README

```
;This collection of source code was typed (for typing practice
;and as an educational exercise) from Volume 1 of THE SOURCE.

;It contains all files required to construct a boot disk. Boot
;sectors can be written with DEBUG (see version 6 documentation)
;I have successfully assembled and booted "LS-DOS Level-xx" from
;these source files.

;The annotated source assembles without error using
;Disk-Editor-Assembler (D-E-A) by D. Goben. Every Hex byte was
;carefully compared to the original listing for correct
;addresses. Slight modifications to arithmetic syntax and some
;additional annotation were also made.

;Also included is a simple filter/utility program, CRLF.EXE,
;which adds linefeeds to carriage return (or vice-versa).

;06 JAN 1998
;Douglas Beattie Jr.             <beattidp@whidbey.net>
;
```

INTRO

LOWCORE/ASM - Low memory assignments and BIOS

This is the source code that assembles the file BOOT/SYS as a core image (no load information) file. The Lowcore file contains the bootstrap loader, certain low memory storage locations, the I/O drivers, the DCBs and DCTs, memory bank handling, byte I/O routines, and other miscellaneous code. The file normally occupies track 0 on a system disk, and is read into memory by the computer's boot ROM. At that point, execution is transferred to the Lowcore boot loader and Sysres is loaded.

The Lowcore source code is divided into several subsections. In order of appearance, they are:

DCB area - Default DCBs for the standard I/O devices.
BOOT - Boot loader for Sysres and CRTC initialization.
SYSINFO & DCT - The stack area, input buffer, and default DCTs.
IODVR - Byte I/O handling for standard and non-standard devices.
MULDIV - Math routines used by the system.
CLOCKS - Heartbeat processing vidram and bank switching.
KIDVR - The keyboard driver and type-ahead processing.
DODVR - The video driver and VDCTL SVC handler.
PRDVR - The printer driver.
FDCDVR - The floppy disk driver

SYSRES - Resident portion of the DOS

This source code assembles the load module file SYS0/SYS, commonly referred to as Sysres. It contains the BDOS and other routines. It is origined at 1300H, which puts it above Lowcore and the low memory driver zone. It is loaded into position by the boot loader in Lowcore. Once loaded, execution is begun at the SYSINIT entry point. This initilizes the remaining machine hardware, loads the configuration file if present, executes any Auto command, and brings in the command interpreter. The main subsections of Sysres are:

Certain low memory values and the system flag table.
FILPOSN - File read/write, positioning, and allocation routines.
LOADER - SVC handling, overlay loading, command file loading.
TASKER - Interrupt processing.
SYSINIT - System initialization after cold or warm boot.
SOUND - Sound, Pause, and some other miscellaneous low memory routines.
LOGO -- The signon graphics display in direct load-to-screen format.

SYS1/SYS

SYS1 is, among other things, the primary command interpreter. As such, in handles all requests for commands in the three system Libraries. It also contains the code for the SVCs @CMNDI, @CMNDR, @FSPEC, @FEXT, @PARAM, @EXIT and @ABORT. SYS1 is normally executed by doing an @EXIT or @ABORT SVC from within a program, or by executin a RET instruction as long as the stack pointer is at the same position it was in when the program was executed by DOS.

SYS2/SYS

SYS2 is a multifunction overlay. It handles creating, opening, and renaming files, hashing filenames and password, checkin a drive for mounted diskette, and locating a specified of free DCB. It contains the code for the SVCs @INIT, @OPEN, @RENAME, @GTDCB, and @CKDRV.

SYS3/SYS

SYS3 handles closing a file or device, and restoring original file spec or device spec to the Control Block. During a file close, it also de-allocates space if the ending record number is smaller than it was upon open. It contains the code for SVCs @CLOSE and @FNAME.

SYS4/SYS

SYS4 handles the system error routines, either displaying an error message or placing the message in a user-specified buffer. Beside the standard error codes, an "extended error" (error number 63) will display an error value placed in the HL register pair. The only SVC handled by SYS4 is @ERROR.

SYS5/SYS

SYS5 is the primary system debugger. It is activated by the system, by the break key, or by the SVC @DEBUG. During assembly, SYS5 is cross-referenced to produce an EQUate file used in assembling SYS9, the extended debugger.

Note: SYS6, SYS7 and SYS8 are library overlays, and contain all the executable DOS commands.

SYS9/SYS

SYS9 is the extended system debugger. When loaded, it resides as a module in high memory, and handles additional functions that require more memory than was available for SYS5 which resides in the overlay region. There are no SVCs directly handled by SYS9.

SYS10/SYS

SYS10 will kill a file or device. It contains code for the SVC @REMOV.

SYS11/SYS

SYS11 handles the execution of a JCL file. It automatically links and unlinks itdelf into the system. It is used to supply input from the JCL file in response to keyboard line requests.

SYS12/SYS

SYS12 handles the two mini directory and free space SVCs, as well as locates or checks for a memory module header. It contains the code for the SVCs @DODIR, @RAMDIR and @GTMOD.

SYS13/SYS

SYS13 holds the place reserved for an extended command interpreter. It performs no function, but is on the disk to set the proper attributes of any file copied over it to act as an extended interpreter. It also prevents a system hangup if the extended interpreter flag (EFLAG$) is set without the presence of a user file in the SYS13 directory slot.

LOWCORE/ASM

```
;LOWCORE/ASM - Low Memory Assignments
      ADISP '<LOWCORE - LS-DOS 6.2>'
@MOD2 EQU   00          ;Set MOD2 false
@MOD4 EQU   -1          ;Set MOD4 true
;
;     LDOS 6.x Low Core RAM storage assignments
;     Copyright (C) 1982 by Logical Systems, Inc.
;
;     Define switches for international or domestic
;
@GERMAN     EQU   0
@FRENCH     EQU   0
      IF    @GERMAN.AND.@FRENCH
      ADISP 'Can't do both French and German'
      ENDIF
      IF    @GERMAN.OR.@FRENCH
@INTL EQU   -1
@USA  EQU   00
@HZ50 EQU   -1
      ELSE
@INTL EQU   00
@USA  EQU   -1
@HZ50 EQU   00
      ENDIF
;
START$      EQU   0
;
;     These EQUs are detailed in SYSRES
;
FDDINT$     EQU   0EH
PDRV$ EQU   1BH
TIMSL$      EQU   2BH
TIMER$      EQU   2CH
TIME$ EQU   TIMER$+1
DATE$ EQU   33H
INTVC$      EQU   3EH
FLGTAB$     EQU   6AH
CFLAG$      EQU   FLGTAB$+'C'-'A'
DFLAG$      EQU   FLGTAB$+'D'-'A'
IFLAG$      EQU   FLGTAB$+'I'-'A'
KFLAG$      EQU   FLGTAB$+'K'-'A'
MODOUT$     EQU   FLGTAB$+'M'-'A'
NFLAG$      EQU   FLGTAB$+'N'-'A'
OPREG$      EQU   FLGTAB$+'O'-'A'
RFLAG$      EQU   FLGTAB$+'R'-'A'
SFLAG$      EQU   FLGTAB$+'S'-'A'
VFLAG$      EQU   FLGTAB$+'V'-'A'
@KITSK      EQU   FLGTAB$+31
;
      ORG   200H+START$
;
;     Page 2 - Device Control Blocks
;
```

```
BUR$  DB    00H         ;Bank use RAM
BAR$  DB    0FEH        ;Bank available RAM
LBANK$      DB    20          ;Dir cyl & logical bank
JCLCB$      DB    1,0,0       ;Mini-DCB for JCL gets
DVRHI$      DW    DVREND$           ;Start of low I/O zone
KIDCB$      DB    5           ;Permit CTL, GET
      DW    KIDVR
      DB    0,0,0,'KI'
DODCB$      DB    7           ;Permit CTL, PUT, GET
      DW    DODVR
      DB    0,0,0,'DO'
PRDCB$      DB    6           ;Permit CTL, PUT
      DW    PRDVR
      DB    0,0,0,'PR'
SIDCB$      DB    15H         ;Routed to *KI
      DW    KIDCB$
      DB    0DH,0,0,'SI'
SODCB$      DB    17H         ;Routed to *DO
      DW    DODCB$
      DB    0FH,0,0,'SO'
JLDCB$      DB    0AH,0,0,0AH,0,0,'JL'
S1DCB$      EQU   $           ;1st spare DCB
DCBKL$      EQU   JLDCB$&0FFH+1     ;Non-killable DCB's
;
;     Now load the BOOT loader - part in this page
;
*GET  'BOOT4:1'
;
      ADISP '<SYSinfo Section>'
;     ?
;
;     Page  3 - System stack and Sysinfo section
;
STACK$      EQU   $-128       ;Start stack 128 bytes low
PAUSE@      EQU   STACK$+2    ;Where pause will be
;
;     Page 4 - Miscellaneous stuff
;
      DB    62H         ;Operating system version
ZERO$ DB    0C9H        ;Config on BOOT, yes = 0
MAXDAY$ EQU $-1         ;Max days per month
      DB    31,28,31,30,31,30,31,31,30,31,30,31
HIGH$ DS    2           ;Highest available memory
PAKNAM$ DB  'LS-DOS62Level-xx'
;
;     Command line input buffer & AUTO buffer area
;
INPBUF$     DB    0DH         ;Input buffer - 80 bytes
      DS    79%0
;
;     System drive code tables
;
DCT$  EQU   $           ;System drive code tables
      JP    FDCDVR            ;Floppy drive 0
      DB    44H,0C1H,0,27H,17,3-1<5+6-1,20
      JP    FDCDVR            ;Floppy drive 1
      DB    44H,42H,-1,27H,17,3-1<5+6-1,20
      RET               ;Disable drive #2
      DW    FDCDVR
      DB    44H,44H,-1,27H,17,3-1<5+6-1,20
      RET               ;Disable drive #3
      DW    FDCDVR
      DB    44H,48H,-1,27H,17,3-1<5+6-1,20
      RET               ;Logical drive 4
      DW    FDCRET
      DB    0,0,0,27H,0,0,0
      RET               ;Logical drive 5
      DW    FDCRET
      DB    0,0,0,27H,0,0,0
      RET               ;Logical drive 6
      DW    FDCRET
      DB    0,0,0,27H,0,0,0
      RET               ;Logical drive 7
      DW    FDCRET
      DB    0,0,0,27H,0,0,0
;
;     SYSINFO - miscellaneous information
;
DSKTYP$     DB    -1          ;0 = DATA, <> 0 = SYS
      DB    0           ;Reserved
DTPMT$      DB    0           ;Date prompt at boot
TMPMT$      DB    -1          ;Time prompt at boot
RSTOR$      DB    0           ;Suppress restores on BOOT
      DS    2           ;Reserved
DAYTBL$     DB    'SunMonTueWedThuFriSat'
MONTBL$     DB    'JanFebMarAprMayJunJulAugSepOctNovDec'
;
;     End of low core assignments
;
*GET  'IODVR:1'         ;I/O driver, KEYIN, etc.
*GET  'MULDIV:1'        ;16-bit MULT & DIV
*GET  'CLOCKS:1'        ;Hardware task stuff
@$SYS EQU   $           ;Pointer for @GTMOD
      IF    @USA
*GET  'KIDVR:1'         ;Keyboard driver
      ENDIF
      IF    @GERMAN
FREN  EQU   00
GERM  EQU   -1
;     ?
      ENDIF
      IF    @FRENCH
FREN  EQU   -1
GERM  EQU   00
;     ?
      ENDIF
*GET  'DODVR:1'         ;Video driver
*GET  'PRDVR:1'         ;Printer driver & filter
*GET  'FDCDVR:1'        ;Floppy disk driver
DVREND$     EQU   $           ;Start of low I/O area, to 12FFH
      IF    $.GT.1200H+START$
      ADISP 'Drivers overflow available RAM'
      ENDIF
      ORG   1300H+START$
```

```
@BYTEIO      EQU   $
      END
□
```

