jnz     bt_i                            ; and, try again
                        C
F8E1  5B                C       pop     bx                              ; get blink status
F8E2  80 CB 01          C       or      bl,1                            ; enable error message blinking
F8E5  53                C       push    bx                              ; save new status
                        C
F8E6  EB A4             C       jmp     bt_o                            ; and try again, for now.
                        C
F8E8                    C  bt_ok:
F8E8  BE DA2B R         C       mov     si,cs:(offset bt_spaces)  ; blink error message off
```

```
F8EB  E8 DFFA R          C       call    DRomString
F8EE  5E                 C       pop     si                              ; discard blink status
F8EF  5E                 C       pop     si                              ; restore SI
F8F0  8B EC              C       mov     bp,sp
F8F2  89 5E 02           C       mov     word ptr ss:[bp+2].bx           ; return IP = BX = 7C00h
F8F5  8C 46 04           C       mov     word ptr ss:[bp+4].es           ; return CS = ES = 0000h
F8F8  5D                 C       pop     bp                              ; restore BP
F8F9  CF                 C       iret                                    ; return flags
                         C
                         C bt_int        endp
                         C
F8FA                     C code ends

                         C include calendar.asm
                         C
                         C ;=============================================================================
                         C ;       Filename:       cal.src
                         C ;
                         C ;    This module includes c_read and c_write of INT 1Ah.
                         C ;
                         C ;=============================================================================
                         C
F8FA                     C code segment  public 'ROM'
                         C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                         C
                         C
                         C ;=============================================================================
                         C ;   Read Clock Calendar Device  (c_read)
                         C ;
                         C ;    Input:   ah = -2  or 2 (Read Clock Calendar Device)
                         C ;
                         C ;    Output:  bx =    day (from 1-1 of leap year up to 12-31 of leap year+7)
                         C ;             ch =    hour
                         C ;             cl =    minutes
                         C ;             dh =    seconds
                         C ;             dl =    hundredths of seconds
                         C ;
                         C ;    Trash:   None.
                         C ;=============================================================================
                         C
                         C
F8FA                     C c_read        proc    near
                         C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                         C
                         C ; Save registers and clear the "Data Changed" flip/flop
                         C
F8FA  50                 C       push    ax
F8FB  E4 7F              C       in      al,7Fh
                         C
                         C ; Years.
                         C
F8FD  B9 0064            C       mov     cx,100                  ; timeout to prevent endless loop
F900  51                 C       push    cx                      ; it resides on the stack
F901                     C c_rdyr:
F901  59                 C       pop     cx                      ; retrieve timeout count
```

```
F902  E2 06              C       loop    no_TO                   ; if not zero, OK to try again
F904  33 DB              C       xor     bx,bx                   ; return zeros for day
F906  33 D2              C       xor     dx,dx                   ; return zeros for seconds, hundredths
F908  EB 5D              C       jmp short c_read_exit           ; if zero, no more retries, exit
F90A                     C  no_TO:
F90A  51                 C       push    cx                      ; save decremented timeout count
F90B  B2 7F              C       mov     dl,7Fh                  ; interrupts (years mod 8)
F90D  E8 F969 R          C       call    c_rBCD                  ; al = years
F910  72 EF              C       jc      c_rdyr                  ; restart if "Data Changed" detected
F912  8A E8              C       mov     ch,al                   ; ch = saves year mod 8
                         C
                         C  ; Months.
                         C
F914  B2 7C              C       mov     dl,07Ch         ; dl = tens of months port = 7Ch
F916  E8 F98E R          C       call    c_rhex                  ; Input:  dl = tens of mon.s port = 7Ch
                         C                                       ; Output: ax = hex of months (1-12)
                         C                                       ;         dx = day of week port = 7Ah
F919  72 E6              C       jc      c_rdyr                  ; restart if "Data Changed" detected
F91B  48                 C       dec     ax                      ; ax = map month (1-12) to month (0-11)
                         C
F91C  8B D8              C       mov     bx,ax                   ; bx = saves month (0-11)
                         C
                         C  ; Days.
                         C
F91E  4A                 C       dec     dx                      ; dl = tens of days port = 79h
F91F  E8 F98E R          C       call    c_rhex                  ; Input:  dl = tens of days port = 79h
                         C                                       ; Output: ax = hex of days (1-?)
                         C                                       ;         dx = tens of hours port = 77h
F922  72 DD              C       jc      c_rdyr                  ; restart if "Data Changed" detected
F924  48                 C       dec     ax                      ; ax = map days (1-?) to days (0-?)
                         C
                         C  ; Calculate Day (ax has day).
                         C
F925  8B D0              C       mov     dx,ax                   ; dx = day
                         C
                         C  ; Calculate Month (bx has month).
                         C
F927  4B                 C       dec     bx                      ; previous month
F928  78 08              C       js      c_rm0                   ; jump if it was zero
                         C
F92A                     C  c_rmlp:
F92A  E8 FA2A R          C       call    c_gdays         ; get days per month
F92D  03 D0              C       add     dx,ax                   ; dx = day + current month
F92F  4B                 C       dec     bx                      ; previous month
F930  79 F8              C       jns     c_rmlp
                         C
F932                     C  c_rm0:                                       ; zero case
                         C                                       ; dx = day + month
                         C  ; Calculate Year (ch has month).
                         C
F932  33 DB              C       xor     bx,bx                   ; clear bh
F934  8A DD              C       mov     bl,ch                   ; get year mod 8
F936  D1 E3              C       shl     bx,1                    ; make word index
F938  2E: 03 97 E95B R   C       add     dx,word ptr cs:[bx+c_dy_yr]
                         C
```

```
F93D  8B DA           C       mov     bx,dx                   ; bx = day + month + year
                      C
                      C  ; Hours.
                      C
F93F  B2 77           C       mov     dl,077h         ; dl = tens of hours port = 77h
F941  E8 F98E R       C       call    c_rhex                  ; Input:  dl = tens of hours port = 77h
                      C                                       ; Output: ax = hexadecimal of hours
                      C                                       ;         dx = tens of min.s port = 75h
F944  72 BB           C       jc      c_rdyr                  ; restart if "Data Changed" detected
F946  8A E8           C       mov     ch,al                   ; ch = hours
                      C
                      C  ; Minutes.
                      C                                       ; dl = tens of minutes port = 75h
F948  E8 F98E R       C       call    c_rhex                  ; Input:  dl = tens of min.s port = 75h
                      C                                       ; Output: ax = hexadecimal of minutes
                      C                                       ;         dx = tens of sec.s port = 73h
F94B  72 B4           C       jc      c_rdyr                  ; restart if "Data Changed" detected
F94D  8A C8           C       mov     cl,al                   ; cl = minutes
                      C
                      C  ; Seconds.
                      C                                       ; dl = tens of seconds port = 73h
F94F  E8 F98E R       C       call    c_rhex                  ; Input:  dl = tens of sec.s port = 73h
                      C                                       ; Output: ax = hexadecimal of seconds
                      C                                       ;         dx = tenths of secs port = 71h
F952  72 AD           C       jc      c_rdyr                  ; restart if "Data Changed" detected
F954  8A E0           C       mov     ah,al                   ; ah = seconds
                      C  ;;;  push    dx                      ; save seconds (dh)
                      C
                      C  ; Hundredths of Seconds.
                      C
                      C                                       ; dl = tenths of seconds port = 71h
F956  E8 F969 R       C       call    c_rBCD                  ; al = tenths of seconds
F959  72 A6           C       jc      c_rdyr                  ; restart if "Data Changed" detected
F95B  8A F4           C       mov     dh,ah                   ; dh = seconds
F95D  8A E0           C       mov     ah,al                   ; move tenths of seconds to high byte
F95F  32 C0           C       xor     al,al                   ; ax = BCD of hundredths of seconds
F961  E8 F99C R       C       call    c_BCD2hex       ; ax = hex of hundredths of seconds
                      C
                      C  ;;;  pop     dx                      ; restore seconds (dh)
F964  8A D0           C       mov     dl,al                   ; dl = hex of hundredths of seconds
                      C
F966  58              C       pop     ax                      ; clear stack of timeout count
                      C
F967                  C  c_read_exit:
F967  58              C       pop     ax                      ;retrieve initial value
F968  C3              C       ret
                      C  c_read       endp
                      C
                      C  ;---------------------------------------------------------------------
                      C  ;    Read a BCD byte (c_rBCD)
                      C  ;
                      C  ;    Input a BCD byte
                      C  ;
                      C  ;    Input:  dl = pointer to whatever port
                      C  ;    Output: al = BCD byte if CF clear (trash otherwise)
```

```
                        C  ;            al = 1 and CF = clear if RTC inoperative
                        C  ;            CF = set if " Data Changed"
                        C  ;            CF = clear if " Data OK"
                        C  ;
                        C  ;    Trash:  None.
                        C  ;-------------------------------------------------------------------------
                        C
F969                    C  c_rBCD       proc    near
F969  51                C      push     cx                      ; save cx
F96A  B1 09             C      mov      cl,9                    ; maximum valid BCD value
F96C  32 F6             C      xor      dh,dh                   ; clear dh
                        C
F96E  EC                C      in       al,dx                   ; get the byte
F96F  24 0F             C      and      al,0Fh                  ; clear high 4 bits
F971  3A C8             C      cmp      cl,al                   ; is it 10 or more?
F973  72 09             C      jb       c_RTCchk          ; if so, go check for RTC OK
F975  8A E8             C      mov      ch,al                   ; save BCD value in ch
F977  EC                C      in       al,dx                   ; try again
F978  24 0F             C      and      al,0Fh                  ; clear high 4 bits
F97A  3A C5             C      cmp      al,ch                   ; same value on second try ?
F97C  74 0E             C      je       c_rBret           ; if so, return with CF clear
                        C
F97E                    C  c_RTCchk:
F97E  B9 0003           C      mov      cx,3
F981  EC                C  c_rBlp: in    al,dx                   ; read again
F982  24 0F             C      and      al,0Fh                  ; clear high 4 bits
F984  3C 0A             C      cmp      al,10                   ; is it less than 10 ?
F986  72 04             C      jb       c_rBret           ; if so, return with CF set
F988  E2 F7             C      loop     c_rBlp                  ; else try again
F98A  B0 01             C      mov      al,1                    ; if timeout, return one with CF clear
                        C
F98C                    C  c_rBret:
F98C  59                C      pop      cx                      ; restore cx
F98D  C3                C      ret
                        C  c_rBCD       endp
                        C
                        C  ;-------------------------------------------------------------------------
                        C  ;    Convert to Hex    (c_rhex)
                        C  ;
                        C  ;    Inputs two BCD bytes and converts to hexdecimal word.
                        C  ;
                        C  ;    Input:  dl = pointer to tens of whatever port
                        C  ;    Output: ax = hexadecimal word (ah = 0) if CF clear (trash otherwise)
                        C  ;            dx = pointer to tens of previous port (dh = 0) if CF clear
                        C  ;            CF = set if " Data Changed"
                        C  ;            CF = clear if " Data OK"
                        C  ;
                        C  ;    Trash:  None.
                        C  ;-------------------------------------------------------------------------
                        C
F98E                    C  c_rhex       proc    near
                        C      assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
F98E  E8 F969 R         C      call     c_rBCD                  ; in from tens of whatever
F991  72 0C             C      jc       c_rhdc                  ; return if " Data Changed" detected
```

```
F993  8A E0           C       mov     ah,al                   ; move tens of whatever to high byte
F995  4A              C       dec     dx                      ; dx points to units of whatever port
F996  E8 F969 R       C       call    c_rBCD                  ; in from units of whatever
F999  72 04           C       jc      c_rhdc                  ; return if "Data Changed" detected
F99B  4A              C       dec     dx                      ; dx points to tens of previous port
                      C
                      C ;     jmp     short c_BCD2hex   ; fall through
                      C
                      C c_rhex        endp
                      C
                      C ;---------------------------------------------------------------------
                      C ;
                      C ;     BCD to Hexadecimal (c_BCD2hex)
                      C ;
                      C ;     Input:  ah = high BCD digit
                      C ;             al = low BCD digit
                      C ;     Output: ax = hexadecimal byte (ah = 0)
                      C ;;;;;;                dh = 0
                      C ;     Trash:  None.
                      C ;---------------------------------------------------------------------
                      C
F99C                  C c_BCD2hex     proc    near
                      C               assume  cs:code, ds:nothing, es:nothing, ss:nothing
                      C
                      C ;;;   mov     dh,ah                   ; dh =      hi BCD digit
                      C ;;;   shl     dh,1                    ; dh =   2*(hi BCD digit)
                      C ;;;   shl     dh,1                    ; dh =   4*(hi BCD digit)
                      C ;;;   add     dh,ah                   ; dh =   5*(hi BCD digit)
                      C ;;;   shl     dh,1                    ; dh = 10*(hi BCD digit)
                      C ;;;   add     al,dh                   ; al = 10*(hi BCD digit)+(low BCD digit)
                      C ;;;   xor     ah,ah                   ; ax = 10*(hi BCD digit)+(low BCD digit)
                      C ;;;   xor     dh,dh                   ; dh = 0, CF = 0
F99C  D5 0A           C       aad                             ; al <- ah * 10 + al; ah <- 0
F99E  F8              C       clc                             ; CF <- 0
F99F  C3              C c_rhdc: ret
                      C
                      C c_BCD2hex     endp
                      C
                      C ;---------------------------------------------------------------------
                      C ;       Write Clock Calendar Device  (c_write)
                      C ;
                      C ;     Input:  ah = -1 or 3 (Write Clock Calendar)
                      C ;             bx =    day (from 1-1 of leap year up to 12-31 of leap year+7)
                      C ;                =    (0-2921) = (0-B69h)
                      C ;             ch =    hour    (0-23)
                      C ;             cl =    minutes (0-59)
                      C ;     Output: ah = -1  implies date/time error
                      C ;             ah = 0   implies date/time OK
                      C ;     Trash:  None.
                      C ;---------------------------------------------------------------------
                      C
F9A0                  C c_write       proc    near
                      C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                      C
                      C ; Check for errors.
```

```
                        C
F9A0  80 F9 3C          C       cmp     cl,60                   ; cl = minutes (0-59)
F9A3  73 73             C       jae     c_werr
F9A5  80 FD 18          C       cmp     ch,24                   ; ch = hour    (0-23)
F9A8  73 6E             C       jae     c_werr
F9AA  81 FB 0B6A        C       cmp     bx,(2*366)+(6*365); bx = day from leap year mod 8
F9AE  73 68             C       jae     c_werr                  ;    = (0-2921) = (0-0B69h)
                        C
                        C  ; Save registers.
                        C
F9B0  50                C       push    ax
F9B1  53                C       push    bx
F9B2  51                C       push    cx
F9B3  52                C       push    dx
                        C
                        C  ; Initialize and Stop Clock.
                        C
F9B4  33 C0             C       xor     ax,ax                   ; ax = 0
F9B6  E6 70             C       out     70h,al                  ; test only port = out of test mode
F9B8  E6 7E             C       out     7Eh,al                  ; stop/start port = stop clock
                        C
                        C  ; Minutes.
                        C
F9BA  B2 74             C       mov     dl,074h                 ; dl = units of minutes port = 74h
F9BC  8A C1             C       mov     al,cl                   ; al = minutes (0-59)
F9BE  E8 FA19 R         C       call    c_whex                  ; Input:  al = hexadecimal of minutes
                        C                                       ;         dl = units of min.s port = 74h
                        C                                       ; Output: ax = trash
                        C                                       ;         dx = units of hours port = 76h
                        C  ; Hours.
                        C                                       ; dl = units of hours port = 76h
F9C1  8A C5             C       mov     al,ch                   ; al = hours (0-23)
F9C3  E8 FA19 R         C       call    c_whex                  ; Input:  al = hexadecimal of hours
                        C                                       ;         dl = units of hours port = 76h
                        C                                       ; Output: ax = trash
                        C                                       ;         dx = units of days port = 78h
                        C  ; Calculate Year.
                        C
F9C6  8B D3             C       mov     dx,bx                   ; dx = day from leap year mod 8
F9C8  BB 0010           C       mov     bx,(8*2)          ; word index of year
F9CB                    C  c_wylp:
F9CB  4B                C       dec     bx
F9CC  4B                C       dec     bx
F9CD  2E: 3B 97 E95B R  C       cmp     dx,word ptr cs:[bx+c_dy_yr]
F9D2  72 F7             C       jb      c_wylp
                        C
F9D4  2E: 2B 97 E95B R  C       sub     dx,word ptr cs:[bx+c_dy_yr]; dx = saves day of year
F9D9  D1 EB             C       shr     bx,1                            ; bl = year mod 8
F9DB  8A EB             C       mov     ch,bl                           ; ch = saves year mod 8
                        C
                        C  ; Calculate Days & Months.
                        C
F9DD  BB FFFF           C       mov     bx,-1                   ; start at January
F9E0                    C  c_wmlp:
F9E0  43                C       inc     bx                      ; next month
```