BOOT4/ASM

```
;BOOT4/ASM - LS-DOS 6.2
      ADISP '<Bootstrap Loader>'
;     ?
;*MOD
;
KEYIN EQU   40H
NMIVECT     EQU   66H
DSPLY EQU   21BH
BUFFER      EQU   1200H
BOOTBUF     EQU   43FFH-9
;
;     Boot loader routine read in by ROM, along with
;       the lowcore I/O drivers.
;     This section loads in SYSRES
;
LBOOT LD    IY,DCT$           ;Set IY for FDCDVR use
      LD    A,(IY+9)    ;Directory track is
      LD    (IY+5),A    ;  the current track
      LD    A,4
      LD    (FLGTAB$+'R'-'A'),A     ;Set retries
      LD    A,0C9H
      LD    (FDDINT$),A ;Return for disk driver
      LD    A,18        ;5" sectors/track, dden
      BIT   5,(IY+4)    ;Dbl sided?
      JR    Z,NOTDBL
      ADD   A,A         ;Adjust to 36 sect/cyl
NOTDBL      LD    (SECTRK),A
;
;     Set up for a fragmented file
;
      EXX
      LD    C,6         ;Sectors/gran
      CALL  GETEXT            ;Pick up extent 1
      EXX
;
      CALL  LOAD        ;Read in SYSRES
      LD    A,0FBH            ;EI instruction
      LD    (DISKEI),A  ;  stuffed into FDCDVR
      JP    (HL)        ;Continue system init
;
LOAD  CALL  RDBYTE            ;Get type code
      DEC   A
      JR    NZ,LOAD2    ;Bypass if not type 1
      CALL  GETADR            ;Get blk len & load adr
LOAD1 CALL  RDBYTE            ;Start reading the block
      LD    (HL),A            ;Stuff into memory
      INC   HL          ;Bump memory pointer
      DJNZ  LOAD1       ;Loop for entire block
      JR    LOAD        ;Restart the process
;
LOAD2 DEC   A           ;Test if type 2 (traadr)
      JR    Z,GETADR    ;Ah, go if transfer addr
      CALL  RDBYTE            ;Assume comment,
      LD    B,A         ;  get comment length
```

```
LOAD3 CALL  RDBYTE            ;  & ignore it
      DJNZ  LOAD3
      JR    LOAD        ;Continue to read
;
;     got the transfer address type code
;
GETADR      CALL  RDBYTE            ;Get block length
      LD    B,A
      CALL  RDBYTE            ;Get low-order byte
      LD    L,A
      DEC   B           ;Adj length for this byte
      CALL  RDBYTE            ;Get high-order byte
      LD    H,A         ;Load address is formed
      DEC   B           ;Adj length for this byte
      RET
;
;     Routine to read a byte
;
RDBYTE      EXX               ;Switch memory/buf ptrs
      INC   L           ;Bump buf pointer
      JR    NZ,RDB2           ;Bypass disk I/O if more
      PUSH  BC
      LD    B,9         ;Read sector function #
      CALL  DCT$        ;Get another sector
      POP   BC
      INC   E           ;Bump sector counter
      LD    A,E
      SUB   $-$         ;Is this the last sector
SECTRK      EQU   $-1         ;  on the cylinder?
      JR    NZ,RDB1
      LD    E,A         ;Yes, restart at 0
      INC   D           ;  & bump the cylinder up
RDB1  DEC   B           ;Dec sectors this extent
      CALL  Z,GETEXT    ;Get next extent if 0
RDB2  LD    A,(HL)            ;P/u a byte
      EXX               ;Exc mem/buf pointers
      RET
;
;     Load DE track,sector, B sectors this extent
;
GETEXT      EQU   $
      INC   IX          ;Index directory entry
      INC   IX          ;Pt at grans this ext.
      LD    A,(IX+0)
      PUSH  AF          ;Save for later
      RLCA
      RLCA              ;Normalize start gran
      RLCA
      AND   7
      CALL  MULTCA            ;Start gran * grans/sec
      LD    E,A         ;This is start sector
      POP   AF
      AND   00011111B   ;Get total grans
      INC   A           ;  this extent
      CALL  MULTCA            ;  * sect/gran
      LD    B,A         ;Sectors this extent
      LD    D,(IX-1)    ;Cyl this extent
      RET
;
;     Short multiply C * A
;
MULTCA      PUSH  BC          ;Save sect/gran in C
      LD    D,A
      XOR   A
      LD    B,8
MLTCA ADD   A,A
      SLA   C
      JR    NC,MLTCA1
      ADD   A,D
MLTCA1      DJNZ  MLTCA
      POP   BC
      RET
;
;     Initialize the MC6835 CRTC
;
INITCRTC EQU      $
      LD    BC,15<8!88H ;Count, CRTC address reg
      LD    HL,CRTCTAB  ;Point HL to data table
$A1   LD    A,(HL)
      OUT   (C),B       ;Pass reg # to CRTC
      OUT   (89H),A           ;Pass value to CRTC reg
      DEC   HL          ;Back up to next value
      DEC   B           ;To next lower reg
      JP    P,$A1
      RET
      DB    99          ;Horiz total MD
      DB    80          ;Horiz displayed MD
      DB    86          ;Horiz sync position MD
      DB    8           ;Horiz sync width
      DB    24          ;Vertical total
      DB    0           ;Vertical total adjust
      DB    24          ;Vertical displayed
      DB    24          ;Vertical sync position
      DB    0           ;Interlace mode
      DB    9           ;Maximum scan line addr
      DB    65H         ;Cursor start
      DB    9           ;Cursor end
      DB    0           ;Start address (H)
      DB    0           ;Start address (L)
      DB    0           ;Cursor     (H)
CRTCTAB     DB    0           ;Cursor (L)
      DS    -$&0FFH%0
;
;     System BOOT entry point, loaded by ROM
;
CORE$ DEFL  $
      ORG   4300H
BOOT  NOP
      CP    14H         ;Directory track location
DIRTRK      EQU   $-1
      DI
      LD    A,86H       ;Bring up the RAM
      OUT   (84H),A
      LD    (OPREG$),A  ;
      LD    HL,CRTBGN$  ;Clear video RAM
      LD    DE,CRTBGN$+1
      LD    BC,CRTSIZE-1
      LD    (HL),' '
      LDIR
      LD    HL,NMIRET   ;Set NMI vector
      LD    (NMIVECT+1),HL
      LD    A,0C3H
      LD    (NMIVECT),A
      LD    A,0C9H            ;Stuff return for ints
      LD    (38H),A
;
;     Read the first 16 sectors of track 0
;
      LD    HL,START$+200H    ;Pt to page 2
      LD    D,L         ;Init to track 0, sec 0
      LD    E,L
RDBOOT      CALL  RDSEQ       ;Read a sector
      INC   H           ;Bump to next page
      INC   E           ;Bump to next sec
      LD    A,16
      CP    E           ;Loop if more
      JR    NZ,RDBOOT
      CALL  INITCRTC    ;Initialize the CRTC
;
;     Now set up to load SYSRES
;
      LD    A,(DIRTRK)  ;P/u dir cyl
      LD    (DCT$+9),A  ;Update DCT to show DIR
      LD    D,A         ;Set starting track and
      LD    E,0         ;  init to read the GAT
      CALL  RDSECT            ;  into BUFFER
      LD    A,(BUFFER+0CDH)   ;Update DCT$ to show
      AND   20H         ;  The # of sides
      LD    HL,DCT$+4
      OR    (HL)
      LD    (HL),A
      LD    E,4         ;pt to SYS0 dir sector
      CALL  RDSECT            ;Read the SYS0 dir sec
      LD    A,(BUFFER)  ;Test if system disk
      AND   10H
      JR    Z,NOTSYS    ;Go if not
      LD    HL,BUFFER+21+8    ;SYS0 extent info
      LD    DE,BOOTBUF  ;Use 43FF-8
      LD    BC,8
      LDDR              ;Store 1st four extents
      PUSH  DE          ;Pt IX to 1 byte
      POP   IX          ; before extent info
      EXX
      LD    HL,BUFFER+255     ;Init to buffer end
      EXX
      JP    LBOOT       ;Load SYSRES
      DB    0,0         ;Padding for posn
;
;     Routine to read a sector
;
RDSECT      LD    HL,BUFFER   ;Set buffer
```

```
RDSEQ LD    B,5         ;Init retry counter
RDS1  PUSH  BC          ;Save counter
      PUSH  HL          ;Save for retries
      CALL  READ        ;Attempt read
      POP   HL
      POP   BC
      AND   1CH         ;Mask status
      RET   Z           ;Return if no error
      DJNZ  RDS1        ;Loop for retry
GOTERR      LD    HL,DISKERR  ;"Disk error"
      DB    0DDH        ;Hide next instruction
NOTSYS      LD    HL,NOSYS    ;"No system"
      LD    BC,ERRLEN
      LD    DE,80*11+CRTBGN$+35     ;Middle of screen
      LDIR
HALTS JR    HALTS       ;Wait for RESET
;
READ  LD    BC,81F4H    ;Set DDEN, DS1, d.s. port
      OUT   (C),B       ;Select it
      DEC   C           ;Point C to data reg
      LD    A,18H       ;Seek command (6 ms)
BOOTST$ EQU $-1         ;Set for boot step rate
      IF    BOOTST$.NEQ.439DH
      ADISP 'Bootstep out of position'
      ENDIF
      OUT   (C),D       ;Set desired track
      CALL  FDCMD       ;Pass command & Delay
SEEK1 IN    A,(0F0H)    ;Get status
      BIT   0,A         ;Busy?
      JR    NZ,SEEK1
      LD    A,E         ;Set sector register
      OUT   (0F2H),A
      LD    A,81H       ;Set DDEN & DS1
      OUT   (0F4H),A
      PUSH  DE
      LD    DE,2!(81H!40H)<8 ;D=DS1 + DDEN + WSGEN
                        ;  E=Mask to see DRQ
      LD    A,80H       ;FDC READ command
      CALL  FDCMD       ;Pass to ctrlr & set B=0
      LD    A,0C0H            ;Enable INTRQ & timeout
      OUT   (0E4H),A
READLP1     IN    A,(0F0H)    ;Grab status
      AND   E           ;Test bit 1
      JR    Z,READLP1
      INI
      LD    A,D         ;Set DDEN & DS1 & WSGEN
READLP2     OUT   (0F4H),A    ;Continue to select
      INI               ;  While inputting
      JR    NZ,READLP2
      JR    $           ;Wait for NMI
NMIRET      POP   DE          ;Pop interrupt ret
      POP   DE          ;Restore DE
      XOR   A           ;Disable INTRQ & timeout
      OUT   (0E4H),A
      LD    A,81H       ;Reselect drive
      OUT   (0F4H),A
      IN    A,(0F0H)    ;Get status
      RET
FDCMD OUT   (0F0H),A    ;Give cmd to ctrlr
      LD    B,24        ;Time delay
      DJNZ  $
      RET
DISKERR DB  'Disk error'
NOSYS DB    'No system '
ERRLEN      EQU   $-NOSYS           ;Length of error msg
      DS    -$&0FFH%0
      ORG   CORE$+256
      END
```

IODVR/ASM

```
;IODVR/ASM - LS-DOS 6.2
      ADISP '<Device I/O handling>'
;     ?
;
HOME  EQU   1CH
CLRFRM      EQU   1FH
;
;     Log out routine - display & log
;
@LOGOT      CALL  @DSPLY
;
;     Job log logerroutine
;
@LOGER      LD    A,(JLDCB$)  ;If NIL, don't do
      XOR   8           ;  anything
      AND   8
      RET   Z
      PUSH  HL          ;Save pointer to command
      LD    HL,LOGBUF   ;Get time string into buf
      PUSH  HL
      CALL  @TIME
      POP   HL
      LD    DE,JLDCB$   ;Log the time
      CALL  @MSG
      POP   HL          ;Log the command
      JR    @MSG
LOGBUF      DB    'hh:mm:ss  ',3
;
;     Line print routine
;
@PRINT      LD    DE,PRDCB$   ;Printer DCB
      JR    @MSG
;
;     Line display routine
;
@DSPLY      LD    DE,DODCB$   ;Video DCB
;
;     Device message routine
;
;*MOD
@MSG  PUSH  HL          ;Save pointer to message
$B1   LD    A,(HL)            ;P/u a message character
      CP    3           ;Exit on ETX
      JR    Z,$B3
      CP    CR          ;Exit & put on ENTER
      JR    Z,$B2
      CALL  NZ,@PUT           ;Else put the char
      INC   HL          ;  & loop on no error
      JR    Z,$B1       ;  else fall thru and exit
$B2   CALL  Z,@PUT
$B3   POP   HL
      RET
;
;     Clear screen routine
```

```
;
@CLS  LD    A,HOME            ;Cursor home to 0,0
      CALL  DSPBYT
      RET   NZ          ;Return on error
      LD    A,CLRFRM    ;Clear to end of frame
DSPBYT      PUSH  DE
      CALL  @DSP
      POP   DE
      RET
;
;     Check and Clear <BREAK> bit SVC
;
@CKBRKC     EQU   $
      PUSH  HL          ;Save registers
      LD    HL,KFLAG$   ;Point to KFLAG$
      BIT   0,(HL)            ;Check break bit
      JR    Z,NOBRK           ;  and ret if none
      PUSH  AF
      PUSH  BC
      PUSH  DE
BRKTEST     RES   0,(HL)            ;Reset the break bit
      LD    BC,0B00H    ;Wait more than 1/30
      CALL  PAUSE@            ;  of a second
      BIT   0,(HL)            ;Test the bit again
      JR    NZ,BRKTEST  ;Loop until gone
      LD    DE,KIDCB$   ;Point at keyboard &
      LD    A,03        ;  clear buffer
      CALL  @CTL        ;  control 3 call
      POP   DE
      POP   BC          ;Recover registers
      POP   AF          ;Recover flags
NOBRK POP   HL
      RET
;
;     Keyboard line input routine
;
;*MOD
;
;     Backspace to beginning of line
;
$C4   CALL  $C6         ;Backspace
      DEC   HL          ;Get the char prior
      LD    A,(HL)            ;  to the current
      INC   HL
      CP    0AH         ;Return if line feed
      RET   Z
$C5   LD    A,B         ;Check for empty buffer
      CP    C
      JR    NZ,$C4            ;Loop if not
      RET               ;  else return
@KEYIN      PUSH  HL          ;Save buffer pointer
      LD    C,B         ;Set C = buffer size
$C1   LD    DE,@KEY           ;Init for standard input
      LD    A,(SFLAG$)  ;If JCL is active,
      AND   20H         ;  then use the JCL input
      JR    Z,$C0       ;Must loop here in case
      LD    E,@JCL&0FFH ;  JCL exits with //STOP
```

```
$C0   LD    ($C1A+1),DE
$C1A  CALL  $-$         ;Get a key
      JR    NZ,$C3B           ;Back on error
      CP    80H         ;Break?
      JR    Z,$C10
      CP    20H         ;Go if not a control
      JR    NC,$C2
      CP    0DH         ;Carriage return?
      JR    Z,$C11
      CP    1FH         ;Clear?
      JR    Z,$C3
      LD    DE,$C1            ;Set return address
      PUSH  DE
      CP    08H         ;Backspace?
      JR    Z,$C6
      CP    18H         ;Backspace to BOL
      JR    Z,$C5
      CP    09H         ;Tab?
      JR    Z,$C8
      CP    'R'&1FH           ;CTL-R?
      JR    Z,$C7
      CP    0AH         ;Line feed?
      RET   NZ          ;Ret if none above
      POP   DE          ;Pop the return
$C2   LD    (HL),A            ;Stuff the char
      LD    A,B         ;Check on buffer full
      OR    A
      JR    Z,$C1       ;Loop if so
      LD    A,(HL)            ;  else get char
      INC   HL          ;  & bump pointer
      DEC   B           ;Count down
      CALL  @DSP        ;Display entry
      JR    $C3A        ;  then loop
;
;     Clear the screen invoked
;
$C3   CALL  @CLS
      LD    B,C         ;Reset to start of
      POP   HL          ;  line & start of
      PUSH  HL          ;  buffer
$C3A  JR    Z,$C1
$C3B  JR    $C11
;
;     Backspace key entry
;
$C6   LD    A,B         ;If buffer is empty
      CP    C           ;  return
      RET   Z
      DEC   HL          ;  else do the backspace
      LD    A,(HL)
      CP    0AH         ;Last char a line feed?
      INC   HL
      RET   Z           ;Return if so
      DEC   HL
      INC   B           ;Add back one char
      LD    A,8         ;Backspace the cursor
      JR    @DSP
```

```
;
;      Test if repeat last command
;
$C7    LD     A,(CFLAG$)  ;Test if SYS1 KEYIN bit
       AND    4           ;  is set (bit 2)
       RET    Z           ;Ignore CTL if not
       LD     A,B         ;If not at 1st position,
       CP     C           ;  dont permit it
       RET    NZ
       POP    HL          ;Pop return to KEY
       POP    HL          ;Point to command buffer
       JP     @DSPLY            ;Display the old command
;
;      Tab    entered
;
$C8    PUSH   HL          ;Get pos on line
       CALL   ADDR_2_ROWCOL     ;Get row,col in HL
       LD     A,L         ;Xfer column to A
       POP    HL
       AND    7
       NEG                ;Negate and add tab
       ADD    A,8
       LD     E,A         ;Reg E has tab length
$C9    LD     A,B         ;Check on buffer full
       OR     A
       RET    Z
       LD     A,' '       ;Put spaces until
       LD     (HL),A            ;  tab expanded
       INC    HL
       CALL   DSPBYT
       RET    NZ
       DEC    B           ;Dec buffer remaining
       DEC    E           ;Dec tab count
       RET    Z
       JR     $C9
;
;      Exit KEYIN routine
;
$C10   SCF                ;BREAK exit with CF
$C11   PUSH   AF          ;Save flag
       LD     A,0DH       ;Stuff CR at end
       LD     (HL),A
       CALL   @DSP        ;  & display it
       LD     A,C         ;Calculate # of chars
       SUB    B           ;  entered
       LD     B,A
       POP    AF          ;Rcvr flag
       POP    HL          ;Restore buffer ptr
       RET
;
;      Byte I/O device handler
;             C => character if PUT or CTL
;             DE => Device Control Block
;
;*MOD
@CTL   PUSH   BC
       LD     B,4         ;Bit 2, CTL
```

```
      JR    IOBGN
@KEY  CALL  @KBD        ;Scan the keyboard
      RET   Z           ;Ret if key available
      OR    A           ;Return if error
      JR    Z,@KEY
      RET
@JCL  LD    DE,JCLCB$   ;JCL file FCB
      JR    @GET
@KBD  LD    DE,KIDCB$   ;Keyboard DCB
@GET  PUSH  BC
      LD    B,1         ;Bit 0, GET
      JR    IOBGN
@PRT  LD    DE,PRDCB$   ;Printer DCB
      JR    @PUT
@DSP  LD    DE,DODCB$   ;Video DCB
@PUT  PUSH  BC
      LD    B,2         ;Bit 1, PUT
IOBGN PUSH  IX          ;Save the registers
      PUSH  HL
      PUSH  DE          ;Xfer DCB to IX
      POP   IX
      PUSH  DE
      LD    C,A         ;Xfer the I/O char
      LD    HL,@RSTREG  ;Restore register routine
      LD    A,(LBANK$)  ;If bank 0 is not
      OR    A           ;  resident, need to
      JR    Z,$DO       ;  get it resident!
;
;     Some other bank is resident - invoke bank 0
;
      PUSH  BC          ;Save reg again
      XOR   A           ;Prepare for bank-0
      LD    B,A
      LD    C,A
      CALL  @BANK       ;Invoke bank-0
      LD    H,B         ;Get old bank data
      LD    L,C         ;  into reg HL
      POP   BC          ;Rcvr BC
      PUSH  HL          ;Bank data to stack
      LD    HL,RSTBNK   ;Set return address
$DO   PUSH  HL          ;  to restore registers
      LD    A,(DE)            ;P/u DCB type byte
      OR    A
      RET   Z           ;Back if nothing
      CP    8           ;Ck on GET/PUT/CTL
      JR    NC,@CHNIO   ;Branch if special
      LD    L,(IX+1)    ;  else p/u the vector
      LD    H,(IX+2)
$D1   LD    A,B         ;Xfer I/O code
      CP    2           ;Set flags state
      JP    (HL)
RSTBNK      POP   BC          ;Get old bank data
      PUSH  AF          ;Can't affect AF
      LD    A,C         ;Request to A
      CALL  @BANK       ;Bring back original bank
      POP   AF
@RSTREG     POP   DE          ;Restore regs
```

```
      POP   HL
      POP   IX
      POP   BC
      RET
;
$D2   PUSH  HL
      POP   IX
@CHNIO      LD    L,(IX+1)    ;P/u vector address
      LD    H,(IX+2)
$D3   LD    A,(IX+0)    ;P/u the DCB type
      OR    A           ;File Control Block?
      JP    M,@BYTEIO
      BIT   3,A         ;Test NIL bit 2nd
      JR    NZ,$D5
      BIT   4,A         ;Routed?
      JR    NZ,$D2            ;Go if routed DCB
      BIT   5,A         ;If not linked, then
      JR    Z,$D1       ;  must be filtered
      PUSH  HL          ;Point to the link DCB
      POP   IX
      LD    (IX+3),B    ;Save the direction
      PUSH  IX
      CALL  @CHNIO            ;I/O to 1st device
      POP   IX
      LD    B,(IX+3)    ;P/u the direction
      JR    NZ,$D6            ;Go on NZ flag
;
;     Z-flag on return - check input/output
;
      BIT   0,B         ;If input & got char,
$D4   LD    L,(IX+4)    ;  p/u the linked DCB
      LD    H,(IX+5)
      JR    Z,$D2
$D5   CP    A
      RET
;
;     1st link got NZ condition - if input, get link
;
$D6   BIT   0,B         ;Was it input/output?
      JR    Z,$D7       ;Output is error
      OR    A           ;If A=0, then no input
      JR    Z,$D4
$D7   OR    A
      RET
      END
```
□

MULDIV/ASM

```
;MULDIV/ASM - 16 x 8 multiplication & division
      ADISP '<16 X 8 multiply/divide>'
;     ?
;*MOD
;
;     Multiply HL by A - SVC 91
;     HL => multiplicand
;     A => multiplier
;     HLA <= 24-bit result
;     DE destroyed
;
@MUL16      PUSH  BC          ;Save reg BC
      EX    DE,HL       ;Multiplicand to DE
      LD    C,A         ;  & multiplier to C
      LD    HL,0        ;Init value to zero
      LD    A,L         ;  in regs HLA
      LD    B,8         ;Init for 8-bit mult
$E1   ADD   HL,HL       ;Shift to next place
      RLA               ;Use A for bits 16-23
      RLC   C           ;Multiply this bit?
      JR    NC,$E2            ;Go if not
      ADD   HL,DE       ;Else add multiplicand
      ADC   A,0         ;  & any overflow to 16
$E2   DJNZ  $E1         ;Loop for 8 bits
      LD    C,A         ;Tempy save
      LD    A,L         ;Xfer low-order to A
      LD    L,H         ;Xfer mid-order to L
      LD    H,C         ;Xfer hi-order to H
      POP   BC
      RET
;
;     Divide HL by A - SVC 94
;     HL => dividend
;     A => divisor
;     HL <= resulting quotient
;     A <= remainder
;
;*MOD
@DIV16      PUSH  DE          ;Save this reg pair
      LD    D,A         ;Xfer divisor to D
      LD    E,16        ;Init for 16 bits
      XOR   A
$F1   ADD   HL,HL       ;Rotate dividend
      RLA               ;  & subtract divisor if
      JR    C,$F2       ;  carry into bit 16
      CP    D           ;Compare divisor
      JR    C,$F3       ;Go if no subtract
$F2   SUB   D           ;  else subtract divisor
      INC   L           ;Set lo-order
$F3   DEC   E           ;Count down one bit
      JR    NZ,$F1            ;Loop for 16 bits
      POP   DE
      RET
;
```

```
;      @HEXDEC - SVC 97
;      Routine to convert 16-bit hexadecimal to decimal
;      HL => value
;      DE => buffer pointer of 5-character buffer
;      HL <= destroyed (always set to zero)
;      DE <= buffer + 5
;      BC <= destroyed
;      Z <= set
;
;*MOD
@HEXDEC     LD    B,5         ;Length max
      LD    A,' '       ;Load blank
HEXDEC1     LD    (DE),A            ;To string
      INC   DE          ;Bump pointer
      DJNZ  HEXDEC1           ;Go for length
      PUSH  DE          ;Save end+1
      DEC   DE          ;Adjust back
HEXDEC2     LD    A,10        ;Base to convert to
      CALL  @DIV16            ;HL+A = HL/A
      ADD   A,'0'       ;Add ASCII to result
      LD    (DE),A            ;  to user string
      DEC   DE          ;Move back
;
;      Check if done
;
      LD    A,H         ;Get subtotal remainder
      OR    L           ;Done?
      JR    NZ,HEXDEC2  ;Go 'til completed
      POP   DE          ;Restore end+1
      RET               ;Return Z set
;
      END
□
```