```
F9E1  E8 FA2A R     C       call    c_gdays         ; get days per month
F9E4  2B D0         C       sub     dx,ax
F9E6  73 F8         C       jae     c_wmlp
                    C                               ; bx = month (0-11)
F9E8  03 C2         C       add     ax,dx           ; ax = day   (0-?)
                    C
                    C  ; Days.
                    C
F9EA  B2 78         C       mov     dl,078h         ; dl = units of days port = 78h
F9EC  40            C       inc     ax              ; al = map days (0-?) to days (1-?)
F9ED  E8 FA19 R     C       call    c_whex          ; Input:  al = hexadecimal of days
                    C                               ;         dl = units of days port = 78h
                    C                               ; Output: ax = trash
                    C                               ;         dx = day of week port = 7Ah
                    C  ; Months.
                    C
F9F0  42            C       inc     dx              ; dl = units of months port = 7Bh
F9F1  8B C3         C       mov     ax,bx           ; al = month (0-11)
F9F3  40            C       inc     ax              ; al = map month (0-11) to month (1-12)
F9F4  E8 FA19 R     C       call    c_whex          ; Input:  al = hexadecimal of months
                    C                               ;         dl = units of mon.s port = 7Bh
                    C                               ; Output: ax = trash
                    C                               ;         dx = leap year port = 7Dh
                    C  ; Leap Years.
                    C                               ; dx = leap year port = 7Dh
F9F7  B0 08         C       mov     al,08h          ; set leap year bit
F9F9  8A CD         C       mov     cl,ch           ; get year mod 8
F9FB  80 E1 03      C       and     cl,03h          ; get year mod 4
F9FE  D2 E8         C       shr     al,cl           ; shift leap year bit into position
FA00  EE            C       out     dx,al
                    C
                    C  ; Years.
                    C
FA01  42            C       inc     dx              ; dx = stop/start = 7Eh
FA02  42            C       inc     dx              ; dx = interrupt  = 7Fh
FA03  8A C5         C       mov     al,ch           ; get year mod 8
FA05  0C 08         C       or      al,08h          ; set 'repeated interrupt' bit
FA07  EE            C       out     dx,al
FA08  90            C       nop
FA09  EC            C       in      al,dx
FA0A  90            C       nop
FA0B  EC            C       in      al,dx
FA0C  90            C       nop
FA0D  EC            C       in      al,dx
                    C
                    C  ; Start Clock.
                    C
FA0E  B0 FF         C       mov     al,0FFh         ; al = 0FFh
FA10  4A            C       dec     dx              ; dx = stop/start = 7Eh
FA11  EE            C       out     dx,al           ; start clock
                    C
                    C  ; Restore registers.
                    C
FA12  5A            C       pop     dx
FA13  59            C       pop     cx
```

```
FA14  5B              C       pop     bx
FA15  58              C       pop     ax
FA16  32 E4           C       xor     ah,ah                   ; ah = 0 no error
                      C
FA18                  C  c_werr:                                      ; ah = -1error
FA18  C3              C       ret
                      C
                      C  c_write      endp
                      C
                      C  ;---------------------------------------------------------------------
                      C  ;       Converts hexadecimal byte to BCD and outputs both bytes. (c_whex)
                      C  ;
                      C  ;    Input:  al = hexadecimal byte
                      C  ;            dl = pointer to units of whatever port
                      C  ;    Output: dx = pointer to units of next port (dh = 0)
                      C  ;
                      C  ;    Trash:  ax destroyed.
                      C  ;---------------------------------------------------------------------
                      C
FA19                  C  c_whex       proc    near
                      C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                      C
FA19  32 E4           C       xor     ah,ah                   ; ax = hexadecimal byte
FA1B  B6 0A           C       mov     dh,10                   ; dh = divisor
FA1D  F6 F6           C       div     dh                      ; ah = remainder = low BCD digit (0-9)
                      C                                       ; al = quotient  = high BCD digit
FA1F  86 C4           C       xchg    al,ah                   ; ah = quotient  = high BCD digit
                      C                                       ; al = remainder = low BCD digit (0-9)
                      C
FA21  32 F6           C       xor     dh,dh                   ; dx points to units of whatever port
FA23  EE              C       out     dx,al                   ; out to units of whatever
FA24  8A C4           C       mov     al,ah                   ; move tens of whatever to low byte
FA26  42              C       inc     dx                      ; dx points to tens of whatever port
FA27  EE              C       out     dx,al                   ; out to tens of whatever
FA28  42              C       inc     dx                      ; dx points to units of next port
FA29  C3              C       ret
                      C
                      C  c_whex       endp
                      C
                      C  ;---------------------------------------------------------------------
                      C  ;
                      C  ;       Get Days per Month. (c_gdays)
                      C  ;
                      C  ;       This routine calculates the number of days
                      C  ;    per month based on the year without checking validity of month.
                      C  ;
                      C  ;    Input:  bx      = month  (assumes bx < 12)
                      C  ;            ch      = year
                      C  ;    Output: ax      = days in month, if month valid; else garbage
                      C  ;
                      C  ;    Trash:  None.
                      C  ;---------------------------------------------------------------------
                      C
FA2A                  C  c_gdays      proc    near
                      C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
```

```
                        C
FA2A  33 C0             C       xor     ax,ax                   ; clear ah
FA2C  2E: 8A 87 E96B R  C       mov     al,cs:[bx+c_dy_mo]
                        C
FA31  80 FB 01          C       cmp     bl,1                    ; is is February?
FA34  75 06             C       jnz     c_gret                  ; if not February, return
                        C
FA36  F6 C5 03          C       test    ch,03h                  ; if year = 0 mod 4, leap year
FA39  75 01             C       jnz     c_gret                  ; if not leap year, ax = 28th, so return
                        C
FA3B  40                C       inc     ax                      ; load ax with February 29th
FA3C  C3                C c_gret:       ret
                        C
                        C c_gdays       endp
                        C
FA3D                    C code ends



                           ;--------------------------------------------------------------------------
                           ;       ORG'd Font Tables
                           ;--------------------------------------------------------------------------

FA3D                       code segment  public 'ROM'
                                assume   cs:code, ds:nothing, es:nothing, ss:nothing

                                XORG     0FA6Eh
FA6E                     1      org      0FA6Eh
FA6E                       font_lo_8x8   label    byte
                        C  include fontlo8.asm
                        C
FA6E                    C  fontlo8 proc near       ;   System Font Table for M24
                        C
FA6E  00 00 00 00 00 00 00 C       DB       00h,00h,00h,00h,00h,00h,00h,00h             ;     0
      00                C
FA76  7E 81 A5 81 BD 99 81 C       DB       7eh,81h,0a5h,81h,0bdh,99h,81h,7eh   ;     1
      7E                C
FA7E  7E FF DB FF C3 E7 FF C       DB       7eh,0ffh,0dbh,0ffh,0c3h,0e7h,0ffh,7eh       ;     2
      7E                C
FA86  6C FE FE FE 7C 38 10 C       DB       6ch,0feh,0feh,0feh,7ch,38h,10h,00h  ;     3
      00                C
FA8E  10 38 7C FE 7C 38 10 C       DB       10h,38h,7ch,0feh,7ch,38h,10h,00h    ;     4
      00                C
FA96  38 7C 38 FE FE 7C 38 C       DB       38h,7ch,38h,0feh,0feh,7ch,38h,7ch   ;     5
      7C                C
FA9E  10 10 38 7C FE 7C 38 C       DB       10h,10h,38h,7ch,0feh,7ch,38h,7ch    ;     6
      7C                C
FAA6  00 00 18 3C 3C 18 00 C       DB       00h,00h,18h,3ch,3ch,18h,00h,00h             ;     7
      00                C
FAAE  FF FF E7 C3 C3 E7 FF C       DB       0ffh,0ffh,0e7h,0c3h,0c3h,0e7h,0ffh,0ffh     ;     8
      FF                C
FAB6  00 3C 66 42 42 66 3C C       DB       00h,3ch,66h,42h,42h,66h,3ch,00h             ;     9
      00                C
FABE  FF C3 99 BD BD 99 C3 C       DB       0ffh,0c3h,99h,0bdh,0bdh,99h,0c3h,0ffh       ;     a
      FF                C
```

```
FAC6  0F 07 0F 7D CC CC CC  C       DB      0fh,07h,0fh,7dh,0cch,0cch,0cch,78h  ;     b
      78                    C
FACE  3C 66 66 66 3C 18 7E  C       DB      3ch,66h,66h,66h,3ch,18h,7eh,18h        ;     c
      18                    C
FAD6  3F 33 3F 30 30 70 F0  C       DB      3fh,33h,3fh,30h,30h,70h,0f0h,0e0h  ;     d
      E0                    C
FADE  7F 63 7F 63 63 67 E6  C       DB      7fh,63h,7fh,63h,63h,67h,0e6h,0c0h  ;     e
      C0                    C
FAE6  99 5A 3C E7 E7 3C 5A  C       DB      99h,5ah,3ch,0e7h,0e7h,3ch,5ah,99h  ;     f
      99                    C
FAEE  80 E0 F8 FE F8 E0 80  C       DB      80h,0e0h,0f8h,0feh,0f8h,0e0h,80h,00h;     10
      00                    C
FAF6  02 0E 3E FE 3E 0E 02  C       DB      02h,0eh,3eh,0feh,3eh,0eh,02h,00h   ;     11
      00                    C
FAFE  18 3C 7E 18 18 7E 3C  C       DB      18h,3ch,7eh,18h,18h,7eh,3ch,18h        ;     12
      18                    C
FB06  66 66 66 66 66 00 66  C       DB      66h,66h,66h,66h,66h,00h,66h,00h        ;     13
      00                    C
FB0E  7F DB DB 7B 1B 1B 1B  C       DB      7fh,0dbh,0dbh,7bh,1bh,1bh,1bh,00h  ;     14
      00                    C
FB16  3E 63 38 6C 6C 38 CC  C       DB      3eh,63h,38h,6ch,6ch,38h,0cch,78h   ;     15
      78                    C
FB1E  00 00 00 00 7E 7E 7E  C       DB      00h,00h,00h,00h,7eh,7eh,7eh,00h        ;     16
      00                    C
FB26  18 3C 7E 18 7E 3C 18  C       DB      18h,3ch,7eh,18h,7eh,3ch,18h,0ffh   ;     17
      FF                    C
FB2E  18 3C 7E 18 18 18 18  C       DB      18h,3ch,7eh,18h,18h,18h,18h,00h        ;     18
      00                    C
FB36  18 18 18 18 7E 3C 18  C       DB      18h,18h,18h,18h,7eh,3ch,18h,00h        ;     19
      00                    C
FB3E  00 18 0C FE 0C 18 00  C       DB      00h,18h,0ch,0feh,0ch,18h,00h,00h   ;     1a
      00                    C
FB46  00 30 60 FE 60 30 00  C       DB      00h,30h,60h,0feh,60h,30h,00h,00h   ;     1b
      00                    C
FB4E  00 00 C0 C0 C0 FE 00  C       DB      00h,00h,0c0h,0c0h,0c0h,0feh,00h,00h ;     1c
      00                    C
FB56  00 24 66 FF 66 24 00  C       DB      00h,24h,66h,0ffh,66h,24h,00h,00h   ;     1d
      00                    C
FB5E  00 18 3C 7E FF FF 00  C       DB      00h,18h,3ch,7eh,0ffh,0ffh,00h,00h  ;     1e
      00                    C
FB66  00 FF FF 7E 3C 18 00  C       DB      00h,0ffh,0ffh,7eh,3ch,18h,00h,00h  ;     1f
      00                    C
FB6E  00 00 00 00 00 00 00  C       DB      00h,00h,00h,00h,00h,00h,00h,00h        ;' ' 20
      00                    C
FB76  30 78 78 30 30 00 30  C       DB      30h,78h,78h,30h,30h,00h,30h,00h        ;'!' 21
      00                    C
FB7E  6C 6C 6C 00 00 00 00  C       DB      6ch,6ch,6ch,00h,00h,00h,00h,00h        ;'"' 22
      00                    C
FB86  6C 6C FE 6C FE 6C 6C  C       DB      6ch,6ch,0feh,6ch,0feh,6ch,6ch,00h  ;'#' 23
      00                    C
FB8E  30 7C C0 78 0C F8 30  C       DB      30h,7ch,0c0h,78h,0ch,0f8h,30h,00h  ;'$' 24
      00                    C
FB96  00 C6 CC 18 30 66 C6  C       DB      00h,0c6h,0cch,18h,30h,66h,0c6h,00h ;'%' 25
      00                    C
FB9E  38 6C 38 76 DC CC 76  C       DB      38h,6ch,38h,76h,0dch,0cch,76h,00h  ;'&' 26
```

```
      C0                    C
FBA6  60 60 C0 00 00 00 00  C       DB      60h,60h,0c0h,00h,00h,00h,00h,00h    ;''' 27
      00                    C
FBAE  18 30 60 60 60 30 18  C       DB      18h,30h,60h,60h,60h,30h,18h,00h             ;'(' 28
      00                    C
FBB6  60 30 18 18 18 30 60  C       DB      60h,30h,18h,18h,18h,30h,60h,00h             ;')' 29
      00                    C
FBBE  00 66 3C FF 3C 66 00  C       DB      00h,66h,3ch,0ffh,3ch,66h,00h,00h    ;'*' 2a
      00                    C
FBC6  00 30 30 FC 30 30 00  C       DB      00h,30h,30h,0fch,30h,30h,00h,00h    ;'+' 2b
      00                    C
FBCE  00 00 00 00 00 30 30  C       DB      00h,00h,00h,00h,00h,30h,30h,60h             ;',' 2c
      60                    C
FBD6  00 00 00 FC 00 00 00  C       DB      00h,00h,00h,0fch,00h,00h,00h,00h    ;'-' 2d
      00                    C
FBDE  00 00 00 00 00 30 30  C       DB      00h,00h,00h,00h,00h,30h,30h,00h             ;'.' 2e
      00                    C
FBE6  06 0C 18 30 60 C0 80  C       DB      06h,0ch,18h,30h,60h,0c0h,80h,00h    ;'/' 2f
      00                    C
                            C
FBEE  7C C6 CE DE F6 E6 7C  C       DB      7ch,0c6h,0ceh,0deh,0f6h,0e6h,7ch,00h;'0' 30
      00                    C
                            C
FBF6  30 70 30 30 30 30 FC  C       DB      30h,70h,30h,30h,30h,30h,0fch,00h    ;'1' 31
      00                    C
FBFE  78 CC 0C 38 60 CC FC  C       DB      78h,0cch,0ch,38h,60h,0cch,0fch,00h  ;'2' 32
      00                    C
FC06  78 CC 0C 38 0C CC 78  C       DB      78h,0cch,0ch,38h,0ch,0cch,78h,00h   ;'3' 33
      00                    C
FC0E  1C 3C 6C CC FE 0C 1E  C       DB      1ch,3ch,6ch,0cch,0feh,0ch,1eh,00h   ;'4' 34
      00                    C
FC16  FC C0 F8 0C 0C CC 78  C       DB      0fch,0c0h,0f8h,0ch,0ch,0cch,78h,00h ;'5' 35
      00                    C
FC1E  38 60 C0 F8 CC CC 78  C       DB      38h,60h,0c0h,0f8h,0cch,0cch,78h,00h ;'6' 36
      00                    C
FC26  FC CC 0C 18 30 30 30  C       DB      0fch,0cch,0ch,18h,30h,30h,30h,00h   ;'7' 37
      00                    C
FC2E  78 CC CC 78 CC CC 78  C       DB      78h,0cch,0cch,78h,0cch,0cch,78h,00h ;'8' 38
      00                    C
FC36  78 CC CC 7C 0C 18 70  C       DB      78h,0cch,0cch,7ch,0ch,18h,70h,00h   ;'9' 39
      00                    C
FC3E  00 30 30 00 00 30 30  C       DB      00h,30h,30h,00h,00h,30h,30h,00h             ;':' 3a
      00                    C
FC46  00 30 30 00 00 30 30  C       DB      00h,30h,30h,00h,00h,30h,30h,60h             ;';' 3b
      60                    C
FC4E  18 30 60 C0 60 30 18  C       DB      18h,30h,60h,0c0h,60h,30h,18h,00h    ;'<' 3c
      00                    C
FC56  00 00 FC 00 00 FC 00  C       DB      00h,00h,0fch,00h,00h,0fch,00h,00h   ;'=' 3d
      00                    C
FC5E  60 30 18 0C 18 30 60  C       DB      60h,30h,18h,0ch,18h,30h,60h,00h             ;'>' 3e
      00                    C
FC66  78 CC 0C 18 30 00 30  C       DB      78h,0cch,0ch,18h,30h,00h,30h,00h    ;'?' 3f
      00                    C
FC6E  7C C6 DE DE DE C0 78  C       DB      7ch,0c6h,0deh,0deh,0deh,0c0h,78h,00h;'@' 40
      00                    C
```

```
FC76  30 78 CC CC FC CC CC  C       DB      30h,78h,0cch,0cch,0fch,0cch,0cch,00h;'A' 41
      00                    C
FC7E  FC 66 66 7C 66 66 FC  C       DB      0fch,66h,66h,7ch,66h,66h,0fch,00h   ;'B' 42
      00                    C
FC86  3C 66 C0 C0 C0 66 3C  C       DB      3ch,66h,0c0h,0c0h,0c0h,66h,3ch,00h  ;'C' 43
      00                    C
FC8E  F8 6C 66 66 66 6C F8  C       DB      0f8h,6ch,66h,66h,66h,6ch,0f8h,00h   ;'D' 44
      00                    C
FC96  FE 62 68 78 68 62 FE  C       DB      0feh,62h,68h,78h,68h,62h,0feh,00h   ;'E' 45
      00                    C
FC9E  FE 62 68 78 68 60 F0  C       DB      0feh,62h,68h,78h,68h,60h,0f0h,00h   ;'F' 46
      00                    C
FCA6  3C 66 C0 C0 CE 66 3E  C       DB      3ch,66h,0c0h,0c0h,0ceh,66h,3eh,00h  ;'G' 47
      00                    C
FCAE  CC CC CC FC CC CC CC  C       DB      0cch,0cch,0cch,0fch,0cch,0cch,0cch,00h      ;'H' 48
      00                    C
FCB6  78 30 30 30 30 30 78  C       DB      78h,30h,30h,30h,30h,30h,78h,00h             ;'I' 49
      00                    C
FCBE  1E 0C 0C 0C CC CC 78  C       DB      1eh,0ch,0ch,0ch,0cch,0cch,78h,00h   ;'J' 4a
      00                    C
FCC6  E6 66 6C 78 6C 66 E6  C       DB      0e6h,66h,6ch,78h,6ch,66h,0e6h,00h   ;'K' 4b
      00                    C
FCCE  F0 60 60 60 62 66 FE  C       DB      0f0h,60h,60h,60h,62h,66h,0feh,00h   ;'L' 4c
      00                    C
FCD6  C6 EE FE FE D6 C6 C6  C       DB      0c6h,0eeh,0feh,0feh,0d6h,0c6h,0c6h,00h      ;'M' 4d
      00                    C
FCDE  C6 E6 F6 DE CE C6 C6  C       DB      0c6h,0e6h,0f6h,0deh,0ceh,0c6h,0c6h,00h      ;'N' 4e
      00                    C
FCE6  38 6C C6 C6 C6 6C 38  C       DB      38h,6ch,0c6h,0c6h,0c6h,6ch,38h,00h  ;'O' 4f
      00                    C
FCEE  FC 66 66 7C 60 60 F0  C       DB      0fch,66h,66h,7ch,60h,60h,0f0h,00h   ;'P' 50
      00                    C
FCF6  78 CC CC CC DC 78 1C  C       DB      78h,0cch,0cch,0cch,0dch,78h,1ch,00h ;'Q' 51
      00                    C
FCFE  FC 66 66 7C 6C 66 E6  C       DB      0fch,66h,66h,7ch,6ch,66h,0e6h,00h   ;'R' 52
      00                    C
FD06  78 CC E0 70 1C CC 78  C       DB      78h,0cch,0e0h,70h,1ch,0cch,78h,00h  ;'S' 53
      00                    C
FD0E  FC B4 30 30 30 30 78  C       DB      0fch,0b4h,30h,30h,30h,30h,78h,00h   ;'T' 54
      00                    C
FD16  CC CC CC CC CC CC FC  C       DB      0cch,0cch,0cch,0cch,0cch,0cch,0fch,00h      ;'U' 55
      00                    C
FD1E  CC CC CC CC CC 78 30  C       DB      0cch,0cch,0cch,0cch,0cch,78h,30h,00h;'V' 56
      00                    C
FD26  C6 C6 C6 D6 FE EE C6  C       DB      0c6h,0c6h,0c6h,0d6h,0feh,0eeh,0c6h,00h      ;'W' 57
      00                    C
FD2E  C6 C6 6C 38 38 6C C6  C       DB      0c6h,0c6h,6ch,38h,38h,6ch,0c6h,00h  ;'X' 58
      00                    C
FD36  CC CC CC 78 30 30 78  C       DB      0cch,0cch,0cch,78h,30h,30h,78h,00h  ;'Y' 59
      00                    C
FD3E  FE C6 8C 18 32 66 FE  C       DB      0feh,0c6h,8ch,18h,32h,66h,0feh,00h  ;'Z' 5a
      00                    C
FD46  78 60 60 60 60 60 78  C       DB      78h,60h,60h,60h,60h,60h,78h,00h             ;'[' 5b
      00                    C
FD4E  C0 60 30 18 0C 06 02  C       DB      0c0h,60h,30h,18h,0ch,06h,02h,00h    ;'\' 5c
```

```
      00                        C
FD56  78 18 18 18 18 18 78      C       DB      78h,18h,18h,18h,18h,18h,78h,00h             ;']' 5d
      00                        C
FD5E  10 38 6C C6 00 00 00      C       DB      10h,38h,6ch,0c6h,00h,00h,00h,00h    ;'^' 5e
      00                        C
FD66  00 00 00 00 00 00 00      C       DB      00h,00h,00h,00h,00h,00h,00h,0ffh    ;'_' 5f
      FF                        C
FD6E  30 30 18 00 00 00 00      C       DB      30h,30h,18h,00h,00h,00h,00h,00h             ;'`' 60
      00                        C
FD76  00 00 78 0C 7C CC 76      C       DB      00h,00h,78h,0ch,7ch,0cch,76h,00h    ;'a' 61
      00                        C
FD7E  E0 60 60 7C 66 66 DC      C       DB      0e0h,60h,60h,7ch,66h,66h,0dch,00h   ;'b' 62
      00                        C
FD86  00 00 78 CC C0 CC 78      C       DB      00h,00h,78h,0cch,0c0h,0cch,78h,00h  ;'c' 63
      00                        C
FD8E  1C 0C 0C 7C CC CC 76      C       DB      1ch,0ch,0ch,7ch,0cch,0cch,76h,00h   ;'d' 64
      00                        C
FD96  00 00 78 CC FC C0 78      C       DB      00h,00h,78h,0cch,0fch,0c0h,78h,00h  ;'e' 65
      00                        C
FD9E  38 6C 60 F0 60 60 F0      C       DB      38h,6ch,60h,0f0h,60h,60h,0f0h,00h   ;'f' 66
      00                        C
FDA6  00 00 76 CC CC 7C 0C      C       DB      00h,00h,76h,0cch,0cch,7ch,0ch,0f8h  ;'g' 67
      F8                        C
FDAE  E0 60 6C 76 66 66 E6      C       DB      0e0h,60h,6ch,76h,66h,66h,0e6h,00h   ;'h' 68
      00                        C
FDB6  30 00 70 30 30 30 78      C       DB      30h,00h,70h,30h,30h,30h,78h,00h             ;'i' 69
      00                        C
FDBE  0C 00 0C 0C 0C CC CC      C       DB      0ch,00h,0ch,0ch,0ch,0cch,0cch,78h   ;'j' 6a
      78                        C
FDC6  E0 60 66 6C 78 6C E6      C       DB      0e0h,60h,66h,6ch,78h,6ch,0e6h,00h   ;'k' 6b
      00                        C
FDCE  70 30 30 30 30 30 78      C       DB      70h,30h,30h,30h,30h,30h,78h,00h             ;'l' 6c
      00                        C
FDD6  00 00 CC FE FE D6 C6      C       DB      00h,00h,0cch,0feh,0feh,0d6h,0c6h,00h;'m' 6d
      00                        C
FDDE  00 00 F8 CC CC CC CC      C       DB      00h,00h,0f8h,0cch,0cch,0cch,0cch,00h;'n' 6e
      00                        C
FDE6  00 00 78 CC CC CC 78      C       DB      00h,00h,78h,0cch,0cch,0cch,78h,00h  ;'o' 6f
      00                        C
FDEE  00 00 DC 66 66 7C 60      C       DB      00h,00h,0dch,66h,66h,7ch,60h,0f0h   ;'p' 70
      F0                        C
FDF6  00 00 76 CC CC 7C 0C      C       DB      00h,00h,76h,0cch,0cch,7ch,0ch,1eh   ;'q' 71
      1E                        C
FDFE  00 00 DC 76 66 60 F0      C       DB      00h,00h,0dch,76h,66h,60h,0f0h,00h   ;'r' 72
      00                        C
FE06  00 00 7C C0 78 0C F8      C       DB      00h,00h,7ch,0c0h,78h,0ch,0f8h,00h   ;'s' 73
      00                        C
FE0E  10 30 7C 30 30 34 18      C       DB      10h,30h,7ch,30h,30h,34h,18h,00h             ;'t' 74
      00                        C
FE16  00 00 CC CC CC CC 76      C       DB      00h,00h,0cch,0cch,0cch,0cch,76h,00h ;'u' 75
      00                        C
FE1E  00 00 CC CC CC 78 30      C       DB      00h,00h,0cch,0cch,0cch,78h,30h,00h  ;'v' 76
      00                        C
FE26  00 00 C6 D6 FE FE 6C      C       DB      00h,00h,0c6h,0d6h,0feh,0feh,6ch,00h ;'w' 77
      00                        C
```

```
FE2E  00 00 C6 6C 38 6C C6  C       DB      00h,00h,0c6h,6ch,38h,6ch,0c6h,00h   ;'x' 78
      00                    C
FE36  00 00 CC CC CC 7C 0C  C       DB      00h,00h,0cch,0cch,0cch,7ch,0ch,0f8h ;'y' 79
      F8                    C
FE3E  00 00 FC 98 30 64 FC  C       DB      00h,00h,0fch,98h,30h,64h,0fch,00h   ;'z' 7a
      00                    C
FE46  1C 30 30 E0 30 30 1C  C       DB      1ch,30h,30h,0e0h,30h,30h,1ch,00h    ;'{' 7b
      00                    C
FE4E  18 18 18 00 18 18 18  C       DB      18h,18h,18h,00h,18h,18h,18h,00h             ;'|' 7c
      00                    C
FE56  E0 30 30 1C 30 30 E0  C       DB      0e0h,30h,30h,1ch,30h,30h,0e0h,00h   ;'}' 7d
      00                    C
FE5E  76 DC 00 00 00 00 00  C       DB      76h,0dch,00h,00h,00h,00h,00h,00h    ;'~' 7e
      00                    C
FE66  00 10 38 6C C6 C6 FE  C       DB      00h,10h,38h,6ch,0c6h,0c6h,0feh,00h  ;'' 7f
      00                    C
                            C   ;End of font matrix
                            C
                            C   fontlo8 endp


FE6E                            code ends


                            C   include rtc.asm
                            C
                            C   ;===========================================================================
                            C   ;    Filename:rtc.src
                            C   ;
                            C   ;    This module includes INT 08h & 1Ah.
                            C   ;
                            C   ;===========================================================================
                            C
FE6E                        C   code segment  public 'ROM'
                            C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                            C
                            C   ;===========================================================================
                            C   ;    INT 1Ah -- Time of Day Software Interrupt Request Routine
                            C   ;
                            C   ;    Input:   ah = 0   Read the Clock, then:
                            C   ;    Output:  cx =     High Portion of Clock (t_hi_order)
                            C   ;             dx =     Low Portion of Clock (t_low_order)
                            C   ;             al =     1 if 24 hours have elapsed (t_overflow); 0 otherwise
                            C   ;
                            C   ;    Input:   ah = 1   Set the Clock, then:
                            C   ;             cx =     High Portion of Clock (t_hi_order)
                            C   ;             dx =     Low Portion of Clock (t_low_order)
                            C   ;
                            C   ;    Trash:   ah =     (ah - 1) if ah <> 0,-1, or -2
                            C   ;===========================================================================
                            C   ;    Input:   ah = -1  Write Clock Calendar Device, then:
                            C   ;             bx =     day (from 1-1 of leap year up to 12-31 of leap year+7)
                            C   ;                =     (0-2921) = (0-B69h)
                            C   ;             ch =     hour     (0-23)
                            C   ;             cl =     minutes  (0-59)
```

```
                        C  ;    Output:  ah = -1  implies date/time error
                        C  ;             ah = 0   implies date/time OK
                        C  ;
                        C  ;    Input:   ah = -2  Read Clock Calendar Device, then:
                        C  ;    Output:  bx =     day (from 1-1 of leap year up to 12-31 of leap year+7)
                        C  ;             ch =     hour
                        C  ;             cl =     minutes
                        C  ;             dh =     seconds
                        C  ;             dl =     hundredths of seconds
                        C  ;
                        C  ;    Trash:   None.
                        C  ;=============================================================================
                        C
                        C       XORG     0FE6Eh
FE6E                    C1      org      0FE6Eh
                        C
FE6E                    C  t_dayproc     near
                        C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
FE6E  FB                C       sti                                      ; enable interrupts
                        C
FE6F  80 FC FE          C       cmp      ah,0FEh                         ; ZF set if FEh, CF reset if FFh
FE72  75 04             C       jne      t_nFE                           ; ah = -2 = 0FEh ?
                        C
FE74  E8 F8FA R         C       call     c_read                          ; read calendar chip
FE77  CF                C       iret
FE78                    C  t_nFE:
                        C
FE78  72 04             C       jb       t_nFF                           ; ah = -1 = 0FFh > 0FEh ?
                        C
FE7A  E8 F9A0 R         C       call     c_write                         ; set calendar chip
FE7D  CF                C       iret
FE7E                    C  t_nFF:
                        C
                        C       assume   cs:code, ds:data, es:nothing, ss:nothing
                        C
FE7E  1E                C       push     ds                              ; save registers
FE7F  E8 E576 R         C       call     set_ds                          ; satisfy assumptions
                        C
FE82  FA                C       cli                                      ; interrupts off!
                        C                                                ; (shared variables)
                        C
FE83  80 EC 01          C       sub      ah,1                            ; DON'T DECREMENT (CF needed!)
FE86  73 0D             C       jae      t_set                           ; ah = 0 < 1?
                        C
                        C  ; Read Time of Day.
                        C
FE88  32 E4             C       xor      ah,ah                           ; ah = 0 & ZF set!
FE8A  8B 0E 006E R      C       mov      cx,word ptr ds:[t_hi_order]
FE8E  8B 16 006C R      C       mov      dx,word ptr ds:[t_low_order]
FE92  A0 0070 R         C       mov      al,byte ptr ds:[t_overflow]; t_overflow = 0 by setting time!
                        C                                                ; fall through (ah = 0 & ZF set)
                        C
FE95  75 0C             C  t_set:        jnz      t_end                          ; was ah = 1? (is ah = 0 now)?
                        C
```

```
                        C  ; Set Time of Day.
                        C
FE97  89 0E 006E R      C       mov     word ptr ds:[t_hi_order],cx; it's ok, if we fell through.
FE9B  89 16 006C R      C       mov     word ptr ds:[t_low_order],dx        ; (a bit slower, but smaller!)
FE9F  88 26 0070 R      C       mov     byte ptr ds:[t_overflow],ah; ah = 0  (in all cases...)
                        C
FEA3  1F                C  t_end:       pop     ds                                  ; restore registers
FEA4  CF                C       iret
                        C
                        C  t_dayendp
                        C
                        C  ;--------------------------------------------------------------------
                        C  ;    INT 08h -- i8253 p_timer Hardware Interrupt Service Routine
                        C  ;--------------------------------------------------------------------
                        C
                        C       XORG    0FEA5h
FEA5                    C1      org     0FEA5h
                        C
FEA5                    C  t_intproc    near
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C                                                   ; interrupts off!
                        C                                                   ; (shared variables)
                        C
FEA5  50                C       push    ax                                  ; preserve registers
FEA6  52                C       push    dx
FEA7  1E                C       push    ds
                        C
                        C       assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
FEA8  2E: 8E 1E E574 R  C       mov     ds,word ptr cs:[set_ds_word]        ; satisfy assumption
                        C
                        C  ; Handle turning off floppy disk drive motor.
                        C
FEAD  FE 0E 0040 R      C       dec     byte ptr ds:[motor_count]  ; decrement motor on count
FEB1  75 08             C       jnz     t_inc                               ; should we turn off drive?
                        C
FEB3  E8 ED1B R         C       call    stop_disk                  ; if so, stop disk motor
FEB6  80 26 003F R F0   C       and     byte ptr ds:[motor_status],0F0h     ; clear low nibble of status
                        C
FEBB                    C  t_inc:
                        C
                        C  ; Increment long p_timer count
                        C
FEBB  FF 06 006C R      C       inc     word ptr ds:[t_low_order]           ; increment low byte of counter
FEBF  75 04             C       jnz     t_hi                                ; skip t_hi_order
                        C
FEC1  FF 06 006E R      C       inc     word ptr ds:[t_hi_order]   ; increment high byte of counter
FEC5                    C  t_hi:
                        C
                        C  ; Handle 24 hour overflow situation
                        C
FEC5  81 3E 006C R 00B0 C       cmp     word ptr ds:[t_low_order],00B0h     ; has 24 hours elapsed?
FECB  75 14             C       jne     t_ofl                               ; if not, skip t_overflow
                        C
```

```
FECD  83 3E 006E R 18       C       cmp     word ptr ds:[t_hi_order],24; has 24 hours elapsed?
FED2  75 0D                 C       jne     t_ofl                       ; if not, skip t_overflow
                            C
FED4  C6 06 0070 R 01       C       mov     byte ptr ds:[t_overflow],01h
FED9  33 C0                 C       xor     ax,ax
FEDB  A3 006C R             C       mov     word ptr ds:[t_low_order],ax
FEDE  A3 006E R             C       mov     word ptr ds:[t_hi_order],ax
FEE1                        C  t_ofl:
                            C
FEE1  FB                    C       sti                                 ; enable interrupts
                            C                                           ; (no more shared variables)
                            C  ; Invoke any user p_timer break routine.
                            C
FEE2  CD 1C                 C       INT     1Ch
                            C
                            C  ; Send specific end of interrupt (SEOI) to pic 'command' port AFTER p_timer
                            C  ; break, because user may be out-to-lunch for quite awhile....
                            C
FEE4  B0 60                 C       mov     al,pic_seoi_0               ; specific end of interrupt command
FEE6  E6 20                 C       out     pic_0,al            ; to pic 'command port.
                            C
FEE8  1F                    C       pop     ds                          ; restore registers
FEE9  5A                    C       pop     dx
FEEA  58                    C       pop     ax
FEEB  CF                    C       iret
                            C
                            C  t_intendp
                            C
FEEC                        C  code ends