CLOCKS/ASM

```
;CLOCKS/ASM - LS-DOS 6.2
      ADISP '<Heartbeat & Bank handling>'
;     ?
;*MOD
;
;     Model IV time clock & blinking cursor
;
TIMETBL     DB    60,60,24,30 ;Sec/min, min/hr, hr/day
TIMTSK$     EQU   $
      LD    A,(CRSAVE)  ;If cursor not on,
      OR    A           ;  then don't blink
      LD    HL,VFLAG$   ;Point to video flag
      JR    Z,$H2
                        ;Check if blinking
      BIT   7,(HL)            ;Check system INHIBIT
      RES   7,(HL)            ;Allow blink next time
      JR    NZ,$H2
      INC   (HL)        ;Increment the counter
      BIT   3,(HL)            ; & see if to 8
      JR    Z,$H2       ;Not this time
      RES   3,(HL)            ;Reset counter
      BIT   6,(HL)            ;Check if SOLID cursor
      JR    Z,NOSOLID   ;If not, then blink
      SET   5,(HL)            ;Force SOLID mode
NOSOLID CALL      ENADIS_DO_RAM     ;Bring up the video RAM
      LD    A,(HL)            ;Grab the toggle bit
      XOR   20H         ;  and flip it
      LD    (HL),A
      AND   20H         ;Was it on?
      LD    DE,(CURSOR) ;Get the cursor pos
      LD    A,(CRSAVE)  ;  and char under cursor
      JR    NZ,$H1            ;Put character if flip on
      LD    A,(CRSCHAR) ;  else put the cursor
$H1   LD    (DE),A            ;Put the char
$H2   LD    IX,TIMETBL  ;Point to data area
      DEC   (IX+3)            ;Count down by 30
      RET   NZ          ;Back if not one second
      IF    @HZ50
      LD    (IX+3),25   ;Set for 50 hertz
HERTZ$      EQU   $-1
      ELSE              ;  else use 60 hertz
      LD    (IX+3),30   ;Reset for one second
HERTZ$      EQU   $-1
      ENDIF
      BIT   4,(HL)            ;Is clock on? (VFLAG$)
      JR    Z,$H3       ;Go if off
      LD    DE,CLOCK    ;Set to display clock
      PUSH  DE
$H3   LD    B,3
      LD    HL,TIME$
      LD    DE,TIMETBL  ;Pt to max sec, min, hr
TIMER1      INC   (HL)        ;Bump time parm
      LD    A,(DE)            ;Constant value into A
      SUB   (HL)        ;Subtract timer datum
      RET   NZ          ;Ret if not max
      LD    (HL),A            ;  else set to 0
      INC   L           ;Pt to next datum
      INC   E
      DJNZ  TIMER1            ;Loop thru 3 parms
;
;     Update date at midnight
;
      LD    L,DATE$+1&0FFH    ;Point to day of the month
      LD    DE,MAXDAY$+1      ;Point to test table
      INC   (HL)        ;Bump the day
      INC   L           ;Point to month
      LD    A,(HL)            ;Get the month
      DEC   L
      DEC   A           ;Index into table
      ADD   A,E
      LD    E,A
      LD    A,(DE)            ;P/u max days
      CP    (HL)        ;Is day in range?
      RET   NC          ;Return if it is
      LD    (HL),1            ;  else reset day to 1
      INC   L           ;  & bump the month
      INC   (HL)
      LD    A,(HL)            ;If went past 'Dec',
      SUB   12+1        ;  then need to fix
      RET   C           ;  else return
      LD    (HL),1            ;Correct to 'Jan'
      DEC   L           ;Backup to year
      DEC   L
      INC   (HL)
      RET
;
;     Clock display processor
;
CLOCK EQU   $
      CALL  ENADIS_DO_RAM     ;Bring up the video
      LD    HL,CRTBGN$+69     ;CRT pos row 0, col 70
@TIME LD    DE,TIME$+2  ;Pt to hr of sc,mn,hr
      LD    C,':'       ;Set the separator
TIME1 LD    B,3         ;Init for 3 fields
TIME2 LD    A,(DE)            ;Get a field item
      LD    (HL),2FH    ;Init display
TIME3 INC   (HL)        ;Bump until proper digit
      SUB   10
      JR    NC,TIME3
      ADD   A,10+'0'    ;Add back 10, conv ASCII
      INC   HL          ;Bump to next display
      LD    (HL),A            ;  & stuff the digit
      INC   HL
      DEC   B
      RET   Z           ;Back when done8
      LD    (HL),C            ;  else stuff separator
      INC   HL
      DEC   DE          ;Pt to next time field
      JR    TIME2       ;  & loop
;
;     Return formatted date, HL => user buffer
```

```
;
@DATE LD    DE,DATE$+2  ;Pt to dy of yr,mn,dy
      LD    C,'/'
      JR    TIME1       ;Identical except HL
;
PCSAVE$     DW    00          ;PC at entry to RST 38H
;
;     Dynamic Trace routine
;
TRACE_INT   EQU   $
      DW    $+2   ;This TCB + 2
      LD    HL,(PCSAVE$)      ;Get interrupt PC value
      EX    DE,HL       ;Program counter to DE
      CALL  ENADIS_DO_RAM     ;Bring up the video
      LD    HL,CRTBGN$+62     ;CRT locn row 0, col 63
;
;     Hexadecimal display routine
;
@HEX16      LD    A,D         ;Convert reg D to
      CALL  @HEX8       ;  two hex digits
      LD    A,E         ;Convert reg E to
@HEX8 PUSH  AF          ;  two hex digits
      RRA               ;Do left nybble first
      RRA
      RRA
      RRA
      CALL  HXD1        ;Bits 0-3 stuffed in hex
      POP   AF          ;Recall the byte
HXD1  AND   0FH         ;  & use right nybble
      ADD   A,90H       ;Convert nybble to hex
      DAA
      ADC   A,40H
      DAA
      LD    (HL),A            ;Stuff in (HL)
      INC   HL
      RET
;
;     Scan for PAUSE or BREAK & set KFLAG$
;
SHIFT EQU   0F480H
      IF    @USA
KB1   EQU   0F401H
      ENDIF
      IF    @GERMAN
KB1   EQU   UNKNOWN
      ENDIF
      IF    @FRENCH
KB1   EQU   UNKNOWN
      ENDIF
KB7   EQU   0F440H
KCK@  CALL  ENADIS_DO_RAM     ;Bring up the keyboard
      LD    HL,KFLAG$   ;Hang onto flag
      LD    A,(SHIFT)   ;P/u SHIFT row & ignore
      AND   7           ;  CTRL key pressed
      CPL
      BIT   2,A
      RET   Z           ;Back if CTRL
```

```
;
;       Set carry flag if a SHIFT key is down
;
        ADD     A,1             ;Set CF if no SHIFT
        CCF                     ;Set CF if SHIFT
        JR      NC,KCK1                 ;No pause if no SHIFT
        LD      A,(KB1)                 ;Test for "@"
        IF      @USA
        BIT     0,A
        ENDIF
        IF      @INTL
        BIT     4,A             ;Foreign keyboard
        ENDIF
        JR      Z,KCK1A                 ;Bypass if no "@"
        SET     1,(HL)                  ;Turn on pause bit
        JR      KCK1A
;
;       Inhibit test of unshifted BREAK if nested ENA_DO
;
KCK1    LD      A,(OPREG_SV_PTR) ;If not at highest level
        SUB     0FFH&(OPREG_SV_AREA+1) ;  then don't allow
        JR      NZ,KCK1B        ;  tasker BREAK handler
KCK1A   LD      A,(KB7)                 ;Check on BREAK & ENTER
        BIT     0,A             ;Check on ENTER
        JR      Z,KCK1B                 ;Go if not
        SET     2,(HL)                  ;  else note set
KCK1B   BIT     2,A             ;Is <BREAK> depressed?
        PUSH    AF
        JR      Z,KCK2                  ;Go if not
        JR      C,KCK2                  ;Ignore if unshifted
        LD      A,(SFLAG$)      ;Permit break bit only
        BIT     4,A             ;  if BREAK enabled?
        JR      NZ,KCK2
        SET     0,(HL)                  ;Turn on BREAK bit
KCK2    POP     AF              ;C=shift, NZ=break
        RET
;
;       Routine to enable video RAM & change stack if necessary
;
;*MOD
ENADIS_DO_RAM     EQU   $
        DI                      ;Can't while we test stack
        LD      (HLSAV),HL      ;Save HL but not on stack
        PUSH    AF              ;Save AF
        POP     HL
        LD      (AFSAV),HL
        LD      HL,0F3FCH.XOR.-1 ;Can't exceed X'F3FC'
        ADD     HL,SP
        JR      NC,$I1
;
;       Switch to the system stack
;
        POP     HL              ;Transfer RET address
        LD      (SPSAV),SP      ;Save stack pointer
        LD      SP,STACK$-20H           ;Keep room at top
        PUSH    HL              ;Put RET back
$I1     LD      HL,DIS_DO_RAM           ;Stack return to disable
```

```
      EX    (SP),HL           ;  video RAM below RET
      PUSH  HL
      LD    HL,OPREG_SV_AREA
OPREG_SV_PTR      EQU   $-2
      INC   HL          ;Get next save location
      LD    A,(OPREG$)  ;P/u port mask
      JR    NC,$I2            ;Bypass if NC (no stack switch)
      AND   7FH         ;Strip bit 7 to use as flag
$I2   LD    (HL),A            ;Save current state
      AND   0FCH        ;Strip SEL1 & SEL0
      OR    82H         ;Set SEL1,0 = (1,0) & NZ cond
      JR    DOOPREG           ;Set new assignment
;
;     Routine to disable video RAM
;
DIS_DO_RAM  EQU   $
      DI                ;Interrupts off
      LD    (HLSAV),HL  ;Save off of stack
      PUSH  AF
      POP   HL          ;Save AF
      LD    (AFSAV),HL
      LD    HL,(OPREG_SV_PTR)
      LD    A,(HL)            ;P/u previous state
      BIT   7,A         ;Test if we switch stack
      SET   7,A         ;Make sure PAGE is set
      DEC   HL
;
DOOPREG     LD    (OPREG_SV_PTR),HL
      LD    (OPREG$),A  ;Restore port image
                        ;  and the port
      OUT   (84H),A
      JR    NZ,$I3
;
;     Switch back to the old stack
;
      LD    SP,$-$            ;Get the old stack
SPSAV EQU   $-2
$I3   LD    HL,$-$
AFSAV EQU   $-2
      PUSH  HL          ;Restore AF
      POP   AF
      LD    HL,$-$            ;Restore HL
HLSAV EQU   $-2
      EI                ;Interrupts back on
      RET
OPREG_SV_AREA     EQU   $-1
      DB    0,0,0,0,0,0,0,0
;
;     Bank selection SVC handler
;     HL=> Transfer address for function B=0
;     C => Bank request <0-2>; Set bit 7 to transfer
;     B => Request function
;           0 => Select bank C
;           1 => Reset in-use bit of bank C
;           2 => Test in-use bit of bank C
;           3 => Set in-use bit of bank C
;
```

```
;*MOD
@BANK EQU   $
      AND   7FH         ;Strip possible bit 7
      CP    2+1         ;Bank out of range?
      JP    NC,PERR             ;Parameter error
      DEC   B           ;Check option
      JP    M,$J3       ;Go if bank select
      LD    C,86H       ;Set for reset BUR$
      JR    Z,$J1       ;Go if function 1
      LD    C,46H       ;Set for test BUR$
      DEC   B
      JR    Z,$J1       ;Go if function 2
      DEC   B
      JR    Z,$J0       ;Go on set BUR$
      DEC   B
PERRX JP    NZ,PERR             ;SVC parameter error
      LD    A,(LBANK$)  ;P/u current bank
      CP    A
      RET
$J0   LD    B,A         ;Save the bank requested
      CALL  $J1         ;Test if in use already
      RET   NZ          ;Back if error
      LD    A,B         ;Recall the request #
      LD    C,0C6H              ;Set for set BUR$
$J1   AND   7           ;Strip to bank 0-7
      RLCA              ;Shift <0-2> to <3-5>
      RLCA
      RLCA
      OR    C           ;Merge the code type
      LD    ($J2+1),A   ;Change the OP code
      XOR   A           ;Init Z flag
      LD    A,8         ;Init "Device not avail
      PUSH  HL          ;Don't alter HL
      LD    HL,BUR$             ;Point to bank-used RAM
$J2   BIT   0,(HL)              ;\\This opcode is altered
      POP   HL
      RET
$J3   PUSH  HL          ;Ck if stack is in upper
      LD    HL,8005H    ;  bank area
      ADD   HL,SP
      POP   HL
      JP    C,PERR              ;Error if > X'7FFB'
      CP    1           ;Change <0, 1, 2, 3>
      RLA               ;  to <1, 2, 4, 6>
      LD    B,A         ;  & save for later
      LD    A,(BAR$)    ;P/u Bank Avail RAM
      AND   B           ;Is the bank installed?
      JR    NZ,PERRX    ;Error if not in machine
      LD    A,B         ;Get the requested bank
      RRA               ;Change <1, 2, 4> to
      CCF               ;  <0, 2, 3> {CF on 0
      ADC   A,0         ;  switched to 2 & 4}
      RLCA              ;Shift bits 0-1
      RLCA              ;  to 4-5 (MBIT0,1)
      RLCA
      RLCA
      LD    B,A         ;Save bit mask
```

```
        LD     A,(OPREG$)  ;P/u current memory
        AND    08FH        ;  configuration &
        OR     B           ;  mask off old &
        LD     (OPREG$),A  ;  merge the new
        OUT    (84H),A           ;Switch the hardware
        LD     A,(LBANK$)  ;Get the old bank #
        LD     B,A         ;  & save it
        LD     A,C         ;P/u new bank #
        AND    7FH         ;Strip any bit 7
        LD     (LBANK$),A  ;  & save new bank #
        XOR    C           ;Keep bit 7
        OR     B           ;Merge in new bank #
        LD     C,A         ;  & replace into C
        BIT    7,C         ;Transfer to new bank?
        LD     B,0         ;Init for invoke later
        RET    Z           ;No if bit 7 = 0
        EX     (SP),HL           ;Exchange RET with new
        CP     A           ;  transfer & go to it
        RET
        END
```

KIDVR/ASM

```
;KIDVR/ASM - LS-DOS 6.2
      ADISP '<Keyboard Driver>'
;     ?
;*MOD
;
LF    EQU   10
CR    EQU   13
KB0   EQU   0F401H            ;Row 0 RAM address
KB6   EQU   0F440H            ;Row 1 RAM address
SHIFT EQU   0F480H            ;Row 7 RAM address
;
KIDVR JR    KIBGN       ;Branch around linkage
      DW    KILAST            ;Last byte used
      DB    3,'$KI'
      DW    KIDCB$            ;Pointer to DCB
      DW    0           ;Spare
KIDATA$     DB    0           ;Last key entered
      DB    0           ;Repeat time check
RPTINIT     EQU   $-KIDATA$
      DB    22          ;22 * 33.3ms = .733 sec
RPTRATE     EQU   $-KIDATA$
      DB    2           ;2 x RTC rate
KBROW0      EQU   $-KIDATA$
      DB    -1,-1,-1,-1 ;Image of rows 0-3
KBROW4      EQU   $-KIDATA$
      DB    -1,-1       ;Image of rows 4-5
KBROW6      EQU   $-KIDATA$
      DB    -1,-1       ;Image of rows 6-7
;
;     Conversion table for keyboard row 7/8
;
KBTBL DB    CR,1DH,1FH,1FH    ;<ENTER> <CLEAR>
      DB    80H,0,0BH,1BH     ;<BREAK> <UPARW>
      DB    LF,1AH,8,18H      ;<DNARW> <LTARW>
      DB    9,19H,20H,20H     ;<RTARW> <SPACE>
      DB    81H,91H,82H,92H   ;<F1> <F2>
      DB    83H,93H           ;<F3>
;
;     Table to generate 5B-5F, 7B-7F
;
SPCLTB      DB    ',/.;',CR
;
;     Entry to keyboard driver
;
KIBGN LD    A,C         ;Get the character
      PUSH  AF          ;Save flags
      CALL  @KITSK            ;Hook for KI task
      POP   AF
;
;     Screen print (Control-*) processing
;
      CALL  TYPAHD            ;Chain downstream
      RET   NC          ;Ret if not <CONTROL>
      PUSH  AF          ;Save flag state
      CP    ':'
      JR    Z,$K1       ;Go if screen print
      POP   AF
      RET
;
;     Perform a screen print
;
$K1   POP   AF          ;Clean the stack
      LD    A,(DFLAG$)  ;Check on Graphic bit
      RLCA
      LD    A,3EH       ;Init for LD A,'.'
      JR    NC,$+4            ;Go if not Graphic
      LD    A,0FEH            ;Change to CPR n
      LD    ($K4),A           ;Stuff cpr or ld
      LD    HL,KFLAG$   ;Reset the BREAK bit
      RES   0,(HL)
      PUSH  HL          ;Save on stack
      LD    HL,0        ;Init for row,col
$K2   LD    B,1         ;Get a character at the
      CALL  @VDCTL            ; row-H, col-L
      JR    NZ,$K6            ;Go on error
      CP    20H
      JR    NC,$+4            ;Convert control codes
      ADD   A,40H       ;  to cap A-Z, +
      CP    80H         ;Cvrt anything from X'80'
      JR    C,$K5       ;  thru X'FF' to a '.'
$K4   LD    A,'.'       ;  unless graphic bit set
$K5   CALL  @PRT        ;Print the char & loop
      JR    NZ,$K6
      INC   L           ;Bump column counter
      LD    A,L         ;Check for end-of-line
      SUB   80
      JR    NZ,$K2            ;Loop if not EOL
      LD    L,A         ;Reset to column 0
      DEC   L           ;Adj for CR force
      EX    (SP),HL           ;Get KFLAG$
      BIT   0,(HL)            ;Exit with A=0 on
      EX    (SP),HL           ;  on entrance of BREAK
      JR    NZ,$K6
      INC   H           ;Bump row counter
      LD    A,H         ;Test for end of screen
      CP    24
      LD    A,CR
      JR    NZ,$K5            ;Put the CR & loop
$K6   LD    A,CR        ;Close out with CR if
      CALL  @PRT        ;  BREAK key detected
      POP   HL          ;Pop the KFLAG
      RES   0,(HL)            ;  & reset BREAK bit
      JR    NOCHAR
;
;     Driver to scan the keyboard
;
;*MOD
KISCAN      LD    IX,KIDATA$  ;Point to data area
      LD    HL,KIDATA$+KBROW0 ;Load kbd image start
      LD    BC,KB0            ;Load start of keyboard
      LD    D,0         ;Zero the key counter
```

```
$L1   LD    A,(BC)            ;Load 1st char from kbd
      LD    E,A
      XOR   (HL)        ;XOR with old value
      JR    NZ,$L2            ;Go if different
      INC   D           ;Bump key counter
      INC   HL          ;Bump image pointer
      RLC   C           ;Go to next row
      JP    P,$L1       ;Loop until end of rows
      LD    A,(BC)            ;Get row 7
      AND   078H        ;Strip SHIFT, CTL
      LD    E,A
      XOR   (HL)
      JR    NZ,$L2
      LD    A,(IX+0)    ;Keydown? It's same as
      OR    A           ;  the last if so
      JR    Z,NOCHAR    ;Ret if no key
      LD    A,(TIMER$)  ;Do we repeat the
      SUB   (IX+1)            ;  same key?
      JR    Z,$L10            ;Go repeat if time up
      SUB   (IX+RPTINIT)      ;Beyond .75 seconds?
      JR    C,$L10            ;Go if yes
NOCHAR      OR    1           ;Else don't repeat
      LD    A,0         ;Show NZ with A=0
      RET
;
;     Found a change in the key matrix
;
$L2   LD    (HL),E            ;Stuff KB image with new
      AND   E           ;  KB row value
      JP    Z,NOKEY           ;Go if new is none
;
;     Convert the depressed key
;
      LD    E,A         ;Save the active bit
      LD    A,D         ;Calculate 8 * row
      RLCA
      RLCA
      RLCA
      LD    D,A         ;Save 8 * row
      LD    C,1         ;Add 8 * row + column
$L3   LD    A,C
      AND   E           ;Check if bits match
      JR    NZ,$L6            ;Go if match
      INC   D           ;  else bump value
      RLC   C           ;Shift compare bit
      JR    $L3         ;Loop to test next
;
;     Key pressed was not an alpha
;
$L4   SUB   90H         ;Adjust for non-alpha
      JR    NC,$L9            ;Go if special key
      ADD   A,40H       ;Cvrt to numeric/symbol
      CP    3CH         ;Manipulate to get
      JR    C,$L5       ;  proper code
      XOR   10H         ;Flip bit 4
$L5   BIT   0,E         ;Check SHIFT
      JR    Z,$L11            ;Go if unshift
      XOR   10H         ;  else adjust for SHIFT
      JR    $L11
;
;     Found a key - Set up the function codes
;
$L6   LD    A,(SHIFT)   ;P/u the SHIFT key
      LD    E,A         ;Merge RH and LH shift keys
      AND   2           ;Only merge bit 1
      RRCA              ;Bit 1 to bit 0
      OR    E           ;Merge bits 0 & 1
      LD    E,A         ;Value of (RHorLF) shift
      LD    A,D         ;Load semi-converted
      ADD   A,60H       ;If alpha, convert to
      CP    80H         ;  correct value
      LD    HL,KFLAG$
      JR    NC,$L4            ;Go if not alpha
;
;     Alpha <@-Z> - If caps lock or <SHIFT>,
;     Convert to caps unless CLEAR
;
      BIT   2,E         ;CTRL key down?
      JR    NZ,CTLA2Z   ;CTRL sets <00-1A>
      CP    60H         ;Invert @ and `
      JR    NZ,$L7
      XOR   20H         ;Invert and bypass test
      JR    $L8         ;  for CAPS lock
$L7   BIT   1,(IX+KBROW6)     ;If clear don't test
      JR    NZ,$L8            ;  for CAPS lock
      BIT   5,(HL)            ;Caps lock?
      JR    NZ,TGLCASE
$L8   BIT   0,E         ;SHIFT key down?
      JR    Z,$L11            ;Bypass if not shifted
      JR    TGLCASE           ;Convert to upper case
CTLA2Z      SUB   60H         ;Convert CTRL A-Z
      JR    NZ,$L11           ;Go on A-Z
      BIT   0,E         ;Shifted?
      SCF               ;Set C-flag for CTL-@
      RET   Z           ;  and return if unshifted
      LD    A,1CH       ;  else set EOF error
      RET
$L10  LD    A,(TIMER$)  ;Advance time check
      ADD   A,(IX+RPTRATE)    ;  by 0.067 seconds
      JR    $L12        ;Go output the key
;
;     Special keys - rows 6 & 7
;
$L9   CP    11          ;Compress F1-F3 keys
      JR    Z,CAPSKEY   ;  while checking for CAP
      JR    C,$+4       ;  F1-F3 to 8-10
      SUB   4
      LD    HL,KBTBL    ;Pt to special char table
      RLCA              ;Index into table,
      BIT   0,E         ;  shifted code is +1
      JR    Z,$+3
      INC   A
      LD    C,A         ;Index the table
      LD    B,0         ;Calculate position of
      ADD   HL,BC       ;  char in table
      LD    A,(HL)            ;Load char from table
      JR    $L11        ;Bypass restore of char
TGLCASE     XOR   20H         ;Toggle case, is bit 5
$L11  CP    80H         ;BREAK key?
      JR    NZ,$L11A    ;Ck on <BREAK> disable
      LD    HL,SFLAG$   ;Pt to System flag
      BIT   4,(HL)            ;<BREAK> key disabled?
      JR    NZ,$L11B    ;Bypass if so
      LD    HL,KFLAG$   ;Point to keyboard flag
      SET   0,(HL)            ; Set Break Pressed bit
      JR    $L11A
$L11B RLA               ;Rotate bit 7 out
$L11A BIT   1,(IX+KBROW6)     ;CLEAR key pressed?
      JR    Z,NOTALPH   ;Go if not down
      LD    D,A         ;Save code
      RES   5,A         ;Set to upper case for
      SUB   'A'         ;  test A-Z
      CP    'Z'-'A'+1   ; Compare to 26 decimal
      LD    A,D         ;Get back actual char
      JR    NC,$+4            ;Go if not A-Z
      XOR   20H         ;Shift keyboard case
      OR    80H         ;Set bit 7 for CLEAR key
NOTALPH     BIT   0,E         ;SHIFT key down?
      JR    Z,FIXCLR    ;Go if not
GOTSHFT     CP    9FH         ;Shift-clear?
      JR    Z,FIXSCL    ;Go if so
TSTSPA      CP    20H         ;Shift 0 or shift spcl?
      JR    NZ,KEYOK    ;Go if not
      BIT   0,(IX+KBROW4)     ;Ck zero key
      JR    Z,KEYOK           ;Go if not down
;
;     Toggle the caps lock bit in the KFLAG$
;
CAPSKEY     LD    A,20H       ;Caps wasn't 20H
CASHK$      LD    HL,KFLAG$   ;Reverse case by
      XOR   (HL)        ;  flipping bit 5
      LD    (HL),A
      JR    NOKEY
FIXSCL      XOR   80H         ;Reset bit 7
FIXCLR      CP    9FH         ;Clear key?
      JR    NZ,KEYOK    ;Go if not
NOKEY XOR   A
KEYOK LD    (IX+0),A
      LD    BC,0184H    ;Delay
TYPHK$      CALL  PAUSE@
      LD    A,(TIMER$)  ;Set initialization
DELAY2      ADD   A,(IX+RPTINIT)    ;  repeat key delay
$L12  LD    (IX+1),A    ;Save new repeat value
      LD    A,(IX+0)    ;Check if any key
      OR    A           ;  code was saved
      JP    Z,NOCHAR    ;Ret if none
      BIT   2,E         ;Shift key down?
      SCF               ;Set Carry Flag
      JR    NZ,SPECL    ;Ret if CTRL
      CCF               ;Complement C Flag
DVREXIT     BIT   7,A         ;Z flag set on non-CLEAR
```

```
      RET   Z           ;Go if not CLEAR+key
SPECL PUSH  AF          ;Save code
$L13  LD    HL,SPCLTB   ;Special char table
      RES   7,A         ;TURN OFF "CLEAR"
      LD    BC,5<8!5BH  ;5 chars, starting char
      JR    NC,$+3            ;  if not CTRL
      DEC   B           ;  else only 4
SPCLLP      CP    (HL)        ;Is this it?
      JR    Z,HIT       ;Go if so
      XOR   10H         ;Flip shift state
      CP    (HL)        ;Is that it?
      JR    Z,HITWS           ;Go if so (with shift)
      XOR   10H         ;Flip back
      INC   HL          ;Bump specl table ptr
      INC   C           ;Bump "convert to" char
      DJNZ  SPCLLP            ;Loop through table
      POP   AF          ;Not found in table
      JR    C,CKCTL2    ;Ck CTL for C flag
CKCTL1      CP    A           ;Set Z flag
      RET
HITWS SET   5,C         ;Move to LC set
HIT   POP   AF          ;Restore orig char
      LD    A,C         ;Load converted one
CKCTL JR    NC,CKCTL1   ;Go if ctl key not down
      AND   1FH         ;Force ctl code
CKCTL2      CP    A           ;Set Z flag
      SCF               ;Set C flag for CTRL
      RET
;
;     Check the type ahead buffer for any character
;
;*MOD
TYPAHD      EQU   $
      CALL  ENADIS_DO_RAM     ;Bring up Keyboard RAM
      LD    HL,TYPBUF   ;P/u start of type buffer
      LD    (HL),0FFH   ;Turn off type ahead
      JR    C,$M1       ;Go on @GET
      JR    Z,TYPON           ;No PUT to *KI
      CP    3           ;CTL 3 function?
      JP    Z,CLRTYP    ;Clear buffer if so
      INC   A
      JR    Z,CTLFF           ;Go if CTL 255 function
      XOR   A           ;Nothing done, No error
      JR    TYPON
;
;     Handle a CTL-255 - scan keyboard into user rowbuf
;
CTLFF EQU   $
      LD    HL,KB0            ;Start of keyboard image
      LD    B,8         ;Do 8 rows
$M0   LD    A,(HL)            ;P/u image
      LD    (IY+0),A    ; and Xfer to user buffer
      INC   IY
      RL    L           ;Pt to next higher row
      DJNZ  $M0
      RET
;
```

```
$M1   PUSH  HL
      INC   HL          ;Bump to PUT pointer
      LD    A,(HL)            ;  & pick it up
      INC   HL          ;Bump to GET pointer
      CP    (HL)        ;The same?
      JR    Z,$M4       ;Go if so
      PUSH  HL          ;Save pointer to GETPTR
      LD    E,(HL)            ;P/u offset to buffer
      INC   HL          ;Pt to buffer start
      LD    D,0         ;Add offset to start
      ADD   HL,DE       ;  to point to char posn
      LD    B,(HL)            ;GET the stored char
      POP   HL          ;Rcvr GETPTR
      INC   (HL)        ;Bump by one for char
      LD    A,80        ;Check for > 80
      CP    (HL)        ;  after INC
      JR    NC,$M2            ;Go if not at end
      LD    (HL),0            ;Reset to start of buf
$M2   LD    A,(HL)            ;If we emptied the
      DEC   HL          ;  type-ahead buffer,
      CP    (HL)        ;  update KFLAG$
      CALL  Z,R7KFLG    ;Reset bit 7 if empty
      POP   HL          ;Pointed to & get switch
      LD    (HL),0            ;Turn type back on
      LD    A,B         ;Transfer char/flag
      CP    A           ;Set the Z flag
      RET
;
;     No character in type ahead buffer - get from kbd
;
$M4   CALL  KISCAN            ;Call keyboard driver
      POP   HL          ;Rcvr switch
TYPON LD    (HL),0            ;Type ahead back on
      RET
;
;     Type ahead task 10 - scans keyboard and saves key
;
TYPTSK$     DW    $M5         ;Task entry for processor
$M5   LD    A,(DFLAG$)  ;If type-ahead suppressed
      AND   2H          ;  then return
      RET   Z
      CALL  ENADIS_DO_RAM     ;Bring up the keyboard
      LD    HL,TYPBUF   ;P/u type switch
      LD    A,(HL)            ;If previous driver is
      OR    A           ;  currently executing,
      RET   NZ          ;  do not stack more keys
      INC   HL          ;Bump to PUTPTR
      PUSH  HL          ;  & save it
KIHOOK      CALL  KISCAN            ;  and scan for a character
      POP   HL
      RET   NZ          ;Ret if no char
      PUSH  AF          ;  else Xfer char
      POP   BC          ;  & flag to BC
      CP    80H         ;Check for <BREAK>
      PUSH  AF
      PUSH  HL
      CALL  Z,$M6       ;If so clear type buf
      POP   HL           ;Restore regs
      POP   AF
      CP    0C0H         ;If CLEAR @, reset keybuf
      JR    Z,$M6
      LD    E,(HL)             ;P/u PUTPTR & compare
      LD    A,E          ;GETPTR
      INC   HL
      CP    (HL)
      JR    Z,$M8        ;Jump if keybuffer empty
      LD    A,(TIMER$)  ;Check if we expired the
      ADD   A,(IX+RPTRATE)    ;  time interval between
      CP    (IX+1)            ;  repeating keys
      JR    NZ,$M7            ;Go if time not up
      ADD   A,(IX+RPTRATE)    ;Re-adjust time check so
      LD    (IX+1),A    ;  we don't repeat in
      RET               ;  type-ahead task
;
;     CLEAR @ control key entered, clear the buffer
;
CLRTYP      INC   HL           ;Bump to PUT pointer
$M6   XOR   A
      LD    (HL),A             ;1st PUT is loc'n 0
      INC   HL           ;Pt to GETPTR
      LD    (HL),A             ;1st GET is loc'n 0
R7KFLG      LD    HL,KFLAG$   ;Show buffer empty
      RES   7,(HL)
      RET
;
;     Char to stuff - check if buffer will overflow
;
$M7   LD    A,E          ;P/u current PUT pointer
      INC   A            ;If the next loc'n wraps
      CP    (HL)         ;  to the GET loc'n,
      RET   Z            ;  don't permit overrun
$M8   PUSH  HL           ;Save ptr to GETPTR
      INC   HL           ;Pt to start of keybuf
      LD    D,0          ;  & calculate PUT loc'n
      ADD   HL,DE
      LD    (HL),B             ;Store the char
      LD    HL,KFLAG$   ;Show type buffer
      SET   7,(HL)             ;  is not empty
      POP   HL           ;Rcvr ptr to GETPTR
      DEC   HL           ;Back up to PUTPTR
      INC   (HL)         ;Bump past the char
      LD    A,80         ;Check for >80
      CP    (HL)
      RET   NC           ;Back if not over 80
      LD    (HL),D             ;  else reset to 1st
      RET               ;  position in buf (0)
;
;     Type ahead buffer area
;
TYPBUF      EQU   0FF80H
;
;     TYPBUF+0 = On/Off flag
;     TYPBUF+1 = Storage pointer
;     TYPBUF+2 = Retrieve pointer
```

```
;	TYPBUF+3 = Start of actual buffer
;
KILAST	EQU	$-1
	END
```