                            C  include vector.asm
                            C
                            C  ;===========================================================================
                            C  ;    Filename:vector.src
                            C  ;
                            C  ;    This module includes the table of ROM interrupt vectors & ill_int
                            C  ;    hardware diagnostic & illegal software interrupt service routine.
                            C  ;
                            C  ;===========================================================================
                            C
FEEC                        C  code segment  public 'ROM'
                            C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                            C
                            C       XORG     0FEF3h
FEF3                        C1      org      0FEF3h
                            C
FEF3                        C  i_vec_tbl     proc     near
                            C
FEF3  FEA5 R                C       dw       t_int                      ; int08locn       see rtc.src
FEF5  E987 R                C       dw       k_int                      ; int09locn       see kb.src
FEF7  FF23 R                C       dw       ill_int                    ; int0Alocn
FEF9  FF23 R                C       dw       ill_int                    ; int0Blocn
FEFB  FF23 R                C       dw       ill_int                    ; int0Clocn
FEFD  FF23 R                C       dw       ill_int                    ; int0Dlocn
FEFF  EF57 R                C       dw       fd_int             ; int0Elocn       see dsk.src
```

```
FF01  FF4B R            C       dw      dummy_iret              ; int0Flocn
                        C
FF03  F065 R            C       dw      v_io                    ; int10locn     see vid.src
FF05  F84D R            C       dw      m_equip                 ; int11locn     see mem.src
FF07  F841 R            C       dw      m_size                  ; int12locn     see mem.src
FF09  EC59 R            C       dw      fd_io                   ; int13locn     see dsk.src
FF0B  E739 R            C       dw      serial_io       ; int14locn     see com.src
FF0D  F859 R            C       dw      m_cass                  ; int15locn     see mem.src
FF0F  E82E R            C       dw      k_io                    ; int16locn     see kb.src
FF11  EFD2 R            C       dw      p_io                    ; int17locn     see prn.src
                        C
FF13  FFD2 R            C       dw      basic_trap              ; int18locn     see int18.src
FF15  F87B R            C       dw      bt_int                  ; int19locn     see boot.src
FF17  FE6E R            C       dw      t_day                   ; int1Alocn     see rtc.src
FF19  FF4B R            C       dw      dummy_iret              ; int1Blocn     see kb.src
FF1B  FF4B R            C       dw      dummy_iret              ; int1Clocn     see rtc.src
FF1D  F0A4 R            C       dw      v_parms                 ; int1Dlocn     see vid.src
FF1F  EFC7 R            C       dw      fd_parms        ; int1Elocn     see dsk.src
FF21  C860 R            C       dw      font_hi_8x8             ; int1Flocn     see graph.src
                        C
                        C  i_vec_tbl    endp
                        C
                        C  ;-------------------------------------------------------------------------
                        C  ;    Interrupt Routine for Unused Hardware & Illegal Software Interrupts
                        C  ;-------------------------------------------------------------------------
                        C
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C       XORG    0FF23h
FF23                    C1      org     0FF23h
                        C
FF23                    C  ill_int      proc    near                            ; trap for illegal interrupts
                        C
FF23  50                C       push    ax                              ; save registers
                        C                                               ; ah = -1; illegal software trap
FF24  B8 FF0B           C       mov     ax,(0FFh*100h)+0Bh      ; al = OCW3 -- read PIC's
FF27  E6 20             C       out     pic_0,al                ; in-service register
                        C
FF29  1E                C       push    ds                              ; save registers & delay
                        C
                        C  ; Determine whether it is a hardware or software interrupt.
                        C
FF2A  E4 20             C       in      al,pic_0                ; get active PIC IR#
FF2C  0A C0             C       or      al,al                           ; are any active?
FF2E  74 03             C       jz      ill_sw                          ; if not, illegal software trap.
FF30  E9 FFDA R         C       jmp     ill_mask                ; if so, illegal hardware int.
                        C
                        C  ; If hardware interrupt, disable the 8259 PIC from further interrupts.
                        C
                        C  ;mov ah,al                           ; return active PIC IR#
                        C  ;in  al,pic_1                ; OCW1 -- get PIC interrupt mask
                        C  ;or  al,ah                           ; shut off (set) IR# bit.
                        C
                        C  ; If hardware interrupt, disable the 8259 PIC from further interrupts.
                        C  ; locate highest priority interrupt and mask only it off
```

```
                        C
FF33                    C  ill_sw:                                          ; illegal software trap; ah = -1
                        C
                        C  ; Turn off floppy disk drives and notify user
                        C
FF33  E8 E50D R         C       call    ill_trap                    ; every register but ds saved!
                        C
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
FF36                    C  ill_flg:                                         ; set illegal trap flag.
                        C                                                   ; illegal software trap; ah = -1
                        C
                        C       assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
FF36  E8 E576 R         C       call    set_ds                              ; satisfy assumptions.
FF39  88 26 006B R      C       mov     byte ptr ds:[intr_flag],ah ; return interrupt flag.
                        C
FF3D  1F                C       pop     ds                                  ; restore registers.
FF3E  58                C       pop     ax                                  ; give user a second chance
FF3F  CF                C       iret
                        C
                        C  ill_int      endp
                        C
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C       XORG    0FF4Bh                              ; ORG 0FF4Bh through 0FF53h
FF4B                    C1      org     0FF4Bh
                        C
FF4B                    C  dummy_iret   proc    near                        ; 'BREAK' key interrupt (1Bh)
FF4B  CF                C       iret                                        ; p_timer break interrupt (1Ch)
                        C
FF4C  CF                C       iret                                        ; 0FF4Ch -- in case someone is
FF4D  CF                C       iret                                  ; 0FF4Dh
FF4E  CF                C       iret                                        ; 0FF4Eh
FF4F  CF                C       iret                                        ; 0FF4Fh
FF50  CF                C       iret                                        ; 0FF50h
FF51  CF                C       iret                                        ; 0FF51h
FF52  CF                C       iret                                        ; 0FF52h
FF53  CF                C       iret                                  ; 0FF53h
                        C
                        C  dummy_iret   endp
                        C
FF54                    C  code ends

                        C  include prnscr.asm
                        C
                        C  ;===========================================================================
                        C  ;    Filename:prnscr.src
                        C  ;
                        C  ;    This module includes INT 05h.
                        C  ;
                        C  ;===========================================================================
                        C
                        C
FF54                    C  code segment  public 'ROM'
```

```
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
                        C ;--------------------------------------------------------------------
                        C ;     INT 05h -- Print Screen
                        C ;
                        C ;     Input:  None.
                        C ;     Output: None.
                        C ;     Trash:  None. (Uses byte at 50:0 = 40:0100 as monitor lock:
                        C ;                                 0 indicates monitor not locked.
                        C ;                                 1 indicates monitor locked.
                        C ;                                -1 indicates printer error.
                        C ;--------------------------------------------------------------------
                        C
                        C       XORG    0FF54h
FF54                    C1      org     0FF54h
                        C
FF54                    C s_intproc     near
                        C       assume  cs:code, ds:nothing, es:nothing, ss:nothing
                        C
FF54  1E                C       push    ds                      ; save ds
                        C
                        C               assume  cs:code, ds:data, es:nothing, ss:nothing
                        C
FF55  2E: 8E 1E E574 R  C       mov     ds,word ptr cs:[set_ds_word] ; satisfy assumptions
                        C
FF5A  52                C       push    dx
FF5B  B2 01             C       mov     dl,1                    ; 1 = locked
FF5D  F0/ 86 16 0100    C lock  xchg    byte ptr ds:[100h],dl   ; check monitor lock
FF62  FE CA             C       dec     dl                      ; already locked ?
FF64  74 4C             C       jz      s_nop                   ; if set, do nothing
FF66  FB                C       sti                             ; enable interrupts after
                        C                                       ; monitor lock code!!!!
                        C
FF67  51                C       push    cx                      ; save registers
FF68  53                C       push    bx
FF69  50                C       push    ax
                        C
                        C ; Get Current Video Width.
                        C
FF6A  B4 0F             C       mov     ah,0Fh                  ; call v_video_state
FF6C  CD 10             C       INT     10h                     ; Output: ah = crt_cols
                        C                                       ;         al = crt_mode
                        C                                       ;         bh = active_page
                        C
FF6E  8A DC             C       mov     bl,ah                   ; save ah = crt_cols
                        C
                        C ; Get Current Cursor Position.
                        C
FF70  B4 03             C       mov     ah,03h                  ; call v_read_cursor
FF72  CD 10             C       INT     10h                     ; Input:  bh = active_page
                        C                                       ; Output:
                        C                                       ; (dh,dl) = (row,col) of cursor
                        C                                       ; (ch,cl) = cursor mode setting
                        C
FF74  8A EB             C       mov     ch,bl                   ;get crt_cols
```

```
FF76  52              C       push    dx                      ; save row, col of cursor
FF77  B1 FF           C       mov     cl,-1                   ; initialize cl = printer error
                      C
                      C  ; Loop Through the Screen.
                      C
FF79  BA 0000         C       mov     dx,0                    ; (dh,dl) = (row,col) of origin
                      C
FF7C  E8 FFB5 R       C       call    s_eol                   ; print a new line
FF7F  75 24           C       jnz     s_err                   ; any errors?
                      C
FF81  E8 FFC9 R       C  s_lp:call    s_get                   ; get next character from screen
                      C
                      C  ; Map Invalid Characters to Space.
                      C
FF84  3C 00           C       cmp     al,0                    ; check validity of character.
FF86  75 02           C       jne     s_ok                    ; if valid, we're ok.
FF88  B0 20           C       mov     al,' '                  ; if invalid, print a space.
FF8A                  C  s_ok:
                      C
                      C  ; Print the Character.
                      C
FF8A  E8 FFBC R       C       call    s_out
FF8D  75 16           C       jnz     s_err                   ; any errors?
                      C
                      C  ; Advance to Next Character.
                      C
FF8F  FE C2           C       inc     dl                      ; advance column (1-crt_cols)
FF91  3A D5           C       cmp     dl,ch                   ; dl < ch = crt_cols?
FF93  7C EC           C       jl      s_lp                    ; if so, continue
                      C
FF95  E8 FFB5 R       C       call    s_eol                   ; else print a new line
FF98  75 0B           C       jnz     s_err                   ; any errors?
                      C
FF9A  32 D2           C       xor     dl,dl                   ; move column back to 0
FF9C  FE C6           C       inc     dh                      ; advance row (1-25)
FF9E  80 FE 19        C       cmp     dh,25                   ; dh < 25
FFA1  7C DE           C       jl      s_lp                    ; if so, continue
                      C
FFA3  33 C9           C       xor     cx,cx                   ; set cl = printer no error
                      C  ;    jmp     short s_err             ; fall through
                      C
FFA5  5A              C  s_err:       pop     dx                      ; restore dx=(row,col) of cursor
FFA6  B4 02           C       mov     ah,02h                  ; call v_set_cpos
FFA8  CD 10           C       INT     10h                     ; Input: bh = active_page
                      C                                       ;        dx =(row,col) of cursor
                      C
FFAA  FB              C       sti                             ; disable interrupts during
                      C                                       ; monitor lock code!!!!
FFAB  88 0E 0100      C       mov     byte ptr ds:[100h],cl   ; reset monitor lock
                      C
FFAF  58              C       pop     ax                      ; restore registers
FFB0  5B              C       pop     bx
FFB1  59              C       pop     cx
FFB2  5A              C  s_nop:       pop     dx
FFB3  1F              C       pop     ds
```

```
FFB4  CF                C       iret
                        C
                        C
FFB5  B0 0A             C  s_eol:       mov     al,LF                   ; print LF & CR
FFB7  E8 FFBC R         C       call    s_out
FFBA  B0 0D             C       mov     al,CR                   ; print CR
                        C  ;    jmp     short s_out             ; fall through
                        C
FFBC                    C  s_out:                                       ; prints out byte in al
FFBC  52                C       push    dx                      ; save dx
FFBD  BA 0000           C       mov     dx,0                    ; address printer port 0.
FFC0  B4 00             C       mov     ah,0                    ; write byte to port 0
FFC2  CD 17             C       INT     17h
FFC4  5A                C       pop     dx                      ; restore dx
                        C  ;    test    ah,025h                 ; test for any errors?
FFC5  F6 C4 01          C       test    ah,001h                 ; test for time out?
FFC8  C3                C       ret
FFC9                    C  s_get:               ; Set Cursor Position and get character @ curs. position
                        C
FFC9  B4 02             C       mov     ah,02h                  ; call v_set_cpos
FFCB  CD 10             C       INT     10h                     ; Input: bh = active_page
                        C                                       ;        dx =(row,col) of cursor
                        C  ; Read Character at Cursor Position.
                        C
FFCD  B4 08             C       mov     ah,08h                  ; call v_read_ac_current
FFCF  CD 10             C       INT     10h                     ; Input:  bh = active_page
                        C                                       ; Output: al = character read
FFD1  C3                C       ret                             ;         ah = attribute
38                        C
                        C  s_intendp
                        C
FFD2                    C  code ends

                        C  include int18.asm
                        C
                        C
                        C  ;===========================================================================
                        C  ;       Filename:       int18.src
                        C  ;
                        C  ;===========================================================================
                        C
FFD2                    C  code segment  public 'ROM'
                        C       assume   cs:code, ds:nothing, es:nothing, ss:nothing
                        C
FFD2                    C  basic_trap    proc     near
                        C
FFD2  BE E6F5 R         C       mov      si,offset trap_mess
FFD5  E8 DFFA R         C       call     DRomString                     ; CS:SI points to string.
FFD8  EB FE             C       jmp      $                              ; loop forever.
                        C
                        C  basic_trap    endp
                        C
                        C
                        C  ;===========================================================================
                        C  ;
```

```
                        C  ; This is part of the illegal interrupt routine, moved here because
                        C  ; we ran out of space in "vector.src".
                        C  ;
                        C  ; This routine is called when it has been determined that there is a
                        C  ; hardware interrupt.  The strategy is to mask off only the offending
                        C  ; interrupt, by finding the highest priority pending interrupt
                        C  ;
                        C  ;---------------------------------------------------------------------
                        C
FFDA                    C  ill_mask proc near
                        C
FFDA  B4 80             C       mov     ah,80H                          ;initial mask (to be shifted)
FFDC                    C  ill_mask_loop:
FFDC  D0 C4             C       rol     ah,1                            ;try next highest priority
FFDE  84 C4             C       test    al,ah                           ;test for match with pic value
FFE0  74 FA             C       jz      ill_mask_loop                   ;if match, we've found it
FFE2  8A C4             C       mov     al,ah                           ;mask off only this interrupt
FFE4  E6 21             C       out     pic_1,al                ; send PIC new mask.
                        C
FFE6  B0 20             C       mov     al,pic_neoi                     ; OCW2 -- send PIC a
FFE8  E6 20             C       out     pic_0,al                ; nonspecific end_of_int
                        C
                        C  ; Return ah = active PIC Interrupt Number in intr_flag.
                        C
FFEA  E9 FF36 R         C       jmp     ill_flg                         ; ill_flg is in vector.src
                        C
                        C  ill_mask endp
                        C
FFED                    C  code ends
0) : error 102: Segment near (or at) 64K limit


                            ;---------------------------------------------------------------------
                            ;    CPU System Reset Vector
                            ;---------------------------------------------------------------------

FFED                        code segment  public 'ROM'
                                assume   cs:code, ds:nothing, es:nothing, ss:nothing

                                XORG     0FFF0h             ; F000:FFF0 = FFFF0 = FFFF:0000
FFF0                        1   org      0FFF0h

FFF0                        vector                proc     near

FFF0  EA                        db       0EAh               ; jmp intersegment  F000:(offset diagnostics_1)
FFF1  E05B R                    dw       i_hard_reset       ; instruction pointer cs:(offset i_hard_reset)
FFF3  F000                      dw       code_seg ; code segment             F000h

FFF5  30 34 2F 30 33 2F 38      db       '04/03/86'         ; release marker (exactly 8 bytes!!!!)
      36
FFFD  41                        db       'A'                ; this is the secondary ID byte (1050=A)

                            ;XORG0FFFEh
FFFE  00                    rom_id                db       0          ; ROM identifier (0 means M24.)
```

```
                        ; chk_hi      db      0       ; space for checksum of F000:E000 to F000:FFFF

                        vector                endp

FFFF                    code ends
ASM(211) : error 102: Segment near (or at) 64K limit

                            end
Macros:

                N a m e                 Lines

JMPF . . . . . .        3
XORG . . . . . .        9

Structures and Records:

                N a m e             Width   # fields
                                    Shift   Width   Mask    Initial

PARM_TBL . . . . . .    0010    000A
  P_CYL . . . . . .     0000
  P_HEADS . . . . . .   0002
  P_WRITE_CUR . . . . . .   0003
  P_PRECOMP . . . . . .     0005
  P_ECC_LEN . . . . . .     0007
  P_CONTROL_BYTE . . . . .  0008
  P_TIMEOUT . . . . . .     0009
  P_FMT_TIMEOUT . . . . .   000A
  P_DRVDIAG_TIMEOUT . . . . .   000B
  P_ZZJ . . . . . .     000C

Segments and Groups:

                N a m e         Size    Align   Combine Class

ABSO . . . . . .        0080    PARA    PUBLIC  'RAM'
CODE . . . . . .        FFFF    PARA    PUBLIC  'ROM'
DATA . . . . . .        0099    PARA    PUBLIC  'RAM'
STACK_RAM . . . . . .   0000    PARA    PUBLIC  'RAM'
V_RAM . . . . . .       0000    PARA    PUBLIC  'RAM'

Symbols:

                N a m e         Type    Value   Attr

A1_BX . . . . . .       Number  000E
A1_CX . . . . . .       Number  000C
A1_DX . . . . . .       Number  000A
ABSO_SEG . . . . . .    Number  0000
ADDR_MARK_ERROR . . . . .   Number  0002
ALT_INPUT . . . . . .   L BYTE  0019    DATA
ALT_KEY . . . . . .     Number  0038
ALT_RET . . . . . .     L WORD  E238    CODE
ALT_SHIFT . . . . . .   Number  0008
```

```
BANNER_M . . . . . L BYTE  D9CA  CODE
BASIC_TRAP . . . . . N PROC  FFD2  CODE    Length = 0008
BC_BAD . . . . . L NEAR  D285  CODE
BEL . . . . . Number  0007
BELL_WAIT . . . . . L NEAR  F61F  CODE
BIOS_BREAK . . . . . L BYTE  0071  DATA
BOOT_BASIC . . . . . L NEAR  D230  CODE
BOOT_INDIRECT . . . . . L NEAR  D237  CODE
BOOT_SUCC . . . . . L NEAR  D232  CODE
BR_TBL . . . . . L WORD  D303  CODE
BS . . . . . Number  0008
BT_AGAIN . . . . . L NEAR  F8CD  CODE
BT_BLNK . . . . . L NEAR  F8D7  CODE
BT_DEC . . . . . L NEAR  F8DD  CODE
BT_I . . . . . L NEAR  F89F  CODE
BT_INT . . . . . N PROC  F87B  CODE    Length = 007F
BT_JMP . . . . . N PROC  E6F2  CODE    Length = 0003
BT_M . . . . . L BYTE  D9EC  CODE
BT_MERR . . . . . L BYTE  DA06  CODE
BT_NXT . . . . . L NEAR  F8B5  CODE
BT_O . . . . . L NEAR  F88C  CODE
BT_OK . . . . . L NEAR  F8E8  CODE
BT_SPACES . . . . . L BYTE  DA2B  CODE
BUFFER_END . . . . . L WORD  0082  DATA
BUFFER_HEAD . . . . . L WORD  001A  DATA
BUFFER_START . . . . . L WORD  0080  DATA
BUFFER_TAIL . . . . . L WORD  001C  DATA
BUFF_IO . . . . . L NEAR  D40A  CODE
BUSY . . . . . L NEAR  D56D  CODE
BUSY_WAIT . . . . . L NEAR  D560  CODE

CAPS_LOCK_KEY . . . . . Number  003A
CAPS_LOCK_MODE . . . . . Number  0040
CAPS_LOCK_SHIFT . . . . . Number  0040
CCB_BLKS . . . . . Number  0004
CCB_BYTE . . . . . L NEAR  D578  CODE
CCB_CMD . . . . . Number  0000
CCB_DRV_B . . . . . Number  0020
CCB_OPT . . . . . Number  0005
CCB_SEND . . . . . L NEAR  D54A  CODE
CC_BUSY . . . . . L NEAR  D594  CODE
CC_ER . . . . . Number  0002
CHK_ANY_DISKS . . . . . L NEAR  D189  CODE
CHK_LO . . . . . L BYTE  C000  CODE
CMD_BLOCK . . . . . L BYTE  0042  DATA
CMD_DONE . . . . . L NEAR  D28C  CODE
CMD_ERROR . . . . . Number  0001
CNTRL_KEY . . . . . Number  001D
CNTRL_SHIFT . . . . . Number  0004
CODE_SEG . . . . . Number  F000
COLOR_POINTER . . . . . Number  03D4
COMMAND_BR . . . . . N PROC  D2B3  CODE    Length = 032B
COMMCONTROL . . . . . Number  0065
COM_BAUD . . . . . L WORD  E729  CODE
COM_CTS . . . . . Number  0010
```

```
COM_DATA1 . . . . . .  N PROC  E729  CODE  Length = 0010
COM_DATA_A . . . . . .  Number  03F8
COM_DATA_B . . . . . .  Number  02F8
COM_DSR . . . . . .  Number  0020
COM_DTR . . . . . .  Number  0001
COM_FE . . . . . .  Number  0008
COM_GB . . . . . .  N PROC  E92B  CODE  Length = 0023
COM_ID_A . . . . . .  Number  03FA
COM_ID_B . . . . . .  Number  02FA
COM_INIT . . . . . .  N PROC  E78F  CODE  Length = 003A
COM_OE . . . . . .  Number  0002
COM_PB . . . . . .  N PROC  E8F0  CODE  Length = 002C
COM_PE . . . . . .  Number  0004
COM_RTS . . . . . .  Number  0002
COM_RXD . . . . . .  Number  0001
COM_STAT . . . . . .  L NEAR  E881  CODE
COM_TE . . . . . .  Number  0080
COM_TXD . . . . . .  Number  0020
CONTROL . . . . . .  L WORD  0089  DATA
CONTROLC . . . . . .  Number  0062
CONTROL_BYTE . . . . . .  L BYTE  0076  DATA
CR . . . . . .  Number  000D
CRC_ERROR . . . . . .  Number  0010
CTLR_INIT . . . . . .  L NEAR  D10C  CODE
CTLR_MISSING . . . . . .  L NEAR  D362  CODE
C_BCD2HEX . . . . . .  N PROC  F99C  CODE  Length = 0004
C_DATA1 . . . . . .  N PROC  E95B  CODE  Length = 001C
C_DY_MO . . . . . .  L BYTE  E96B  CODE
C_DY_YR . . . . . .  L WORD  E95B  CODE
C_GDAYS . . . . . .  N PROC  FA2A  CODE  Length = 0013
C_GRET . . . . . .  L NEAR  FA3C  CODE
C_RBCD . . . . . .  N PROC  F969  CODE  Length = 0025
C_RBLP . . . . . .  L NEAR  F981  CODE
C_RBRET . . . . . .  L NEAR  F98C  CODE
C_RDYR . . . . . .  L NEAR  F901  CODE
C_READ . . . . . .  N PROC  F8FA  CODE  Length = 006F
C_READ_EXIT . . . . . .  L NEAR  F967  CODE
C_RHDC . . . . . .  L NEAR  F99F  CODE
C_RHEX . . . . . .  N PROC  F98E  CODE  Length = 000E
C_RMO . . . . . .  L NEAR  F932  CODE
C_RMLP . . . . . .  L NEAR  F92A  CODE
C_RTCCHK . . . . . .  L NEAR  F97E  CODE
C_WERR . . . . . .  L NEAR  FA18  CODE
C_WHEX . . . . . .  N PROC  FA19  CODE  Length = 0011
C_WMLP . . . . . .  L NEAR  F9E0  CODE
C_WRITE . . . . . .  N PROC  F9A0  CODE  Length = 0079
C_WYLP . . . . . .  L NEAR  F9CB  CODE

DATA_SEG . . . . . .  Number  0040
DCOLON . . . . . .  N PROC  E589  CODE  Length = 000C
DCRLF . . . . . .  N PROC  E57C  CODE  Length = 000D
DC_TBL . . . . . .  L BYTE  D32D  CODE
DELETE_KEY . . . . . .  Number  0053
DETT_BOOT . . . . . .  L NEAR  D1D4  CODE
DETT_BOOT_END . . . . . .  L NEAR  D1F1  CODE
```

DETT_BOOT_NXT	L NEAR	D1EA	CODE	
DHEXBOK	L NEAR	E5B5	CODE	
DHEXBYTE	N PROC	E5AD	CODE	Length = 0014
DHEXLONG	N PROC	E595	CODE	Length = 0011
DHEXNIB	N PROC	E5C1	CODE	Length = 0015
DHEXOK	L NEAR	E59D	CODE	
DHEXWORD	N PROC	E5A6	CODE	Length = 0007
DIAG1_END	L NEAR	DF7E	CODE	
DIAGNOSTICS_1	N PROC	DB8F	CODE	Length = 03F7
DISABLE_NMI_FLAG	L BYTE	0098	DATA	
DISABLE_PARITY	N PROC	E666	CODE	Length = 0009
DISKETTE_IO1	L NEAR	EC86	CODE	
DISKETTE_IO2	L NEAR	EC89	CODE	
DISKETTE_STATUS	L BYTE	0041	DATA	
DISK_STATUS	L BYTE	0074	DATA	
DISP_PASS	L NEAR	DD64	CODE	
DLX_KB	Number	0001		
DMA_64K	L NEAR	D466	CODE	
DMA_ADDR_0	Number	0000		
DMA_ADDR_1	Number	0002		
DMA_ADDR_2	Number	0004		
DMA_ADDR_3	Number	0006		
DMA_CMD_DISABLE	Number	0004		
DMA_CMD_ENABLE	Number	0000		
DMA_COMMAND	Number	0008		
DMA_COUNT_0	Number	0001		
DMA_COUNT_1	Number	0003		
DMA_COUNT_2	Number	0005		
DMA_COUNT_3	Number	0007		
DMA_ERROR	Number	0008		
DMA_FF_CLR	Number	000C		
DMA_MASK_BIT	Number	000A		
DMA_MASK_CLR	Number	000E		
DMA_MASK_WRITE	Number	000F		
DMA_MASTER_CLR	Number	000D		
DMA_MODE	Number	000B		
DMA_MODE_0	Number	0058		
DMA_MODE_1	Number	0041		
DMA_MODE_2	Number	0056		
DMA_MODE_3	Number	0043		
DMA_NO	L NEAR	D469	CODE	
DMA_REQUEST	Number	0009		
DMA_SEGM_0	Number	0080		
DMA_SEGM_1	Number	0082		
DMA_SEGM_2	Number	0081		
DMA_SEGM_3	Number	0083		
DMA_SEG_ERROR	Number	0009		
DMA_START	L NEAR	D41B	CODE	
DMA_STATUS	Number	0008		
DMA_TEMP	Number	000D		
DMA_UNMASK_0	Number	0000		
DNUM	N PROC	E5D6	CODE	Length = 0008
DNUMW	N PROC	E5DE	CODE	Length = 0036
DNUMWEXIT	L NEAR	E612	CODE	
DNUMW_LOOP	L NEAR	E5ED	CODE	

```
DNUMW_SKIP . . . . . L NEAR  E603   CODE
DNUMW_SPACES . . . . . L NEAR  E5FB   CODE
DROMEXIT . . . . . L NEAR  E007   CODE
DROMSTRING . . . . . N PROC  DFFA   CODE    Length = 000F
DRVN_OK . . . . . L NEAR  D289   CODE
DRV_1 . . . . . L NEAR  D5CB   CODE
DRV_RDY . . . . . L NEAR  D149   CODE
DSTRING . . . . . N PROC  E009   CODE    Length = 0017
DS_LP . . . . . L NEAR  E010   CODE
DS_RET . . . . . L NEAR  E01B   CODE
DUMMY_IRET . . . . . N PROC  FF4B   CODE    Length = 0009
D_C_2 . . . . . L NEAR  D1B2   CODE

EGA_SWITCHES . . . . . L BYTE  0088   DATA
ENABLE_PARITY . . . . . N PROC  E614   CODE    Length = 0052
ENABLE_RAM_EXIT . . . . . L NEAR  E68B   CODE
ENABLE_RAM_PARITY . . . . . N PROC  E66F   CODE    Length = 001E
ER_MASTER_TBL . . . . . L BYTE  D5DF   CODE

FAIL_M . . . . . L BYTE  DA7A   CODE
FAR_CALLS . . . . . F PROC  C004   CODE    Length = 005C
FDC_ERROR . . . . . Number  0020
FDU_DATA1 . . . . . N PROC  EE10   CODE    Length = 0009
FDU_DATA2 . . . . . N PROC  EFC7   CODE    Length = 000B
FD_INT . . . . . N PROC  EF57   CODE    Length = 0014
FD_IO . . . . . F PROC  EC59   CODE    Length = 006D
FD_PARMS . . . . . L BYTE  EFC7   CODE
FLAGS_DATA1 . . . . . N PROC  C000   CODE    Length = 0004
FONTHI8 . . . . . N PROC  C860   CODE    Length = 0400
FONTLO16 . . . . . N PROC  C060   CODE    Length = 0800
FONTLO8 . . . . . N PROC  FA6E   CODE    Length = 0400
FONT_HI_8X8 . . . . . L BYTE  C860   CODE
FONT_LO_8X16 . . . . . L BYTE  C060   CODE
FONT_LO_8X8 . . . . . L BYTE  FA6E   CODE
F_BUFOFF . . . . . Text  [bp+4]
F_CHECK_VALID . . . . . N PROC  E805   CODE    Length = 001A
F_COMMAND . . . . . Text  [bp+3]
F_CONT . . . . . L NEAR  EF16   CODE
F_CV_RET . . . . . L NEAR  E81D   CODE
F_CYL . . . . . Text  [bp+7]
F_DRIVE . . . . . Text  [bp+0]
F_FORMAT_CMD . . . . . Number  004D
F_GB_DECODE . . . . . L NEAR  EE55   CODE
F_GB_JMP . . . . . L NEAR  EE5D   CODE
F_GB_LOOP . . . . . L NEAR  EE24   CODE
F_GB_LOOP1 . . . . . L NEAR  EE4E   CODE
F_GB_OUT . . . . . L NEAR  EE34   CODE
F_GB_RET . . . . . L NEAR  EE61   CODE
F_GB_TABLE . . . . . L BYTE  EE10   CODE
F_GET_BYTE . . . . . N PROC  EE19   CODE    Length = 004A
F_GET_VAR . . . . . N PROC  EDEF   CODE    Length = 000D
F_HEAD . . . . . Text  [bp+1]
F_HEAD_SETTLE . . . . . L NEAR  EF31   CODE
F_HLT . . . . . Number  0001
F_HUT . . . . . Number  000F
```

```
F_IO1 . . . . . L NEAR  EC7F    CODE
F_IO_RET . . . . . L NEAR  ECA1    CODE
F_MOTOR_ON . . . . . N PROC  E7E0    CODE    Length = 0025
F_MOTOR_PORT . . . . . Number  03F2
F_MOTOR_WAIT . . . . . Number  0025
F_MO_RET . . . . . L NEAR  E804    CODE
F_NDMA . . . . . Number  0000
F_NEC_DATA . . . . . Number  03F5
F_NEC_RDY . . . . . N PROC  EF6B    CODE    Length = 0017
F_NEC_RESET . . . . . N PROC  E7C9    CODE    Length = 0017
F_NEC_RESET_RET . . . . . L NEAR  E7DF    CODE
F_NEC_STATUS . . . . . Number  03F4
F_NR1 . . . . . L NEAR  EF6E    CODE
F_NR_RET . . . . . L NEAR  EF80    CODE
F_NUMSECS . . . . . Text  [bp+2]
F_PB_ERRET . . . . . L NEAR  EE08    CODE
F_PB_RET . . . . . L NEAR  EE07    CODE
F_PUT_BYTE . . . . . N PROC  EDFC    CODE    Length = 0014
F_R1 . . . . . L NEAR  ECDF    CODE
F_R2 . . . . . L NEAR  ECE7    CODE
F_RD1 . . . . . L NEAR  ED69    CODE
F_RDATA . . . . . N PROC  ED4E    CODE    Length = 0029
F_RD_LOOP . . . . . L NEAR  ED64    CODE
F_READ_CMD . . . . . Number  00E6
F_REAL_DRIVE . . . . . Text  [bp+8]
F_RECAL_CMD . . . . . Number  0007
F_RESET . . . . . N PROC  ECC6    CODE    Length = 0055
F_RW1 . . . . . L NEAR  EDA3    CODE
F_RW2 . . . . . L NEAR  EDCD    CODE
F_RW3 . . . . . L NEAR  EDE4    CODE
F_RW_COMMON . . . . . N PROC  ED77    CODE    Length = 0078
F_RW_RET . . . . . L NEAR  EDEC    CODE
F_RW_SKIP . . . . . L NEAR  EDC7    CODE
F_S1 . . . . . L NEAR  EEF2    CODE
F_SD1 . . . . . L NEAR  EE82    CODE
F_SD2 . . . . . L NEAR  EEAD    CODE
F_SD_RET . . . . . L NEAR  EEBC    CODE
F_SECNUM . . . . . Text  [bp+6]
F_SEEK . . . . . N PROC  EEBD    CODE    Length = 007A
F_SEEK_CMD . . . . . Number  000F
F_SET_DMA . . . . . N PROC  EE63    CODE    Length = 005A
F_SIS . . . . . N PROC  EF82    CODE    Length = 000B
F_SNSDRV_CMD . . . . . Number  0004
F_SNSINT_CMD . . . . . Number  0008
F_SPECIFY_CMD . . . . . Number  0003
F_SRT_48 . . . . . Number  000C
F_SRT_96 . . . . . Number  000E
F_S_RECAL . . . . . L NEAR  EED1    CODE
F_S_RET . . . . . L NEAR  EF36    CODE
F_TABLE . . . . . L WORD  EC95    CODE
F_WAIT_FOR_NEC . . . . . N PROC  EF8D    CODE    Length = 0020
F_WAIT_ONE_MS . . . . . N PROC  EF37    CODE    Length = 0008
F_WD1 . . . . . L NEAR  ED45    CODE
F_WDATA . . . . . N PROC  ED22    CODE    Length = 002C
F_WD_LOOP . . . . . L NEAR  ED40    CODE
```

```
F_WRITE_CMD . . . . . Number  00C5

GAME_CARD . . . . . Number  0201
GRF_GRAPHICS_DOWN . . . . . N PROC  D738  CODE  Length = 0063
GRF_GRAPHICS_READ . . . . . N PROC  D79B  CODE  Length = 00FC
GRF_GRAPHICS_UP . . . . . N PROC  D6A3  CODE  Length = 0055
GRF_GRAPHICS_WRITE . . . . . N PROC  D897  CODE  Length = 0106
GRF_LIGHT_PEN . . . . . N PROC  F633  CODE  Length = 0098
GRF_READ_DOT . . . . . N PROC  D5FF  CODE  Length = 001A
GRF_WRITE_DOT . . . . . N PROC  D619  CODE  Length = 002B
G_72 . . . . . L NEAR  D9AD  CODE
G_8X16_2 . . . . . L NEAR  D866  CODE
G_8X8_2 . . . . . L NEAR  D8BA  CODE
G_ADDR . . . . . N PROC  D644  CODE  Length = 005F
G_ADDR_TEST . . . . . L NEAR  D8C2  CODE
G_ALIGN_DOT . . . . . L NEAR  D637  CODE
G_BITMASK . . . . . L NEAR  D691  CODE
G_CHAR_LP . . . . . L NEAR  D952  CODE
G_CMP_MOD . . . . . L NEAR  D75E  CODE
G_COLOR_TABLE . . . . . L BYTE  D999  CODE
G_CURS_OFF . . . . . N PROC  D99D  CODE  Length = 001E
G_DETMODE . . . . . L NEAR  D8E5  CODE
G_EXP_BYT . . . . . L NEAR  D964  CODE
G_FILLER . . . . . N PROC  D71A  CODE  Length = 001E
G_F_CONT . . . . . L NEAR  D83E  CODE
G_F_EXIT . . . . . L NEAR  D88F  CODE
G_F_I_LP . . . . . L NEAR  D71C  CODE
G_F_MACH . . . . . L NEAR  D843  CODE
G_F_NOT_FOUND . . . . . L NEAR  D87C  CODE
G_F_RV_CONT . . . . . L NEAR  D83A  CODE
G_F_S_LP . . . . . L NEAR  D71E  CODE
G_HI_WR . . . . . L NEAR  D909  CODE
G_IA_LP . . . . . L NEAR  D95D  CODE
G_I_A_LP . . . . . L NEAR  D919  CODE
G_JSFY_DOT . . . . . L NEAR  D614  CODE
G_LDS_R . . . . . L NEAR  D7BB  CODE
G_LDS_W . . . . . L NEAR  D8DD  CODE
G_LINELP . . . . . L NEAR  D915  CODE
G_MATCHB . . . . . L NEAR  D836  CODE
G_MEDGET . . . . . L NEAR  D801  CODE
G_MED_BIT . . . . . L NEAR  D81A  CODE
G_MED_IA . . . . . L NEAR  D806  CODE
G_MED_STORE . . . . . L NEAR  D979  CODE
G_MED_WR . . . . . L NEAR  D941  CODE
G_M_AREA . . . . . L NEAR  D6FB  CODE
G_RDLOOP . . . . . L NEAR  D7DB  CODE
G_RD_IA . . . . . L NEAR  D7DF  CODE
G_RD_MED . . . . . L NEAR  D7FF  CODE
G_REPCHAR . . . . . L NEAR  D90E  CODE
G_RETURN . . . . . L NEAR  D995  CODE
G_SCAN_LP . . . . . L NEAR  D958  CODE
G_SCROLLER . . . . . N PROC  D6F8  CODE  Length = 0022
G_SELFONT . . . . . L NEAR  D8CF  CODE
G_SETDOWN . . . . . L NEAR  D76C  CODE
G_SET_UP . . . . . L NEAR  D6D8  CODE
```

```
G_SKP_1 . . . . . L NEAR  D655  CODE
G_SKP_2 . . . . . L NEAR  D65B  CODE
G_SKP_3 . . . . . L NEAR  D661  CODE
G_SKP_4 . . . . . L NEAR  D684  CODE
G_SKP_5 . . . . . L NEAR  D6A2  CODE
G_SUPER_WR . . . . . L NEAR  D905  CODE
G_TEST_ADDR . . . . . L NEAR  D872  CODE
G_TINYTEXT . . . . . L NEAR  D901  CODE
G_TST_MOD . . . . . L NEAR  D6C9  CODE
G_T_XOR . . . . . L NEAR  D921  CODE
G_UNREVERSE_VIDEO_LOOP . . . . . L NEAR  D887  CODE
G_W_BYTE . . . . . L NEAR  D928  CODE
G_XORBIT . . . . . L NEAR  D62E  CODE

HC_BF_R . . . . . Number  000E
HC_BF_W . . . . . Number  000F
HC_DG_C . . . . . Number  00E4
HC_DG_D . . . . . Number  00E3
HC_DG_R . . . . . Number  00E0
HC_ECC . . . . . Number  000D
HC_FBT . . . . . Number  0007
HC_FD . . . . . Number  0004
HC_FT . . . . . Number  0006
HC_P_S . . . . . Number  000C
HC_RCL . . . . . Number  0001
HC_RD . . . . . Number  0008
HC_RDL . . . . . Number  00E5
HC_RDY . . . . . Number  0000
HC_SK . . . . . Number  000B
HC_STAT . . . . . Number  0003
HC_VR . . . . . Number  0005
HC_WR . . . . . Number  000A
HC_WRL . . . . . Number  00E6
HDU_PARM_TBL . . . . . L NEAR  CF9F  CODE
HD_CFG . . . . . Number  0002
HD_ERROR . . . . . L BYTE  0042  DATA
HD_MSK . . . . . Number  0003
HD_RS . . . . . Number  0001
HD_SL . . . . . Number  0002
HD_ST . . . . . Number  0001
HF_NUM . . . . . L BYTE  0075  DATA
H_BAD_TRK . . . . . Number  000B
H_BF_R . . . . . Number  000E
H_BF_W . . . . . Number  000F
H_BOOT . . . . . F PROC  D1B3  CODE  Length = 0088
H_BUSY . . . . . Number  0008
H_CC . . . . . Number  0001
H_CD . . . . . Number  0004
H_DG_C . . . . . Number  0014
H_DG_D . . . . . Number  0013
H_DG_R . . . . . Number  0012
H_DMA_3 . . . . . Number  0003
H_DMA_R . . . . . Number  004B
H_DMA_S . . . . . Number  0004
H_DMA_W . . . . . Number  0047
```

```
H_DRQ . . . . . Number  0010
H_ECC_COR . . . . . Number  0011
H_ERC . . . . . Number  0018
H_ERRORS . . . . . N PROC  D5DF  CODE  Length = 0020
H_FD . . . . . Number  0007
H_FMT . . . . . N PROC  D5DE  CODE  Length = 0001
H_FT . . . . . Number  0005
H_FTB . . . . . Number  0006
H_INIT . . . . . N PROC  D09F  CODE  Length = 0114
H_INITZ . . . . . Number  0007
H_INT . . . . . N PROC  D23B  CODE  Length = 0011
H_INTP . . . . . Number  0020
H_IO . . . . . F PROC  D24C  CODE  Length = 0067
H_IO? . . . . . Number  0002
H_MKDMA . . . . . Number  0001
H_MKINT . . . . . Number  0002
H_NO_ERR . . . . . Number  0000
H_OCW1_M0 . . . . . Number  0001
H_OCW1_M5 . . . . . Number  0020
H_PARAMS . . . . . N PROC  CF9F  CODE  Length = 0100
H_P_R . . . . . Number  0008
H_P_S . . . . . Number  0009
H_RCL . . . . . Number  0011
H_RD . . . . . Number  0002
H_RDL . . . . . Number  000A
H_RDY . . . . . Number  0010
H_RESET . . . . . Number  0005
H_RQS . . . . . Number  0001
H_RST . . . . . Number  0000
H_RST_1 . . . . . Number  000D
H_SK . . . . . Number  000C
H_STATUS . . . . . Number  00FF
H_UNDEFD . . . . . Number  00BB
H_VR . . . . . Number  0004
H_WR . . . . . Number  0003
H_WRL . . . . . Number  000B

I13_BUFF_RD . . . . . L NEAR  D408  CODE
I13_BUFF_WR . . . . . L NEAR  D40E  CODE
I13_CC . . . . . L NEAR  D3F2  CODE
I13_PAR_RD . . . . . L NEAR  D3CB  CODE
I13_PAR_WR . . . . . L NEAR  D365  CODE
I13_RD . . . . . L NEAR  D412  CODE
I13_RDL . . . . . L NEAR  D3FB  CODE
I13_RESET . . . . . L NEAR  D342  CODE
I13_WR . . . . . L NEAR  D3F7  CODE
I13_WRL . . . . . L NEAR  D404  CODE
ILL_FLG . . . . . L NEAR  FF36  CODE
ILL_INT . . . . . N PROC  FF23  CODE  Length = 001D
ILL_LN . . . . . L NEAR  E545  CODE
ILL_LP . . . . . L NEAR  E54A  CODE
ILL_M1 . . . . . L BYTE  DA50  CODE
ILL_M2 . . . . . L BYTE  DA69  CODE
ILL_M3 . . . . . L BYTE  DA6F  CODE
ILL_MASK . . . . . N PROC  FFDA  CODE  Length = 0013
```

```
ILL_MASK_LOOP . . . . . L NEAR  FFDC  CODE
ILL_SW . . . . . L NEAR  FF33  CODE
ILL_TRAP . . . . . N PROC  E50D  CODE    Length = 0049
INSERT_KEY . . . . . Number  0052
INSERT_MODE . . . . . Number  0080
INSERT_SHIFT . . . . . Number  0080
INT00LOCN . . . . . L DWORD  0000  ABS0
INT01LOCN . . . . . L DWORD  0004  ABS0
INT02LOCN . . . . . L DWORD  0008  ABS0
INT03LOCN . . . . . L DWORD  000C  ABS0
INT04LOCN . . . . . L DWORD  0010  ABS0
INT05LOCN . . . . . L DWORD  0014  ABS0
INT06LOCN . . . . . L DWORD  0018  ABS0
INT07LOCN . . . . . L DWORD  001C  ABS0
INT08LOCN . . . . . L DWORD  0020  ABS0
INT09LOCN . . . . . L DWORD  0024  ABS0
INT0ALOCN . . . . . L DWORD  0028  ABS0
INT0BLOCN . . . . . L DWORD  002C  ABS0
INT0CLOCN . . . . . L DWORD  0030  ABS0
INT0DLOCN . . . . . L DWORD  0034  ABS0
INT0ELOCN . . . . . L DWORD  0038  ABS0
INT0FLOCN . . . . . L DWORD  003C  ABS0
INT10LOCN . . . . . L DWORD  0040  ABS0
INT11LOCN . . . . . L DWORD  0044  ABS0
INT12LOCN . . . . . L DWORD  0048  ABS0
INT13LOCN . . . . . L DWORD  004C  ABS0
INT14LOCN . . . . . L DWORD  0050  ABS0
INT15 . . . . . F PROC  E68D  CODE    Length = 0006
INT15A . . . . . L NEAR  E68D  CODE
INT15LOCN . . . . . L DWORD  0054  ABS0
INT16LOCN . . . . . L DWORD  0058  ABS0
INT17LOCN . . . . . L DWORD  005C  ABS0
INT18DATA . . . . . N PROC  E6F5  CODE    Length = 0034
INT18LOCN . . . . . L DWORD  0060  ABS0
INT19LOCN . . . . . L DWORD  0064  ABS0
INT1ALOCN . . . . . L DWORD  0068  ABS0
INT1BLOCN . . . . . L DWORD  006C  ABS0
INT1CLOCN . . . . . L DWORD  0070  ABS0
INT1DLOCN . . . . . L DWORD  0074  ABS0
INT1ELOCN . . . . . L DWORD  0078  ABS0
INT1FLOCN . . . . . L DWORD  007C  ABS0
INTR_FLAG . . . . . L BYTE  006B  DATA
INT_WAIT . . . . . L NEAR  D4B2  CODE
IO_LONG . . . . . L NEAR  D3FD  CODE
IO_NORM . . . . . L NEAR  D414  CODE
IO_ROM_INIT . . . . . L WORD  0067  DATA
IO_ROM_SEG . . . . . L WORD  0069  DATA
I_ALT_CONT . . . . . L NEAR  E422  CODE
I_ALT_CPU . . . . . L NEAR  E409  CODE
I_ALT_ECHO . . . . . L NEAR  E466  CODE
I_ALT_END . . . . . L NEAR  E499  CODE
I_ALT_FOUND . . . . . L NEAR  E443  CODE
I_ALT_INQ . . . . . L NEAR  E454  CODE
I_ALT_RESTART . . . . . L NEAR  E477  CODE
I_ALT_SELECT_M . . . . . L BYTE  DB6A  CODE
```

```
I_ALT_TEST . . . . . L NEAR  E438  CODE
I_CAL  . . . . . L NEAR  DE59  CODE
I_CAL_0  . . . . . L NEAR  DEAE  CODE
I_CAL_1_1_80 . . . . . L NEAR  DE7E  CODE
I_CAL_END  . . . . . L NEAR  DEDB  CODE
I_CAL_ERR  . . . . . L NEAR  DEC1  CODE
I_CAL_MAX  . . . . . L NEAR  DE8D  CODE
I_CAL_OK . . . . . L NEAR  DED3  CODE
I_CAL_VAL  . . . . . L BYTE  DB86  CODE
I_COM_M  . . . . . L BYTE  DB12  CODE
I_CPU  . . . . . L NEAR  DB97  CODE
I_CPU_ERR  . . . . . L NEAR  DBC2  CODE
I_CPU_M  . . . . . L BYTE  DA80  CODE
I_CPU_OK . . . . . L NEAR  DBD5  CODE
I_DMAC . . . . . L NEAR  DC1D  CODE
I_DMAC_ERR . . . . . L NEAR  DC95  CODE
I_DMAC_LP  . . . . . L NEAR  DC2D  CODE
I_DMAC_M . . . . . L BYTE  DAA7  CODE
I_DMAC_NIB . . . . . L NEAR  DC75  CODE
I_DMAC_OK  . . . . . L NEAR  DCA4  CODE
I_DMAC_PASS2 . . . . . L NEAR  DC2A  CODE
I_DMAC_RET . . . . . L NEAR  DC88  CODE
I_DMAT . . . . . L NEAR  DC00  CODE
I_DMAT_ERR . . . . . L NEAR  DC0E  CODE
I_DMAT_M . . . . . L BYTE  DA9A  CODE
I_DMAT_OK  . . . . . L NEAR  DC16  CODE
I_DMAT_RET . . . . . L NEAR  DC0C  CODE
I_D_80X25  . . . . . L NEAR  E0E9  CODE
I_D_CK_EGA . . . . . L NEAR  E0C2  CODE
I_D_COLOR  . . . . . L NEAR  E102  CODE
I_D_E  . . . . . L NEAR  DC9F  CODE
I_D_EGA_ROM_LOOP . . . . . L NEAR  E0C7  CODE
I_D_INIT . . . . . N PROC  E05E  CODE    Length = 00B1
I_D_M  . . . . . L BYTE  DAC1  CODE
I_D_MODE . . . . . L NEAR  E109  CODE
I_D_MONO . . . . . L NEAR  E0AA  CODE
I_D_NO_EGA . . . . . L NEAR  E0D6  CODE
I_D_NXT_ROM  . . . . . L NEAR  E0DA  CODE
I_D_OK . . . . . L NEAR  E0F3  CODE
I_D_RET  . . . . . L NEAR  E10D  CODE
I_FATAL  . . . . . N PROC  DF86  CODE    Length = 0068
I_FATAL_RET  . . . . . L NEAR  DFDF  CODE
I_FDUA_M . . . . . L BYTE  DB36  CODE
I_FDUB_M . . . . . L BYTE  DB43  CODE
I_FDU_END  . . . . . L NEAR  E406  CODE
I_FDU_LP . . . . . L NEAR  E3E0  CODE
I_FDU_NOT_M  . . . . . L BYTE  DB50  CODE
I_FDU_OK . . . . . L NEAR  E403  CODE
I_FDU_RDY_M  . . . . . L BYTE  DB54  CODE
I_HARD_RESET . . . . . N PROC  E05B  CODE    Length = 0003
I_HDU_M  . . . . . L BYTE  DB5D  CODE
I_HDU_OK . . . . . L NEAR  E392  CODE
I_INIT_END . . . . . L NEAR  E49E  CODE
I_KB_M . . . . . L BYTE  DAF4  CODE
I_KB_ST_M  . . . . . L BYTE  DB01  CODE
```