DODVR/ASM

```
;DODVR/ASM - LS-DOS 6.2
      ADISP '<Video Driver>'
;     ?
;*MOD
@OPREG      EQU   84H         ;Mem mgt & video control
CRTCADD     EQU   88H         ;CRTC address port
CRTCDAT     EQU   89H         ;CRTC data port
LINESIZ     EQU   80
NUMROWS     EQU   24
NEGLINE     EQU   -LINESIZ
CRTSIZE     EQU   LINESIZ*NUMROWS
RAMSIZE     EQU   2048
CRTBGN$ EQU 0F800H
CRTEND      EQU   CRTBGN$+CRTSIZE-1
;
;     Driver entry point
;
DODVR JR    DOBGN       ;Branch around linkage
      DW    DOEND       ;Last memory location used
      DB    3,'$DO'
      DW    DODCB$            ;DCB used
      DW    0           ;Reserved
DODATA$     EQU   $
DO_MASK     EQU   $-DODATA$
SCRPROT EQU 7           ;Bits 0-2: scroll protect
TABS  EQU   3           ;Bit 3: 0=tabs, 1=chars
CTL   EQU   4           ;Bit 4, display controls
      IF    @USA
      DB    0
      ENDIF
      IF    @INTL
      DB    08          ;Space compression off
      ENDIF
CURSOR      DW    CRTBGN$
CRSAVE      DB    20H         ;Character under cursor
CRSCHAR     DB    '_'         ;Cursor character
;
;     Entry from SVC 15, @VDCTL
;
@VDCTL      JP    @_VDCTL
;
;     Continue regular driver functions
;
DOBGN LD    IX,DODATA$
      CALL  ENADIS_DO_RAM     ;Bring up the video RAM
      JP    C,$N0       ;Go on 'GET' request
      CALL  $N0         ;Handle cursor
      PUSH  BC          ;Need to save C
      LD    A,C         ;Get char to display
      BIT   CTL,(IX+DO_MASK) ;Display controls set?
      JR    NZ,$N1A           ;Go if so
      OR    A           ;Char a 0?
      JP    Z,TGGLCTL   ;Switch Bit CTL if so
      CP    20H         ;Video control char?
      JP    C,DO_CONTROL      ;Go if so
$N1A  CP    0C0H        ;Tab or special?
      JR    C,DONORM    ;Go on normal characters
;
;     Character is => 0C0H
;
      BIT   TABS,(IX+DO_MASK) ;Tabs or spec chars
      JR    Z,DO_TABS   ;Go if video tabs
;
;     Character is not tab expansion - do it
;
DONORM      CALL  DO_DSPCHAR  ;Display the char
      RES   CTL,(IX+DO_MASK) ;Turn off CTL bit
DO_RET      POP   BC          ;Get orig char
DO_RETI     DI                ;Disable intr
      LD    A,(CRSAVE)  ;If a cursor is on, then
      OR    A           ;  we need to save the
      JR    Z,$N1       ;  current char & display
      LD    A,(DE)            ;  the cursor character
      LD    (CRSAVE),A  ;Save current char
      LD    A,(VFLAG$)  ;Allow tasker to blink
      RES   7,A
      LD    (VFLAG$),A
      LD    A,(CRSCHAR) ;P/u cusor character
      LD    (DE),A            ;Put it on the screen
$N1   LD    (CURSOR),DE ;Update cursor position
      CP    A           ;Clear status
      LD    A,C         ;Restore the char
      RET
;
;     Perform a tab expansion {C0H-FFH}
;
DO_TABS     EQU   $
      SUB   0C0H        ;Compute spaces
      JR    Z,DO_RET    ;Forget it if TAB(0)
      LD    B,A         ;Display requested
$N2   LD    C,' '       ;  number of spaces
      CALL  DO_DSPCHAR
      DJNZ  $N2
      JR    DO_RET
;
;     Routine to move the cursor to begin of line {29}
;
CRSBOL      EQU   $
      EX    DE,HL       ;Cursor addr to HL
      CALL  ADDR1       ;Find row,col
      LD    L,A         ;set col to start
      JP    ROWCOL_2_ADDR     ;Calc address of BOL
;
;     Routines to turn on/off the cursor {14/15}
;
CRSON LD    A,(DE)            ;Get screen character
CRSOFF      LD    (CRSAVE),A  ;Save zero or CRT char
      RET
;
;     Routine moves bursor to start of video page {28}
;       set to 80 column, and turns off inverse video
```

```
;
CRSHOME     EQU   $
      LD    DE,CRTBGN$  ;Home the cursor
      LD    A,(MODOUT$) ;P/u the mask &
      AND   0FBH        ;  set to 80 cpl
      CALL  SETMOD
      JR    DO_INVERT_DIS     ;Set to normal video
;
;     Routine to backspace & erase cursor {08}
;
BACKSPA     EQU   $
      CALL  CRSBKSP           ;Backspace the cursor
      RET   Z           ;if not at start,
      LD    C,' '       ;  put a space at
      JP    PUT_@       ;  at the new loc'n
;
;     Routine to backspace the cursor {24}
;
CRSBKSP     EQU   $
      LD    A,(MODOUT$) ;If double width chars,
      AND   4           ;  need to do twice
      CALL  NZ,$+3
      LD    HL,CRTBGN$  ;See if at home position
      SBC   HL,DE       ;  prior to adjusting
      RET   Z
      DEC   DE          ;Decrement the cursor pos
      RET
;
;     Routine to move the cursor up one line {27}
;
CRSUP EQU   $
      LD    HL,NEGLINE  ;Move up one line
      JR    MOVCRS
;
;     Routine to move the cursor down on line {26}
;
CRSDOWN     EQU   $
      LD    HL,LINESIZ  ;Add the line length
MOVCRS      ADD   HL,DE       ;  to the current pos
      LD    A,H         ;Make sure we did not
      CP    CRTBGN$>8   ;  go over the top
      RET   C
      EX    DE,HL       ;  & switch back to DE
      DEC   DE          ;Adjust for fall thru
      JP    CRSFRW0
;
;     Set to 40 cpl mode {23}
;
SET40 LD    A,(MODOUT$) ;Get image of the port
      OR    04H         ;Merge in 40 cpl bit
      JR    SETMOD
;
;     Routines to parse control functions
;
DO_CONTROL  EQU   $
      LD    HL,DO_RET   ;Establish RET
      PUSH  HL
      CP    08H         ;Backspace?
      JR    Z,BACKSPA
      CP    0AH         ;Line feed?
      JR    Z,$+4       ;  is same as <ENTER>
      SUB   0DH         ;Carriage return?
      JP    Z,LINFEED
      DEC   A           ;Cursor on?
      JR    Z,CRSON
      DEC   A           ;Cursor off?
      JR    Z,CRSOFF
      DEC   A           ;Reverse video?
      JR    Z,DO_INVERT_ENA
      DEC   A
      JR    Z,DO_INVERT_OFF
      SUB   4           ;Swap tab/alternate?
      JR    Z,TGGLTAB
      DEC   A           ;Special/alternate?
      JR    Z,TGGLALT
      DEC   A           ;40 cpl?
      JR    Z,SET40
      DEC   A           ;Cursor backspace?
      JR    Z,CRSBKSP
      DEC   A           ;Cursor forward?
      JR    Z,CRSFRWD
      DEC   A           ;Cursor down?
      JR    Z,CRSDOWN
      DEC   A           ;Cursor up?
      JR    Z,CRSUP
      DEC   A           ;Cursor home?
      JP    Z,CRSHOME
      DEC   A           ;Cursor BOL?
      JP    Z,CRSBOL
      DEC   A           ;Clear to EOL?
      JP    Z,CLREOL
      DEC   A
      JP    Z,CLREOF    ;Clear to end-of-frame?
      XOR   A           ;Clear A reg.
      RET
;
;     Routine to enable inverse video
;
DO_INVERT_ENA     EQU   $
      LD    B,8         ;Set for Enable
      DB    21H         ;Ignore next load
DO_INVERT_DIS     EQU   $
      LD    B,0
      LD    HL,(OPREG_SV_PTR) ;Real OPREG$
      LD    A,(HL)            ;P/u OPREG mask
      AND   0F7H        ;Strip bit 3
      OR    B           ;Set/reset invideo bit
      LD    (HL),A            ;  and restuff
      LD    A,B         ;Get mode mask byte
      RLCA              ;Rotate left 4 times to
      RLCA              ;  make an 8 into 80H
      RLCA              ;  for inverse on
      RLCA              ;Inverse off remains 0
DO_INVERT_OFF     EQU   $
```

```
      LD    (INVIDEO),A ;Set the mask byte
      RET
;
;     Routine to toggle display of controls
;
TGGLCTL     LD    HL,DO_RET   ;Establish ret addr
      PUSH  HL
      LD    A,10H       ;Toggle bit 4
      DB    21H         ;Ignore next
;
;     Toggle tabs & alternate character set
;
TGGLTAB     EQU   $
      LD    A,8         ;Toggle bit 3
      XOR   (IX+DO_MASK)      ;P/u mask value
      JR    SETMASK
;
;     Toggle special & alternate character set
;
TGGLALT     EQU   $
      LD    A,(MODOUT$) ;P/u port mask
      XOR   8           ;Flip the bit
SETMOD      LD    (MODOUT$),A ;Resave port mask
      OUT   (0ECH),A    ;  and send the byte
      RET
;
;     Display character <C> at current position
;
DO_DSPCHAR  EQU   $
      CALL  PUT_@       ;Display the Char
;
;     Routine to perform cursor forward {25}
;
CRSFRWD     EQU   $
      LD    A,(MODOUT$) ;If double width chars,
      AND   4           ;  need to do twice
      JR    Z,CRSFRW0
      INC   DE          ;Move cursor forward
CRSFRW0     INC   DE
      LD    HL,CRTEND   ;Off the screen?
      SBC   HL,DE
      RET   NC          ;Back if not
      CALL  CRSUP       ;Put cursor back on
      PUSH  DE          ;Save cursor position
DO_SCROLL   EQU   $
      LD    A,(IX+DO_MASK)    ;Get scroll protect
      AND   SCRPROT
      LD    HL,CRTBGN$  ;Point to CRT start
      LD    DE,CRTSIZE  ;P/u CRT size
      PUSH  BC
      LD    BC,LINESIZ  ;Set line size
      INC   A           ;Adjust scroll protect
$N4   ADD   HL,BC       ;Move logical start
      EX    DE,HL       ;  down one line
      OR    A           ;  and subtract one line
      SBC   HL,BC       ;  from the CRT size for
      EX    DE,HL       ;  each protected line
      DEC   A           ;Dec scroll protect
      JR    NZ,$N4            ;Loop until done
      PUSH  DE          ;Save the move length
      PUSH  HL          ;Save the move-from
      SBC   HL,BC       ;Move start back one
      EX    DE,HL       ;  line, Source =
      POP   HL          ;  start + one
      POP   BC          ;Get back dest locn
      LDIR              ;Scroll unprotected
      POP   BC          ;Recover line size
      JR    CLREOF1           ;Clear to EOF from DE
;
;     Set scroll protect value
;           C = scroll protect <0-7>
;           B = 7
;           SVC = 15, @VDCTL
;
SET_SCROLL  EQU   $
      LD    A,C         ;Get user value
      AND   7           ;Make modulo 8
      LD    C,A
      LD    A,(DODATA$) ;P/u current mask
      AND   0F8H        ;Remove current scroll
      OR    C           ;Merge in the new value
SETMASK     LD    (DODATA$),A ;  & reload mask
      XOR   A           ;Z-flag return
      RET
;
;     Routine to move down one line {10/13}
;
LINFEED     CALL  CRSBOL            ;Move to BOL
      PUSH  DE          ;Save cursor position
      CALL  CRSDOWN           ;Move down one line
      OR    A           ;Reset the carry flag
      LD    HL,CRTEND+1 ;  & check if off of
      SBC   HL,DE       ;  the screen
      JR    Z,DO_SCROLL ;Scroll if so
      POP   HL          ;Discard old position
CLREOL      PUSH  DE          ;Save new cursor pos
      CALL  CRSBOL            ;Get start of line
      LD    HL,79       ;Calculate end of line
      ADD   HL,DE       ;HL = end of line
      POP   DE          ;DE = current position
      PUSH  DE
      JR    CLREOF2           ;Clear the line
;
;     Clear to the end of the frame
;
CLREOF      PUSH  DE          ;Save current cursor pos
CLREOF1     LD    HL,CRTEND   ;Point to last RAM byte
CLREOF2     LD    A,(INVIDEO) ;P/u normal/reverse
      SET   5,A         ;  & make it a space
      LD    (DE),A            ;Stuff the "space"
      OR    A           ;Reset carry for subtract
      SBC   HL,DE       ;Calculate length
      JR    Z,CLREOF3   ;Back if at end already
      PUSH  BC
      LD    B,H         ;Xfer length to BC
      LD    C,L
      LD    H,D         ;Xfer start to HL
      LD    L,E
      INC   DE          ;Bump up by one
      LDIR              ;Propagate the space
      POP   BC
CLREOF3     POP   DE
      RET
;
;     Routine to stuff the video cursor RAM address
;
@VDCTL3     CALL  ROWCOL_2_ADDR     ;Calculate video address
      RET   NZ          ;Back on error
      DI                ;Disable any video tasks
      LD    (CURSOR),DE ;  until cursor is updated
      RET
;
;     Video control SVC processor
;
@_VDCTL     EQU   $
      CALL  ENADIS_DO_RAM     ;Bring up the video RAM
;
;     Test if in Task processor
;
      LD    A,(NFLAG$)  ;P/u NFLAG$
      BIT   6,A         ;Test for task process
      JR    NZ,VDCTL    ;If so skip setup
;
;     HANDLES @VDCTL    screen setup for normal use
;
      PUSH  DE
      CALL  $N0         ;Normalize character at cursor
      POP   DE          ;Recover value
      PUSH  DE
      CALL  VDCTL       ;Do function request
      PUSH  AF          ;Save the error status
      DI                ;Stop video tasks tempy
      LD    DE,(CURSOR)
      CALL  DO_RETI           ;Normalize screen and cursor
      POP   AF
      POP   DE
      RET
;
VDCTL LD    A,9         ;Check for VIDLINE,
      CP    B           ;  function 9
      JR    Z,VIDLIN
      LD    A,43        ;Prepare for user ERROR
      DEC   B
      JR    Z,GET_@_ROWCOL    ;<Ch> from row-H, col-L
      DEC   B
      JR    Z,PUT_@_ROWCOL    ;<Ch> to row-H, col-L
      DEC   B
      JR    Z,@VDCTL3   ;Set cursor to H,L
      DEC   B
      JR    Z,ADDR_2_ROWCOL ;Cursor row,col to H,L
      LD    DE,CRTBGN$  ;Init to start of video
      DEC   B
      JR    Z,VIDMOV1   ;User RAM to video
      DEC   B
      JR    Z,VIDMOVE   ;Video RAM to user
      DEC   B
      JP    Z,SET_SCROLL      ;Set scroll protect
      DEC   B
      RET   NZ          ;Return if bad request
;
;     Establish cursor character
;
      PUSH  HL
      LD    HL,CRSCHAR  ;Point to cursor char storage
      LD    A,(HL)            ;P/u current cursor character
      LD    (HL),C            ;  & update with new one
      POP   HL
      RET
;
;     VIDLIN routine function - 9 in register B
;
VIDLIN      LD    L,0         ;Always starts at col 0
      PUSH  DE          ;Save user buffer
      CALL  ROWCOL_2_ADDR     ;Get address into DE
      POP   HL          ;Recover user buffer
      RET   NZ          ;Quit on bad address
      INC   C           ;Check direction
      DEC   C           ;If Z then to screen
      JR    Z,MOVLIN    ;Set to go
      EX    DE,HL       ;Reverse direction
MOVLIN      LD    BC,LINESIZ  ;Set to go
      LDIR              ;Move it
      XOR   A           ;Z on RET
      RET
;
;     Routine to move video RAM
;
VIDMOVE     LD    A,H         ;Check on user buffer
      ADD   A,8         ;  not above X'0F800' &
      CP    24H+8       ;  not below X'2400'
      JR    C,PERR
      EX    DE,HL       ;Xchng user buffer,screen
VIDMOV1     LD    BC,CRTSIZE  ;Set for full screen xfer
      LDIR
      CP    A           ;Set Z flag
      RET
;
;     Routine to get the character at row,col
;
GET_@_ROWCOL      EQU   $
      CALL  ROWCOL_2_ADDR     ;Get Address of req
      LD    A,(DE)            ;P/u the character
      RET               ;Back on error or no error
;
;           Routine to halt blinking cursor & restore char
;
$N0   PUSH  HL
      LD    HL,VFLAG$
      SET   7,(HL)            ;Disable blinking cursor
      POP   HL
      LD    DE,(CURSOR) ;Get cursor pos in DE
      LD    A,(CRSAVE)  ;P/u saved character
      OR    A           ;If one is saved, put
                        ;  it on screen, else
      JR    NZ,PUTA@DE  ;  ignore it
      LD    A,(DE)            ;Cursor no ON but get
      RET               ;  character anyway
;
;     Routine to put a character at row,col
;
PUT_@_ROWCOL      EQU   $
      CALL  ROWCOL_2_ADDR     ;Get address of req
      RET   NZ          ;Back on error
PUT_@ LD    A,0         ;Merge in reverse video
INVIDEO     EQU   $-1
      OR    C
PUTA@DE     LD    (DE),A            ;Put the character
      CP    A           ;Set Z-flag for return
      RET
;
;     Routine to calculate cursor position from row,col
;
ROWCOL_2_ADDR     EQU   $
      LD    A,79        ;Logical line length
      CP    L           ;Compare to column pos
      JR    C,PERR            ;Error if > 79
      LD    A,H         ;P/u row number
      CP    24          ;Number of screen rows
      JR    NC,PERR           ;Error if > 24
      PUSH  HL
      PUSH  BC
      LD    C,L         ;Save column
      LD    B,CRTBGN$>8 ;Set to start of DO RAM
      LD    HL,LINESIZ
      CALL  @MUL16            ;Rows * line size
      LD    H,L         ;Shift to HL
      LD    L,A
      ADD   HL,BC       ;Add in col & RAM start
      EX    DE,HL       ;Address to DE
      POP   BC
      POP   HL
      XOR   A           ;Set Z flag
      RET
PERR  LD    A,43        ;SVC parameter error
      OR    A           ;Set NZ condition
      RET
;
;     Routine to get the row,col of video cursor
;
ADDR_2_ROWCOL     EQU   $
      LD    HL,(CURSOR) ;Get addr into HL
ADDR1 LD    A,H         ;Make address relative
      AND   7           ;  to logical 0 origin
      LD    H,A
      LD    A,LINESIZ   ;Set divisor
      CALL  @DIV16
      LD    H,L          ;Row to register H
      LD    L,A          ;Column to register L
      XOR   A            ;Set zero return code
      RET
DOEND EQU   $-1
      END
```