```
I_NO_COM_A . . . . . L NEAR  E311  CODE
I_NO_COM_B . . . . . L NEAR  E320  CODE
I_NO_GAME_CARD . . . . . L NEAR  E348  CODE
I_NO_SCCS . . . . . L NEAR  E33D  CODE
I_NPU_M . . . . . L BYTE  DACE  CODE
I_OPTROM_M . . . . . L BYTE  DB29  CODE
I_OUT_MASK . . . . . N PROC  E15C  CODE  Length = 0009
I_O_PARITY_DISABLE . . . . . Number  0020
I_PIC . . . . . L NEAR  DCB0  CODE
I_PIC_0_OK . . . . . L NEAR  DCEE  CODE
I_PIC_1_OK . . . . . L NEAR  DCF2  CODE
I_PIC_2_OK . . . . . L NEAR  DCF6  CODE
I_PIC_3_OK . . . . . L NEAR  DCFA  CODE
I_PIC_4_OK . . . . . L NEAR  DCFE  CODE
I_PIC_END . . . . . L NEAR  DD5B  CODE
I_PIC_ERR . . . . . L NEAR  DD21  CODE
I_PIC_HARD . . . . . L NEAR  DCE3  CODE
I_PIC_HOT . . . . . L NEAR  DD1C  CODE
I_PIC_INIT . . . . . N PROC  E14B  CODE  Length = 0011
I_PIC_M . . . . . L BYTE  DAB4  CODE
I_PIC_NO_HOT . . . . . L NEAR  DD4A  CODE
I_PIC_OK . . . . . L NEAR  DD55  CODE
I_PIC_SOFT . . . . . L NEAR  DCD3  CODE
I_PIC_TEST . . . . . L NEAR  DD00  CODE
I_PRT_EXIT . . . . . L NEAR  E2FF  CODE
I_PRT_LOOP . . . . . L NEAR  E2E6  CODE
I_PRT_M . . . . . L BYTE  DB05  CODE
I_RAM_M . . . . . L BYTE  DB1F  CODE
I_ROM . . . . . L NEAR  DBE9  CODE
I_ROM_CHECK . . . . . N PROC  E4CC  CODE  Length = 0036
I_ROM_ERR . . . . . L NEAR  DBF1  CODE
I_ROM_M . . . . . L BYTE  DA8D  CODE
I_ROM_OK . . . . . L NEAR  DBF9  CODE
I_ROM_RET . . . . . L NEAR  DBEF  CODE
I_RTC . . . . . L NEAR  DEDB  CODE
I_RTC_END . . . . . L NEAR  DF46  CODE
I_RTC_ERR . . . . . L NEAR  DF19  CODE
I_RTC_HI_M . . . . . L BYTE  DAEC  CODE
I_RTC_LO_M . . . . . L BYTE  DAE8  CODE
I_RTC_M . . . . . L BYTE  DADB  CODE
I_RTC_NR_M . . . . . L BYTE  DAF0  CODE
I_RTC_OK . . . . . L NEAR  DF3A  CODE
I_VEC0 . . . . . L NEAR  E11F  CODE
I_VEC8 . . . . . L NEAR  E13B  CODE
I_VECTOR . . . . . N PROC  E10F  CODE  Length = 003C
I_VEC_TBL . . . . . N PROC  FEF3  CODE  Length = 0030

KBALT . . . . . Number  00C4
KBBRK . . . . . Number  00C9
KBCAP . . . . . Number  00C1
KBCTL . . . . . Number  00C5
KBINS . . . . . Number  00C0
KBLSH . . . . . Number  00C6
KBNUL . . . . . Number  00CC
KBNUM . . . . . Number  00C2
```

```
KBPRT . . . . . Number  00CB
KBRES . . . . . Number  00C8
KBRSH . . . . . Number  00C7
KBSCR . . . . . Number  00C3
KB_BUFFER . . . . . L WORD  001E  DATA  Length = 0010
KB_CAP_FLAGS . . . . . L BYTE  CC60  CODE
KB_CMD_SEND . . . . . N PROC  E502  CODE  Length = 000B
KB_CMD_WLUP . . . . . L NEAR  E502  CODE
KB_DATA1 . . . . . N PROC  CC60  CODE  Length = 033F
KB_DATA_TABLE . . . . . L BYTE  CC67  CODE
KB_FLAG . . . . . L BYTE  0017  DATA
KB_FLAG_1 . . . . . L BYTE  0018  DATA
KB_FLUSH . . . . . L NEAR  E272  CODE
KB_FLUSH_BACK . . . . . L NEAR  E281  CODE
KB_HERE . . . . . L BYTE  008B  DATA
KB_NOT_DLX . . . . . L NEAR  E2C5  CODE
KB_STATUS . . . . . Number  0064
KB_TYPE_READ . . . . . L NEAR  E2B5  CODE
KB_TYPE_WAIT . . . . . L NEAR  E2AA  CODE
KDBL0 . . . . . Number  00D8
KDEC0 . . . . . Number  00D7
KDEC1 . . . . . Number  00D6
KDEC2 . . . . . Number  00D5
KDEC3 . . . . . Number  00D4
KDEC4 . . . . . Number  00D3
KDEC5 . . . . . Number  00D2
KDEC6 . . . . . Number  00D1
KDEC7 . . . . . Number  00D0
KDEC8 . . . . . Number  00CF
KDEC9 . . . . . Number  00CE
KNONE . . . . . Number  00CD
K_00 . . . . . L NEAR  EB39  CODE
K_1 . . . . . L NEAR  E84D  CODE
K_2RES . . . . . L NEAR  EB1C  CODE
K_2RET . . . . . L NEAR  EB0E  CODE
K_2TOG . . . . . L NEAR  EB13  CODE
K_4RES . . . . . L NEAR  EAEB  CODE
K_4RET . . . . . L NEAR  EAEA  CODE
K_4TOG . . . . . L NEAR  EAD8  CODE
K_ADV_END . . . . . L NEAR  E87C  CODE
K_ADV_PTR . . . . . N PROC  E870  CODE  Length = 000D
K_ALT . . . . . L NEAR  EAF2  CODE
K_ALT0 . . . . . L NEAR  EB2C  CODE
K_ALT1 . . . . . L NEAR  EB2B  CODE
K_ALT2 . . . . . L NEAR  EB2A  CODE
K_ALT3 . . . . . L NEAR  EB29  CODE
K_ALT4 . . . . . L NEAR  EB28  CODE
K_ALT5 . . . . . L NEAR  EB27  CODE
K_ALT6 . . . . . L NEAR  EB26  CODE
K_ALT7 . . . . . L NEAR  EB25  CODE
K_ALT8 . . . . . L NEAR  EB24  CODE
K_ALT9 . . . . . L NEAR  EB23  CODE
K_BEEP . . . . . N PROC  EBD7  CODE  Length = 0039
K_BIT . . . . . N PROC  EB98  CODE  Length = 001F
K_BRK . . . . . L NEAR  EB46  CODE
```

```
K_BUF . . . . . L NEAR  EA53  CODE
K_CAP . . . . . L NEAR  EAB4  CODE
K_CASE . . . . . L WORD  EC10  CODE
K_CTL . . . . . L NEAR  EB04  CODE
K_DATA1 . . . . . N PROC  EC10  CODE    Length = 0032
K_DO_BRK . . . . . L NEAR  EB55  CODE
K_EOI . . . . . N PROC  EBB7  CODE    Length = 0005
K_HOLD . . . . . L NEAR  EA91  CODE
K_INS . . . . . L NEAR  EAA6  CODE
K_INT . . . . . N PROC  E987  CODE    Length = 01E5
K_IO . . . . . N PROC  E82E  CODE    Length = 0017
K_IX . . . . . L NEAR  E9FF  CODE
K_JMP . . . . . L NEAR  EA2A  CODE
K_LED_CAP . . . . . L NEAR  EB76  CODE
K_LED_CMD . . . . . L NEAR  EB83  CODE
K_LED_DAT . . . . . L NEAR  EB8D  CODE
K_LED_NUM . . . . . N PROC  EB6C  CODE    Length = 002C
K_LED_RET . . . . . L NEAR  EB97  CODE
K_LOCK . . . . . L NEAR  E9F1  CODE
K_LOOK . . . . . F PROC  E862  CODE    Length = 0009
K_LP . . . . . L NEAR  EBF0  CODE
K_LSH . . . . . L NEAR  EB08  CODE
K_NON1 . . . . . L NEAR  EABE  CODE
K_NONE . . . . . L NEAR  EA58  CODE
K_NOP . . . . . L NEAR  EA5C  CODE
K_NOP1 . . . . . L NEAR  EB43  CODE
K_NO_CAP . . . . . L NEAR  E9E4  CODE
K_NO_CASE . . . . . L NEAR  EA33  CODE
K_NO_HOLD . . . . . L NEAR  EA49  CODE
K_NO_LOCK . . . . . L NEAR  E9F9  CODE
K_NO_XCODE . . . . . L NEAR  EA51  CODE
K_NUL . . . . . L NEAR  EAA1  CODE
K_NUM . . . . . L NEAR  EAC0  CODE
K_OK . . . . . L NEAR  E9B2  CODE
K_PAUSE . . . . . L NEAR  EA76  CODE
K_PRT . . . . . L NEAR  EA9A  CODE
K_READ . . . . . N PROC  E845  CODE    Length = 001D
K_RES . . . . . L NEAR  EA65  CODE
K_RET . . . . . L NEAR  E842  CODE
K_RSH . . . . . L NEAR  EB0C  CODE
K_RST . . . . . L NEAR  D0FF  CODE
K_SCR . . . . . L NEAR  EACC  CODE
K_SEE . . . . . L NEAR  E856  CODE
K_STAT . . . . . N PROC  E86B  CODE    Length = 0005
K_TRY . . . . . N PROC  EBBC  CODE    Length = 001B
K_TRY_BEEP . . . . . L NEAR  EBD4  CODE
K_XLAT . . . . . L NEAR  EA13  CODE

LEAVE_NMI_DISABLED . . . . . L NEAR  E687  CODE
LEFT_SHIFT . . . . . Number  0002
LEFT_SHIFT_KEY . . . . . Number  002A
LF . . . . . Number  000A
LOOP_WAIT . . . . . L NEAR  D34A  CODE
LP_TABLE . . . . . N PROC  F62B  CODE    Length = 0008
```

```
MASTAB . . . . . L WORD   E23C   CODE
MASTER_TBL_PTR . . . . . L DWORD  0084   DATA
MEMORY_SIZE . . . . . L WORD   0013   DATA
MEMTST . . . . . N PROC   E1D4   CODE     Length = 0047
MEMTST_ERR . . . . . L NEAR   E219   CODE
MEMTST_ERR_C . . . . . L NEAR   E215   CODE
MEMTST_R1 . . . . . L NEAR   E1E7   CODE
MEMTST_R2 . . . . . L NEAR   E1FE   CODE
MEMTST_W1 . . . . . L NEAR   E1DB   CODE
MEMTST_W2 . . . . . L NEAR   E1F4   CODE
MES_1701 . . . . . L NEAR   D12C   CODE
MES_RDY . . . . . L NEAR   D138   CODE
MFG_ERR_FLAG . . . . . L BYTE   0015   DATA     Length = 0002
MFG_TST . . . . . L BYTE   0012   DATA
MFG_TST_RET . . . . . L NEAR   DFF9   CODE
MOTOR_COUNT . . . . . L BYTE   0040   DATA
MOTOR_STATUS . . . . . L BYTE   003F   DATA
MT_END . . . . . L WORD   E252   CODE
M_CASS . . . . . F PROC   F859   CODE     Length = 0003
M_EQUIP . . . . . N PROC   F84D   CODE     Length = 000C
M_SIZE . . . . . N PROC   F841   CODE     Length = 000C

NEC_STATUS . . . . . L BYTE   0042   DATA     Length = 0007
NIBOK . . . . . L NEAR   E5CF   CODE
NMI_ENABLE . . . . . Number   0080
NMI_ENABLE_PORT . . . . . Number   00A0
NON_OLI_VID . . . . . Number   0008
NO_RESET . . . . . L NEAR   D26A   CODE
NO_TO . . . . . L NEAR   F90A   CODE
NPU_ERR . . . . . L NEAR   DF70   CODE
NUL . . . . . Number   0000
NUM_LOCK_KEY . . . . . Number   0045
NUM_LOCK_MODE . . . . . Number   0020
NUM_LOCK_SHIFT . . . . . Number   0020
NXT_CTLR . . . . . L NEAR   D128   CODE
NXT_DRV . . . . . L NEAR   D153   CODE
N_1 . . . . . L NEAR   F870   CODE
N_INT . . . . . N PROC   F85F   CODE     Length = 001C
N_OUT . . . . . L NEAR   F879   CODE

OPT_ROM_M . . . . . L BYTE   E21B   CODE

P0_DATA1 . . . . . N PROC   E21B   CODE     Length = 0037
P1_DATA1 . . . . . N PROC   D9BB   CODE     Length = 01D4
P4_DATA1 . . . . . N PROC   E573   CODE     Length = 0003
PARA_GRAPH . . . . . Number   B800
PARA_MONO . . . . . Number   B000
PARITY . . . . . Number   0000
PARITY1_M . . . . . L BYTE   E625   CODE
PARITY2_M . . . . . L BYTE   E644   CODE
PARITY_DISABLE . . . . . Number   0030
PAR_WR . . . . . L NEAR   D37C   CODE
PAR_WR_ERX . . . . . L NEAR   D3BB   CODE
PASS_M . . . . . L BYTE   DA72   CODE
PAUSE . . . . . Number   00CA
```

```
PAUSE_MODE . . . . . .  Number  0008
PCINIT . . . . . .  N PROC  E252  CODE  Length = 027A
PIC_0 . . . . . .  Number  0020
PIC_1 . . . . . .  Number  0021
PIC_ICW1 . . . . . .  Number  0013
PIC_ICW2 . . . . . .  Number  0008
PIC_ICW3 . . . . . .  Number  0008
PIC_ICW4 . . . . . .  Number  000D
PIC_NEOI . . . . . .  Number  0020
PIC_OFF_MSK . . . . . .  Number  00FF
PIC_SEOI_0 . . . . . .  Number  0060
PIC_SEOI_1 . . . . . .  Number  0061
PIC_SEOI_6 . . . . . .  Number  0066
PORT_OFF . . . . . .  L BYTE  0077  DATA
PRINTER_ADDR . . . . . .  L WORD  0008  DATA  Length = 0004
PRINTER_T_OUT . . . . . .  L BYTE  0078  DATA  Length = 0004
PRT_DATA_A . . . . . .  Number  03BC
PRT_DATA_B . . . . . .  Number  0378
PRT_DATA_C . . . . . .  Number  0278
P_8253_0 . . . . . .  Number  0040
P_8253_1 . . . . . .  Number  0041
P_8253_2 . . . . . .  Number  0042
P_8253_CTRL . . . . . .  Number  0043
P_INIT . . . . . .  L NEAR  F027  CODE
P_IO . . . . . .  N PROC  EFD2  CODE  Length = 0069
P_KCTRL . . . . . .  Number  0061
P_KSCAN . . . . . .  Number  0060
P_LP . . . . . .  L NEAR  F00F  CODE
P_NOP . . . . . .  L NEAR  F00A  CODE
P_OK . . . . . .  L NEAR  F01D  CODE
P_OUT . . . . . .  L NEAR  F00B  CODE
P_RET . . . . . .  L NEAR  F005  CODE
P_STAT . . . . . .  L NEAR  F034  CODE
P_TBL . . . . . .  L WORD  E22C  CODE
P_WX2 . . . . . .  Number  0320

RAM_ERROR . . . . . .  L NEAR  DE24  CODE
RAM_PARITY_DISABLE . . . . . .  Number  0010
RAM_SIZE_END . . . . . .  L NEAR  DE00  CODE
RAM_SIZE_END_1 . . . . . .  L NEAR  DE0A  CODE
RAM_SIZE_LP . . . . . .  L NEAR  DDC1  CODE
RAM_SIZE_NXT . . . . . .  L NEAR  DDF8  CODE
RAM_SIZE_TST . . . . . .  L NEAR  DDAE  CODE
REQ_L . . . . . .  L NEAR  D5A4  CODE
REQ_SUCC . . . . . .  L NEAR  D5B0  CODE
RESET_FLAG . . . . . .  L WORD  0072  DATA
RET_NEAR . . . . . .  L NEAR  D364  CODE
RET_NEAR_1 . . . . . .  L NEAR  D3BD  CODE
RET_NEAR_2 . . . . . .  L NEAR  D56C  CODE
RET_NEAR_3 . . . . . .  L NEAR  D468  CODE
RET_NEAR_4 . . . . . .  L NEAR  D59F  CODE
RET_NEAR_5 . . . . . .  L NEAR  D546  CODE
RET_NO_ERR . . . . . .  L NEAR  D3EF  CODE
RET_STC . . . . . .  L NEAR  D59E  CODE
RET_TIME . . . . . .  L NEAR  D569  CODE
```

```
RET_TIME_J . . . . . L NEAR  D4BD  CODE
RE_INIT_VEC . . . . . L NEAR  E04E  CODE
RE_I_VECTOR . . . . . N PROC  E03C  CODE    Length = 001F
RE_W . . . . . L NEAR  D34E  CODE
RE_W_1 . . . . . L NEAR  D351  CODE
RIGHT_SHIFT . . . . . Number  0001
RIGHT_SHIFT_KEY . . . . . Number  0036
ROM_CHECKSUM . . . . . N PROC  E566  CODE    Length = 000D
ROM_CHECKSUM_CNT . . . . . L NEAR  E569  CODE
ROM_CHECKSUM_LOOP . . . . . L NEAR  E56B  CODE
ROM_CHKSUM_OK . . . . . L NEAR  E4EC  CODE
ROM_ERR . . . . . N PROC  E556  CODE    Length = 0010
ROM_ID . . . . . L BYTE  FFFE  CODE
ROM_MT . . . . . L WORD  C002  CODE
ROM_SCAN_EXIT . . . . . L NEAR  E3BD  CODE
ROM_SCAN_LOOP . . . . . L NEAR  E395  CODE
ROM_SCAN_NEXT . . . . . L NEAR  E3B8  CODE
RS232_ADDR . . . . . L WORD  0000  DATA    Length = 0004
RS_DLY . . . . . N PROC  E8E6  CODE    Length = 000A
RS_GBE . . . . . L NEAR  E949  CODE
RS_INIT . . . . . L NEAR  E363  CODE
RS_LP . . . . . L NEAR  E8EB  CODE
RS_NOP . . . . . L NEAR  E77D  CODE
RS_NORM . . . . . L NEAR  E749  CODE
RS_OK . . . . . L NEAR  E771  CODE
RS_PBE . . . . . L NEAR  E914  CODE
RS_PB_GB . . . . . L NEAR  E913  CODE
RS_RET . . . . . L NEAR  E77A  CODE
RS_STAT . . . . . N PROC  E87D  CODE    Length = 0043
RS_TBL . . . . . L WORD  E77F  CODE
RS_WS . . . . . N PROC  E8C0  CODE    Length = 0026
RS_WS_COM . . . . . L NEAR  E8D1  CODE
RS_WS_EXIT . . . . . L NEAR  E8E2  CODE
RS_WS_LP . . . . . L NEAR  E8C7  CODE
RTC_CHK . . . . . N PROC  E165  CODE    Length = 006F
RTC_CHK_HIGH . . . . . L NEAR  E1D3  CODE
RTC_CHK_LOW . . . . . L NEAR  E1D3  CODE
RTC_CHK_RESET_ERR . . . . . L NEAR  E186  CODE
RTC_CHK_RESET_LP . . . . . L NEAR  E174  CODE
RTC_CHK_RESET_OK . . . . . L NEAR  E189  CODE
RTC_CHK_SET_ERR . . . . . L NEAR  E1A6  CODE
RTC_CHK_SET_LP . . . . . L NEAR  E193  CODE
RTC_CHK_SET_OK . . . . . L NEAR  E1A9  CODE

SCC_CTL_A . . . . . Number  0050
SCC_CTL_B . . . . . Number  0052
SCC_DBIT . . . . . L BYTE  F786  CODE
SCC_FE . . . . . Number  0040
SCC_GB . . . . . N PROC  E94E  CODE    Length = 000D
SCC_INIT . . . . . N PROC  F6CB  CODE    Length = 00BF
SCC_NO_FE . . . . . L NEAR  E8AF  CODE
SCC_NO_OE . . . . . L NEAR  E8BD  CODE
SCC_NO_PE . . . . . L NEAR  E8B6  CODE
SCC_NO_RXD . . . . . L NEAR  E8A1  CODE
SCC_NO_TXD . . . . . L NEAR  E89A  CODE
```

```
SCC_OE . . . . . Number  0020
SCC_PB . . . . . N PROC  E91C  CODE    Length = 000F
SCC_PE . . . . . Number  0010
SCC_PWRUP . . . . . L NEAR  F77B  CODE
SCC_RXD . . . . . Number  0001
SCC_STAT . . . . . L NEAR  E88C  CODE
SCC_TBL . . . . . L WORD  E234  CODE
SCC_TXD . . . . . Number  0004
SCRL_LOCK_KEY . . . . . Number  0046
SCRL_LOCK_MODE . . . . . Number  0010
SCRL_LOCK_SHIFT . . . . . Number  0010
SECT_NOT_FOUND . . . . . Number  0004
SEC_SIZE . . . . . Number  0200
SEC_SIZE_NORM . . . . . L NEAR  D418  CODE
SEEK_ERROR . . . . . Number  0040
SEEK_STATUS . . . . . L BYTE  003E  DATA
SEND_BYTE . . . . . L NEAR  D3BE  CODE
SEND_ERR . . . . . L NEAR  D3C8  CODE
SERIAL_IO . . . . . N PROC  E739  CODE    Length = 0056
SERIAL_T_OUT . . . . . L BYTE  007C  DATA    Length = 0004
SET_DS . . . . . N PROC  E576  CODE    Length = 0006
SET_DS_WORD . . . . . L WORD  E574  CODE
SET_MFG_TST . . . . . N PROC  DFEE  CODE    Length = 000C
SKIP_BEEP . . . . . L NEAR  EC0D  CODE
SKIP_PARITY . . . . . L NEAR  DDB9  CODE
STACK_ROM . . . . . L WORD  D9BC  CODE
STACK_SEG . . . . . Number  0030
START . . . . . L NEAR  0000  CODE
STAT_ERR . . . . . L NEAR  D547  CODE
STAT_LOOP . . . . . L NEAR  D4E8  CODE
STOP_DISK . . . . . N PROC  ED1B  CODE    Length = 0007
SUBTABLE . . . . . L NEAR  D5B2  CODE
SWITCH_8087 . . . . . Number  0010
SWITCH_BITS . . . . . L WORD  0010  DATA
SYS_CONF_A . . . . . Number  0066
SYS_CONF_B . . . . . Number  0067
S_EOL . . . . . L NEAR  FFB5  CODE
S_ERR . . . . . L NEAR  FFA5  CODE
S_GET . . . . . L NEAR  FFC9  CODE
S_INT . . . . . N PROC  FF54  CODE    Length = 007E
S_LP . . . . . L NEAR  FF81  CODE
S_NOP . . . . . L NEAR  FFB2  CODE
S_OK . . . . . L NEAR  FF8A  CODE
S_OUT . . . . . L NEAR  FFBC  CODE

TOCMD . . . . . Number  0036
TOCOUNT . . . . . Number  0000
TOL . . . . . Number  0009
TO_TBL . . . . . L NEAR  D5E7  CODE
T1CMD . . . . . Number  0074
T1COUNT . . . . . Number  0013
T1L . . . . . Number  000A
T1_TBL . . . . . L NEAR  D5F0  CODE
T2CMD . . . . . Number  00B6
T2COUNT . . . . . Number  0266
```

```
T2L . . . . . Number  0002
T2_TBL . . . . . L NEAR  D5FA  CODE
T3CMD . . . . . Number  0054
T3L . . . . . Number  0003
T3_TBL . . . . . L NEAR  D5FC  CODE
TIME_OUT . . . . . Number  0080
TI_FIN . . . . . Number  01BE
TI_K_RST . . . . . Number  0196
TRAP_MESS . . . . . L BYTE  E6F5  CODE
TSTVID . . . . . N PROC  E020  CODE  Length = 001C
TSTVIDEXIT . . . . . L NEAR  E039  CODE
TSTVIDOK . . . . . L NEAR  E038  CODE
TST_DRV_RDY . . . . . L NEAR  D118  CODE
TST_NXT_CNTRLR . . . . . L NEAR  D182  CODE
T_DAY . . . . . N PROC  FE6E  CODE  Length = 0037
T_END . . . . . L NEAR  FEA3  CODE
T_HI . . . . . L NEAR  FEC5  CODE
T_HI_ORDER . . . . . L WORD  006E  DATA
T_INC . . . . . L NEAR  FEBB  CODE
T_INT . . . . . N PROC  FEA5  CODE  Length = 0047
T_LOW_ORDER . . . . . L WORD  006C  DATA
T_NFE . . . . . L NEAR  FE78  CODE
T_NFF . . . . . L NEAR  FE7E  CODE
T_OFL . . . . . L NEAR  FEE1  CODE
T_OVERFLOW . . . . . L BYTE  0070  DATA
T_SET . . . . . L NEAR  FE95  CODE

UNDEF . . . . . L NEAR  D544  CODE

VECTOR . . . . . N PROC  FFF0  CODE  Length = 000F
V_0 . . . . . L NEAR  F381  CODE
V_01 . . . . . L NEAR  F38E  CODE
V_02 . . . . . L NEAR  F38F  CODE
V_1 . . . . . L NEAR  F396  CODE
V_2 . . . . . L NEAR  F3A0  CODE
V_21 . . . . . L NEAR  F3A5  CODE
V_22 . . . . . L NEAR  F3DA  CODE
V_3 . . . . . L NEAR  F3DC  CODE
V_31 . . . . . L NEAR  F3DE  CODE
V_3X8 . . . . . L BYTE  0065  DATA
V_4 . . . . . L NEAR  F3EE  CODE
V_6845 . . . . . L NEAR  F2B6  CODE
V_8253 . . . . . L NEAR  F401  CODE
V_APAGE . . . . . L BYTE  0062  DATA
V_BASE6845 . . . . . L WORD  0063  DATA
V_BELL . . . . . N PROC  F606  CODE  Length = 0025
V_BS . . . . . L NEAR  F57B  CODE
V_C2 . . . . . L NEAR  F37C  CODE
V_CLR . . . . . L NEAR  F32B  CODE
V_CLR_FAST . . . . . L NEAR  F342  CODE
V_CLR_FIN . . . . . L NEAR  F34C  CODE
V_CLR_TOP . . . . . L NEAR  F433  CODE
V_COL . . . . . N PROC  F4D2  CODE  Length = 002C
V_COLORPAL . . . . . L BYTE  0066  DATA
V_COLOUR . . . . . L NEAR  F13A  CODE
```

```
V_COLS . . . . . L NEAR  F379  CODE
V_COL_0 . . . . . L NEAR  F4E7  CODE
V_COL_1 . . . . . L NEAR  F4EE  CODE
V_CR . . . . . L NEAR  F583  CODE
V_CURPOS . . . . . L WORD  0050  DATA  Length = 0008
V_CURSIZE . . . . . L WORD  0060  DATA
V_CURS_POS . . . . . N PROC  F24B  CODE  Length = 001E
V_CURS_TYPE . . . . . N PROC  F23D  CODE  Length = 000E
V_DATA1 . . . . . N PROC  F045  CODE  Length = 0020
V_DATA2 . . . . . N PROC  F0A4  CODE  Length = 0058
V_FPOS . . . . . N PROC  F5C5  CODE  Length = 001E
V_FPOS_0 . . . . . L NEAR  F5D7  CODE
V_FPOS_LP . . . . . L NEAR  F5D0  CODE
V_HEIGHT . . . . . L WORD  004C  DATA
V_IO . . . . . N PROC  F065  CODE  Length = 000A
V_IO_2 . . . . . N PROC  F0FC  CODE  Length = 0054
V_IO_OK . . . . . L NEAR  F120  CODE
V_IO_RET . . . . . L NEAR  F14A  CODE
V_KSCROLL1 . . . . . Number  00E0
V_KSCROLL2 . . . . . Number  0162
V_LF . . . . . L NEAR  F533  CODE
V_LP_CALC . . . . . L NEAR  F687  CODE
V_LP_CALC_GRAF . . . . . L NEAR  F6AA  CODE
V_LP_GRAF_MODES . . . . . L NEAR  F6A0  CODE
V_LP_READ . . . . . L NEAR  F65B  CODE
V_LP_READ_END . . . . . L NEAR  F6BC  CODE
V_LP_READ_LP . . . . . L NEAR  F661  CODE
V_LP_RET . . . . . L NEAR  F6CA  CODE
V_LP_TABLE . . . . . L BYTE  F62B  CODE
V_LP_VALID_MODE . . . . . L NEAR  F63D  CODE
V_LROW . . . . . L NEAR  F53E  CODE
V_MD_40 . . . . . L BYTE  F0A4  CODE
V_MD_80 . . . . . L BYTE  F0B4  CODE
V_MD_CLR . . . . . L NEAR  F1C5  CODE
V_MD_CLR_2K . . . . . L NEAR  F1C0  CODE
V_MD_CLR_8K . . . . . L NEAR  F1C2  CODE
V_MD_CLR_GRAPHICS . . . . . L NEAR  F1BC  CODE
V_MD_DBL . . . . . L NEAR  F1DC  CODE
V_MD_ENABLE . . . . . L BYTE  F0F4  CODE
V_MD_GRAPH . . . . . L BYTE  F0C4  CODE
V_MD_LEN . . . . . L WORD  F0E4  CODE
V_MD_MONO . . . . . L BYTE  F0D4  CODE
V_MD_WID . . . . . L BYTE  F0EC  CODE
V_MODE . . . . . L BYTE  0049  DATA
V_MV . . . . . L NEAR  F2E9  CODE
V_MV1 . . . . . L NEAR  F303  CODE
V_MV2 . . . . . L NEAR  F315  CODE
V_MV_DN . . . . . L NEAR  F42C  CODE
V_MV_FAST . . . . . L NEAR  F31B  CODE
V_MV_FLP . . . . . L NEAR  F31E  CODE
V_NOP . . . . . L NEAR  F06E  CODE
V_OUT_BYTE . . . . . N PROC  F2C7  CODE  Length = 000C
V_OVR_NOT_OK . . . . . L NEAR  F218  CODE
V_OVR_OK . . . . . L NEAR  F21A  CODE
V_PAGE . . . . . N PROC  F280  CODE  Length = 0047
```

```
V_PAGE_O . . . . . L NEAR  F292  CODE
V_PARMS . . . . . L BYTE  F0A4  CODE
V_POINTER . . . . . Number  03B4
V_POSN . . . . . N PROC  F5E3  CODE  Length = 0011
V_RAC . . . . . N PROC  F438  CODE  Length = 002A
V_RAC_INBLANK . . . . . L NEAR  F454  CODE
V_RAC_INLINE . . . . . L NEAR  F44E  CODE
V_ROWS . . . . . L NEAR  F378  CODE
V_R_CURS_POS . . . . . N PROC  F269  CODE  Length = 0017
V_SCRL_DN . . . . . N PROC  F417  CODE  Length = 0021
V_SCRL_MODE_7 . . . . . L NEAR  F35A  CODE
V_SCRL_MV_AND_CLR . . . . . L NEAR  F5C4  CODE
V_SCRL_POS . . . . . N PROC  F596  CODE  Length = 002F
V_SCRL_TTY . . . . . L NEAR  F549  CODE
V_SCRL_TTY_GRAPHICS . . . . . L NEAR  F558  CODE
V_SCRL_UP . . . . . N PROC  F2D3  CODE  Length = 0144
V_SCROLL_OR_CLEAR . . . . . L NEAR  F35F  CODE
V_SET_CURS . . . . . L NEAR  F264  CODE
V_SET_CUR_POS . . . . . L NEAR  F2AA  CODE
V_SET_MODE . . . . . N PROC  F150  CODE  Length = 00ED
V_SET_MODE_COLOR . . . . . L NEAR  F16B  CODE
V_SET_MODE_LP . . . . . L NEAR  F19B  CODE
V_SET_NEW_CUR . . . . . L NEAR  F544  CODE
V_STAT . . . . . N PROC  F587  CODE  Length = 000F
V_SYNC . . . . . L NEAR  F40C  CODE
V_SYNC2 . . . . . L NEAR  F411  CODE
V_TBL . . . . . L WORD  F045  CODE
V_TERMINAL . . . . . N PROC  F4FE  CODE  Length = 0089
V_TERM_NOBELL . . . . . L NEAR  F505  CODE
V_TERM_NOP . . . . . L NEAR  F573  CODE
V_TERM_RET . . . . . L NEAR  F571  CODE
V_TOP . . . . . L WORD  004E  DATA
V_TXT_DN . . . . . L NEAR  F420  CODE
V_TXT_MD . . . . . N PROC  F5F4  CODE  Length = 0012
V_TXT_OK . . . . . L NEAR  F604  CODE
V_TXT_RAC . . . . . L NEAR  F440  CODE
V_TXT_UP . . . . . L NEAR  F2DB  CODE
V_TXT_WAC . . . . . L NEAR  F46A  CODE
V_TXT_WC . . . . . L NEAR  F49F  CODE
V_V . . . . . L NEAR  F3AC  CODE
V_V2 . . . . . L NEAR  F3E7  CODE
V_WAC . . . . . N PROC  F462  CODE  Length = 0035
V_WAC_END . . . . . L NEAR  F492  CODE
V_WAC_HI . . . . . L NEAR  F481  CODE
V_WAC_LO . . . . . L NEAR  F487  CODE
V_WC . . . . . N PROC  F497  CODE  Length = 003B
V_WC_END . . . . . L NEAR  F4CD  CODE
V_WC_HI . . . . . L NEAR  F4B6  CODE
V_WC_LO . . . . . L NEAR  F4BC  CODE
V_WC_NEXT . . . . . L NEAR  F4B6  CODE
V_WIDTH . . . . . L WORD  004A  DATA

W1 . . . . . L NEAR  D497  CODE
W2 . . . . . L NEAR  D4A2  CODE
WAIT_FOR_IRQ5 . . . . . L NEAR  D4AD  CODE
```

```
WINS_USABLE . . . . . .  L NEAR   D19C     CODE
WIN_BC . . . . . .  L NEAR   D257     CODE
WIN_BC_1 . . . . . .  L NEAR   D268     CODE
WIN_BOOT . . . . . .  L NEAR   D20E     CODE
WIN_BOOT_NXT . . . . . .  L NEAR   D22C     CODE
WRITE_PROTECT . . . . . .  Number  0003
WST_CYL . . . . . .  Number  0000
WST_ER_BUR . . . . . .  Number  0007
WST_HEADS . . . . . .  Number  0002
WST_OPT . . . . . .  Number  0008
WST_RE_WR . . . . . .  Number  0003
WST_WR_PRE . . . . . .  Number  0005
WX2_CC . . . . . .  L NEAR   D583     CODE
WX2_INT . . . . . .  L NEAR   D4C0     CODE
WX2_INT_1 . . . . . .  L NEAR   D4C3     CODE
WX2_INT_2 . . . . . .  L NEAR   D4C3     CODE
WX2_REQ . . . . . .  L NEAR   D5A0     CODE
WX2_WAIT . . . . . .  L NEAR   D470     CODE
W_NEC . . . . . .  L NEAR   EF92     CODE
W_NEC_RET . . . . . .  L NEAR   EFA6     CODE
W_ONE . . . . . .  L NEAR   EF3B     CODE

$$$X . . . . . .  Number  FFED


 13144 Source  Lines
 13400 Total   Lines
  1153 Symbols

  9120 Bytes symbol space free

     2 Warning Errors
     0 Severe  Errors
```