PRDVR/ASM

```
;PRDVR/ASM - LS-DOS 6.2
      ADISP '<Printer Driver>'
;     ?
;*MOD
PRPORT      EQU   0F8H
;
;     PR driver entry point
;     It passes X'00'-X'FF'
;     Unless INTL version
;
PRDVR JR    PRBGN       ;Branch around linkage
      DW    PREND       ;Last byte used
      DB    3,'$PR'
      DW    PRDCB$            ;Pointer to its DCB
      DW    0           ;Reserved
;
;     Driver code
;
PRBGN JR    Z,$O2       ;Go if output
      JR    C,$O1       ;Go if input req
;
;     Character CTL request
;
      LD    A,C         ;If CTL 0, return
      OR    A           ;  status else
      JR    Z,$O4       ;  treat as a GET
;
;     Character GET request
;
$O1   OR    0FFH        ;Set NZ flag
      CPL               ;  & A=0    to show
      RET               ;  no char available
;
;     Character PUT request
;
$O2   LD    DE,2000           ;Check status 2000 times
$O2A  CALL  $O4         ;PR ready?
      JR    Z,$O3       ;Go if so
;
;     Ten second time-out delay loop
;
      PUSH  BC          ;Printer was not ready
      LD    BC,340
      CALL  PAUSE@            ;Delay for a bit
      POP   BC
      DEC   DE          ;2000 times expired?
      LD    A,D
      OR    E
      JR    NZ,$O2A           ;Nope, contiune check
      LD    A,8         ;Device not Available"
      OR    A           ;Set NZ condition
      RET
$O3   EQU   $
;
```

```
      IF    @INTL
      LD    A,(IFLAG$)
      BIT   6,A          ;Special DMP PR?
      ENDIF
;
      LD    A,C
;
      IF    @INTL
      JR    Z,PVAL3
      CP    0C0H         ;Values C0-FF (-20H)
      JR    C,PVAL2            ;Go if less
      SUB   20H          ;Shift to European chars
      JR    PVAL3
PVAL2 CP    0A0H         ;A0-BF (+40H)
      JR    C,PVAL3            ;Go if less
      ADD   A,40H        ;Shift to graphics
      ENDIF
;
PVAL3 OUT   (PRPORT),A  ;Put out char
;
      IF    @INTL
      LD    A,C          ;Restore original
      CP    A            ;Set Z flag
      ENDIF
;
      RET
;
$O4   IN    A,(PRPORT)  ;Scan PR status
      AND   0F0H         ;Mask unused potions
      CP    30H          ;PR ready?
      RET                ;Return with answer
PREND EQU   $-1
      END
□
```

FDCDVR/ASM

```
;FDCDVR/ASM - LS-DOS 6.2
      ADISP '<Floppy Disk Driver>'
;     ?
;
;      HL=> buffer address
;       D=> track desired
;       E=> sector desired
;       C=> drive desired
;       B=> disk primitive command
;
WRNMIPORT   EQU   0E4H  ;NMI mask register
FDCADR      EQU   0F0H        ;FDC command
FDCSTAT     EQU   0F0H        ;FDC status
TRKREG      EQU   0F1H        ;FDC track register
SECREG      EQU   0F2H        ;FDC sector register
DATREG      EQU   0F3H        ;FDC data register
DSELCT      EQU   0F4H        ;Drive select port
;
;
;     Disk Driver Entry Point
;
FDCDVR      JR    FDCBGN            ;Branch to entry code
      DW    FDCEND            ;Last byte used
      DB    3,'$FD'           ;Module name
;
;     Automatic density recognition and retry density switch
;
SWDEN EQU   $
      LD    A,3         ;Check counter for 2
      CP    B           ;  tries left after this one
      JR    Z,RESTOR    ;If so try a RESTORE
;
      LD    A,(IY+3)    ;Flip the density bit,
      XOR   40H         ;  Bit 6, (IY+3)
      LD    (IY+3),A
      LD    BC,2409H    ;Set alloc to SDEN
      BIT   6,A         ;Test SDEN/DDEN
      JR    Z,SDEN            ;Do SDEN if it was DDEN
      LD    BC,4511H    ;  else set alloc to DDEN
SDEN  LD    (IY+7),C
      LD    (IY+8),B
      RET
;
;     Verify routine
;
VERFIN      LD    HL,BUCKET   ;Set byte bucket
      LD    A,2DH       ;Set for DEC L,...
      DB    1EH         ;Ignore next with LD E,n
;
;     Read routine
;
RDIN  XOR   A           ;Set for NOP
      LD    (CKVER),A
      CALL  RWINIT            ;Initialize
      LD    E,16H       ;Status mask
RDIN1 IN    A,(FDCSTAT) ;Get status
      AND   E           ;Loop until DRQ
      JR    Z,RDIN1           ;  or error
      INI               ;Grab byte
      DI
      LD    A,D         ;Get drive sel + WSGEN
RDIN2 OUT   (DSELCT),A  ;Initiate wait state
CKVER NOP               ;DEC L: if verify
      INI               ;Xfer byte
      JR    NZ,RDIN2    ;Loop then TSTBSY
;
;     Reselect drive while controller is busy
;
TSTBSY      IN    A,(FDCSTAT) ;Ck FDC status
      BIT   0,A         ;Busy?
      RET   Z           ;RET if not
      LD    A,(PDRV$)   ;P/u drive
      OUT   (DSELCT),A  ; & reselect
      JR    TSTBSY            ;Loop until idle
;
;     Driver start
;
FDCBGN      LD    A,B         ;P/u primitive request
      AND   A           ;NOP?
      RET   Z           ;Quit if so
      CP    7
      JR    Z,TSTBSY    ;Jump on TSTBSY request
      JP    NC,IORQST   ;Jump on I/O request
      CP    6
      JR    Z,SEEKTRK   ;Jump on track seek
      DEC   A
      JR    Z,SELECT    ;Jump on drive select
      INC   (IY+5)            ;Bump current cylinder
      CP    4
      LD    B,58H       ;FDC step-in command
      JR    Z,STEPIN
RESTOR      LD    (IY+5),0    ;Set to track 0
      LD    B,8         ;Restore drive
      JR    STEPIN
;
SELECT      CALL  TSTBSY            ;Check drive status
      RLCA              ;Bit 7 to Carry flag
      PUSH  AF          ;Save NOT READY flag
      PUSH  BC
      LD    A,(IY+3)    ;P/u SDEN/DDEN
      RLA               ;Bits left, then copy
      SRA   A           ;  bit 6=>7, bit 4=>4
      AND   90H         ;Keep only DDEN & side 1
      LD    C,A         ;Save the bits
      BIT   7,A         ;Check if SDEN or DDEN
      JR    Z,NOPCMP    ;No precomp if SDEN
      LD    A,(IY+9)    ;Set precomp on all
      CP    D           ;  tracks above DIR
      JR    NC,NOPCMP   ;No precomp if SDEN
      SET   5,C         ;Request precomp
NOPCMP      LD    A,(IY+4)    ;Get drive sel code
```

```
      AND   0FH         ;Keep only sel bits
      OR    C           ;Merge in bits 4,5,7
      POP   BC
      OUT   (DSELCT),A  ;Select drive
      LD    (PDRV$),A   ;Store port byte
      POP   AF          ;Retrieve NOT READY bit
      RET   NC          ;Ret if was ready
      BIT   2,(IY+3)    ;Check DELAY=0.5 or 1.0
      CALL  Z,FDCDLY    ;Double delay if 1.0
FDCDLY      PUSH  BC          ;Delay routine
      LD    B,7FH
      CALL  PAUSE@            ;Delay for B
      POP   BC
      RET
;
;     Routine to seek a track
;
SEEKTRK     CALL  TSTBSY            ;Wait until not busy
      LD    A,(IY+5)    ;P/u current cylinder
      OUT   (TRKREG),A  ;  & set FDC to current
      LD    A,(IY+7)    ;P/u alloc data
      AND   1FH         ;Get highest # sector
      SUB   E           ;Form req sector minus
      CPL               ;  max, setting CY flag if
      RES   4,(IY+3)    ;  init side select to 0
      JR    NC,SETSECT  ;Go if sector on side 0
      BIT   5,(IY+4)    ;If not 2 sided media,
      JR    Z,FRCSID0   ;  don't set side 1
      SET   4,(IY+3)    ;Set side 1
      DB    1EH         ;Ignore the next with LD E,n
SETSECT     LD    A,E         ;Restore unaltered sect
FRCSID0     OUT   (SECREG),A  ;Set sector
      LD    A,D
      OUT   (DATREG),A  ;Set desired track
      CP    (IY+5)            ;If at desired track,
      LD    B,18H       ;  use seek, else use
      JR    Z,STEPIN    ;  seek w/verify
      LD    (IY+5),D    ;Update current cylinder
      LD    B,1CH       ;Seek w/verify command
STEPIN      CALL  SELECT            ;Select drive
      LD    A,(IY+3)
      AND   3           ;Strip all but step rate
      OR    B
PASSCMD     OUT   (FDCADR),A  ;Give FDC its command
      LD    B,12H
      DJNZ  $           ;Wait
      XOR   A
FDCRET      RET
;
;     Read and write init routines
;
RWINIT      LD    A,D         ;Restuff track reg
      OUT   (TRKREG),A
      LD    A,(PDRV$)   ;Get select code
      OR    40H         ;Set WSGEN bit
      LD    D,A         ;Save code in D
      AND   10H         ;Get side select bit
      RRCA              ;  to bit 3
      BIT   1,C         ;Check if doing side cmp
      JR    NZ,GETCMD   ;Go if so
      XOR   A
GETCMD      OR    C
      LD    C,DATREG    ;Get port into C
      CALL  FDDINT$           ;Interrupts on or off?
      JR    PASSCMD           ;Pass command to ctrlr
;
;     I/O request handler
;
IORQST      BIT   2,B         ;Write command?
      LD    BC,(RFLAG$-1)     ;P/u retry count
      LD    C,82H       ;FDC cmd=readsec
      JR    NZ,WRCMD    ;Go if write command
      CP    10          ;Verify sector?
      JR    Z,VERFY
      CALL  GRABNDO           ;Grab next code & insert
      DB    1           ;ERROR code start
      DW    RDIN        ;Read entry point
VERFY CALL  GRABNDO           ;Stuff I/O direction
      DB    1           ;Error code start
      DW    VERFIN            ;Verify entry point
WRCMD BIT   7,(IY+3)    ;Software Write-Protect?
      JR    Z,WRCMD1    ;Bypass if not
      LD    A,15        ;Else set WP error
      RET
WRCMD1      LD    C,0A2H            ;Write sector FDC command
      CP    14          ;Directory sector?
      JR    C,DOWRIT
      LD    C,0A3H            ;Chg Data Address Mark
      JR    Z,DOWRIT    ;  if directory
      LD    C,0F0H            ;  else write track
DOWRIT      CALL  GRABNDO           ;Switch code
      DB    9           ;Error code start
      DW    WROUT       ;Write entry point
;
;     Routine stuffs error start byte & I/O vector
;
GRABNDO     EX    (SP),HL           ;Save HL & get ret addr
      LD    A,(HL)            ;P/u & stuff error code
      INC   HL          ;  start byte
      LD    (ERRSTRT+1),A
      LD    A,(HL)            ;Set up data transfer
      INC   HL          ;  direction vector
      LD    H,(HL)
      LD    L,A
      LD    (CALLIO),HL ;Stuff CALL vector
      POP   HL          ;Restore buffer addr
;
;     Main I/O handler routine
;
RETRY PUSH  BC          ;Save retry & FDC command
      PUSH  DE          ;Save track/sector
      PUSH  HL          ;Save buffer
      BIT   4,C         ;Test for track command
      CALL  Z,SEEKTRK   ;Seek if not track write
      CALL  TSTBSY            ;Wait until not busy
      CALL  0           ;Call inserted I/O routn
CALLIO      EQU   $-2         ;Data Xfer direction
DISKEI      NOP               ;Will be changed to EI
                        ;  after BOOT reads in SYS0
      IN    A,(FDCSTAT) ;Get status
      AND   7CH         ;Use only bits 2-6
      POP   HL
      POP   DE          ;Rcvr track & sector
      POP   BC          ;Rcvr retry count & cmd
      RET   Z           ;Return if no error
      BIT   2,A         ;Lost data?
      JR    NZ,RETRY    ;Don't count this retry
      PUSH  AF
      AND   18H         ;Record not found or CRC
      JR    Z,DISKDUN   ;No retries if otherwise
      BIT   4,A         ;Record Not Found?
      PUSH  BC          ;If so, switch
      CALL  NZ,SWDEN    ;  density or restore
      POP   BC
      POP   AF
      DJNZ  RETRY       ;Count down retry
      DB    6           ;Ignore next with LD B,n
DISKDUN     POP   AF          ;Adjust ret code
      LD    B,A
ERRSTRT     LD    A,0         ;Start with R=1, W=9
ERRTRAN RRC B           ;Bit number = err code
      RET   C           ;  is returned in A
      INC   A           ;Count each bit
      JR    ERRTRAN           ;  and loop until Carry
;
;     Write routine
;
WROUT CALL  RWINIT            ;Set up initialization
      LD    E,76H       ;Status mask
WRO1  IN    A,(FDCSTAT) ;P/u status
      AND   E           ;Fall out on DRQ or error
      JR    Z,WRO1            ;  else loop
      OUTI              ;Xfer byte to FDC
      DI                ;Now kill the interrupts
      IN    A,(FDCSTAT) ;Check for errors
      RRA               ;Did BUSY drop?
      RET   NC          ;Quit now if so
      LD    A,0C0H            ;Enable INTRQ and time out
      OUT   (WRNMIPORT),A
      LD    B,50H       ;Time delay for WRSEC
      DJNZ  $
      LD    B,(HL)            ;Get next byte early
      INC   HL
WRO3  LD    A,D         ;Enable wait states
      OUT   (DSELCT),A
      IN    A,(FDCSTAT) ;Check if timed out
      AND   E           ;Loop back if it timed
      JR    Z,WRO3            ;  out (must be WRTRK)
      OUT   (C),B       ;Pass 2nd byte
      LD    A,D         ;Get sel code + WSGEN bit
WRO2  OUT   (DSELCT),A  ;Pass until FDC times out
```

```
      OUTI              ;  & generates NMI
      JR    WRO2
      IF    $&0FFH.EQU.0FFH
      ADISP 'WARNING... BUCKET POSITION ERROR'
      ENDIF
BUCKET      DB    'S'
;
@RSTNMI     XOR   A           ;NMI vectors here
      OUT   (WRNMIPORT),A     ;Disable INTRQ & time out
      LD    BC,100            ;Delay for FDC sync
      CALL  PAUSE@            ;Call pause
      POP   HL          ;Discard return
      RET
FDCEND      EQU   $-1
      END
```

LDOS60/EQU

```
;LDOS60/EQU -Equates from cross reference of Lowcore
      ADISP '<LDOS60/EQU>'
;
@$SYS EQU   08F0H
@@1   DEFL  0000H
@@2   DEFL  0000H
@@3   DEFL  0000H
@@4   DEFL  0000H
@BANK EQU   0877H
@BYTEIO     EQU   1300H
@CHNIO      EQU   0689H
@CKBRKC     EQU   0553H
@CLS  EQU   0545H
@CTL  EQU   0623H
@DATE EQU   07A8H
@DIV16      EQU   06E3H
@DSP  EQU   0642H
@DSPLY      EQU   052DH
@FRENCH     EQU   0000H
@GERMAN     EQU   0000H
@GET  EQU   0638H
@HEX16      EQU   07BDH
@HEX8 EQU   07C2H
@HEXDEC     EQU   06F6H
@HZ50 EQU   0000H
@INTL EQU   0000H
@JCL  EQU   0630H
@KBD  EQU   0635H
@KEY  EQU   0628H
@KEYIN      EQU   0585H
@KITSK      EQU   0089H
@LOGER      EQU   0503H
@LOGOT      EQU   0500H
@MOD2 EQU   0000H
@MOD4 EQU   0FFFFH
@MSG  EQU   0530H
@MUL16      EQU   06C9H
@OPREG      EQU   0084H
@PRINT      EQU   0528H
@PRT  EQU   063DH
@PUT  EQU   0645H
@RSTNMI     EQU   0FE9H
@RSTREG     EQU   0680H
@TIME EQU   078DH
@USA  EQU   0FFFFH
@VDCTL      EQU   0B99H
@VDCTL3     EQU   0D38H
@_VDCTL     EQU   0D42H
ADDR_2_ROWCOL     EQU   0DF1H
BAR$  EQU   0201H
BOOTST$     EQU   439DH
BUR$  EQU   0200H
CASHK$      EQU   0A7BH
CFLAG$      EQU   006CH
```

```
CORE$ DEFL  0300H
CRTBGN$     EQU   0F800H
DATE$ EQU   0033H
DAYTBL$     EQU   04C7H
DCBKL$      EQU   0031H
DCT$  EQU   0470H
DFLAG$      EQU   006DH
DIS_DO_RAM  EQU   0846H
DODATA$     EQU   0B94H
DODCB$      EQU   0210H
DO_CONTROL  EQU   0C44H
DO_DSPCHAR  EQU   0CB8H
DO_INVERT_DIS     EQU   0C8CH
DO_INVERT_ENA     EQU   0C89H
DO_INVERT_OFF     EQU   0C9BH
DO_MASK     EQU   0000H
DO_RET      EQU   0BCBH
DO_RETI     EQU   0BCCH
DO_SCROLL   EQU   0CCEH
DO_TABS     EQU   0BEAH
DSKTYP$     EQU   04C0H
DTPMT$      EQU   04C2H
DVREND$     EQU   0FF4H
DVRHI$      EQU   0206H
ENADIS_DO_RAM     EQU   0817H
FDDINT$     EQU   000EH
FLGTAB$     EQU   006AH
GET_@_ROWCOL      EQU   0DAEH
HERTZ$      EQU   0750H
HIGH$ EQU   040EH
IFLAG$      EQU   0072H
INBUF$      EQU   0420H
INTVC$      EQU   003EH
JCLCB$      EQU   0203H
JLDCB$      EQU   0230H
KCK@  EQU   07D6H
KFLAG$      EQU   0074H
KIDATA$     EQU   08FCH
KIDCB$      EQU   0208H
LBANK$      EQU   0202H
MAXDAY$     EQU   0401H
MODOUT$     EQU   0076H
MONTBL$     EQU   04DCH
NFLAG$      EQU   0077H
OPREG$      EQU   0078H
OPREG_SV_AREA     EQU   086EH
OPREG_SV_PTR      EQU   0835H
PAKNAM$     EQU   0410H
PAUSE@      EQU   0382H
PCSAVE$     EQU   07AFH
PDRV$ EQU   001BH
PRDCB$      EQU   0218H
PUTA@DE     EQU   0DCDH
PUT_@ EQU   0DCAH
PUT_@_ROWCOL      EQU   0DC6H
RFLAG$      EQU   007BH
ROWCOL_2_ADDR     EQU   0DD0H
```

```
RSTOR$      EQU   04C4H
S1DCB$      EQU   0238H
SET_SCROLL  EQU   0CF3H
SFLAG$      EQU   007CH
SIDCB$      EQU   0220H
SODCB$      EQU   0228H
STACK$      EQU   0380H
START$      EQU   0000H
TIME$ EQU   002DH
TIMER$      EQU   002CH
TIMSL$      EQU   002BH
TIMTSK$     EQU   0713H
TMPMT$      EQU   04C3H
TRACE_INT   EQU   07B1H
TYPHK$      EQU   0A8FH
TYPTSK$     EQU   0B26H
VFLAG$      EQU   007FH
ZERO$ EQU   0401H
;
□
```

SYSRES/ASM

```
;SYSRES/ASM - LS-DOS 6.2
      ADISP '<SYSRES - LS-DOS 6.2>'
LF    EQU   10
CR    EQU   13
;
*LIST OFF               ;Xref of Lowcore
*REF  'LDOS60/EQU:1'
*LIST ON
*GET  'COPYCOM:1'       ;Embed copyright notice
;
      ADISP '<System low core assignments>'
;
;     LDOS 6.2 Low Core RAM storage assignments
;     Copyright (C) 1982 by Logical Systems, Inc.
;
START$      EQU   0
      ORG   0+START$
;
;     Page 0 - RST's, data, and buffers
;
@RST00      DI                ;IPL Entry for R/S 4-P
      LD    A,00000001B ;Set image in A
      OUT   (9CH),A           ;Toggle in BOOT/ROM
      DB    0,0,0       ;CP/M emulator SVC
@RST08      RET
      DW    0
SVCRET$     DW    0           ;Return address from SVC
LSVC$ DB    0           ;Last SVC executed
FDDINT$     DI                ;NOP or DI (F3H) for
      RET               ;  System (Smooth)
@RST10      RET
      DW    0
USTOR$      DS    5           ;User storage area
@RST18      RET
      DW    0
PDRV$ DB    1           ;Current drive, physical
PHIGH$      DW    0           ;Physical HIGH$
LOW$  DW    3000H       ;Lowest usable memory
@RST20      RET
      DW    0
LDRV$ DB    0           ;Current drive, logical
JDCB$ DW    0           ;Saved FCB pointer
JRET$ DW    0           ;Saved I/O return address
@RST28      JP    RST28       ;System SVC processor
TIMSL$      DB    55H         ;Fast=55, slow=FF
TIMER$      DB    0           ;RTC counter
TIME$ DS    3%0         ;SS:MM:HH storage area
@RST30      JP    @DEBUG            ;DEBUG call address
DATE$ DS    5           ;YY/DD/MM/packed
@RST38      JP    RST38@            ;Interrupt RST
OSRLS$      DB    00H         ;OS release #
;
;     INTIM$ stores the image read from RDINTSTATUS*
;
```

```
INTIM$      DB    0           ;Interrupt latch image
;
;     INTMSK$ masks the image read from RDINTSTATUS*
;     LDOS 6.x permits only RS-232 RCV INT, IOBUS INT,
;     and RTC INT to be used by the TASKER off of RST38
;
INTMSK$     DB    2CH         ;Mask for INTIM$
;
;     INTVC$ stores the eight vectors associated
;     with the INTIM$ bit assignments
;
INTVC$      DW    RETINST           ;Primary interrupts
      DW    RETINST,RTCPROC,RETINST
      DW    RETINST,RETINST,RETINST,RETINST
;
;     TCB$ stores the TCB vectors for task slots 0-11
;
TCB$  DS    24          ;Interrupt task vectors
;
;     NMI vector used in disk I/O
;
@NMI  DS    3           ;Don't overlay this
;
;     OVRLY$ stores the system's overlay request #
;
OVRLY$      DB    0           ;Current overlay resident
;
;     FLGTAB$ stores 26 flags and images. A pointer
;     to this table is obtained from SVC-@FLAGS
;
FLGTAB$     EQU   $
;
;
;     AFLAG$ - Start CYL for Allocation search
;
AFLAG$      DB    01          ;AFLAG
      DB    0           ;BFLAG
;
;     CFLAG$ assignments:
;      0 - Cannot change HIGH$ via SVC-100
;      1 - @CMNDR in execution
;      2 - @KEYIN request from SYS1
;      3 - System request for drivers, filters, DCTs
;      4 - @CMNDR to only execute LIB commands
;      5 - Sysgen inhibit bit
;      6 - @ERROR inhibit display
;      7 - @ERROR to use user (DE) buffer
;
CFLAG$      DB    0           ;Condition flag
;
;     DFLAG$ assignments:
;      0 - SPOOL is active
;      1 - TYPE ahead is active
;      2 - VERIFY is on
;      3 - SMOOTH active
;      4 - MemDISK active
;      5 - FORMS active
;       6 - KSM active
;       7 - accept GRAPHICS in screen print
;
DFLAG$      DB    00001010B   ;DEV Flag (SMOOTH,TYPE)
;
;      EFLAG$ - Assignments (sys13 usage)
;      use only bits 4, 5 and 6 to indicate user
;      entry code to be passed to SYS13. SYS13
;      will be executed from SYS1 if this byte
;      is NON/0, bit 4, 5 and 6 will be merged into
;      the SYS13 (1000,1111b) overlay request
;
EFLAG$      DB    0           ;Flag E
FEMSK$      DB    0           ;Port FE mask
      DS    2%0         ;Flags G-H
;
;      IFLAG$ - Assignments: (INTERNATIONAL)
;       0 - FRENCH
;       1 - GERMAN
;       2 - SWISS
;       3 - reserved for future languages
;       4 - reserved for future languages
;       5 - reserved for future languages
;       6 - Special DMP mode ON/OFF
;       7 - '7' bit mode ON/OFF
;
IFLAG$      EQU   $
      IF    @FRENCH
      DB    01000001B
      ENDIF
      IF    @GERMAN
      DB    01000010B
      ENDIF
      IF    @USA
      DB    0
      ENDIF
      DB    0           ;Flag J
;
;      KFLAG$ assignments:
;       0 - BREAK latch
;       1 - PAUSE latch
;       2 - ENTER latch
;       3 - reserved
;       4 - reserved
;       5 - CAPS lock
;       6 - reserved
;       7 - character in TYPE ahead
;
KFLAG$      DB    0           ;Keyboard flag
;
;      LFLAG$ assignments:
;       0 - inhibit step rate question in FORMAT
;       4 - inhibit 8" query in FLOPPY/DCT
;       5 - inhibit # sides question in FORMAT
;       6,7 - Reserved for IM 2 hardware
;
LFLAG$      DB    00110001B   ;LDOS feature inhibit
```

```
;
;      MODOUT$ mask assignments
;       0 - undefined
;       1 - cassette motor on/off
;       2 - mode select (0 = 80/64, 1 = 40/32)
;       3 - enable alternate character set
;       4 - enable external I/O
;       5 - video wait states (0 = disable, 1 = enable)
;       6 - clock speed (1 = 4 Mhz, 0 = 2 Mhz)
;       7 - undefined
;
      IF    @INTL
MODOUT$     DB    01110000B   ;MODOUT international
      ELSE
MODOUT$     DB    01111000B   ;MODOUT port image (FAST)
      ENDIF
;
;
;      NFLAG$ - Network flag$
;       0 - Allow setting of file open bit in DIR
;       1 / 5 - Reserved
;       6 - Set if in Task Processor
;       7 - Reserved
;
      DB    0           ;Inhibit open bit in DIR
;
;      OPREG$ memory management image port
;       0 - SEL0 - Select map overlay bit 0
;       1 - SEL1 - Select map overlay bit 1
;       2 - 80/64 - 1 = 80 x 24
;       3 - Inverse video
;       4 - MBIT0 - memory map bit 0
;       5 - MBIT1 - memory map bit 1
;       6 - FXUPMEM - fix upper memory
;       7 - PAGE - page 1K video RAM (set for 80x24)
;
OPREG$      DB    10000111B   ;Memory management image
;
;      PFLAG$ - Printer flag
;      7 = Printer spooler is paused
;      0 - 6 = Reserved
;
      DB    0
      DB    0           ;QFLAG$
;
;      RFLAG$ - Retry init for FDC driver
;
RFLAG$      DB    08          ;FDC retry count >=2
;
;      SFLAG$ assignments:
;       0 - inhibit file open bit
;       1 - set to 1 if bit-2 set & EXEC file opened
;       2 - set by @RUN to permit load of EXEC file
;       3 - SYSTEM (FAST)
;       4 - BREAK key disabled
;       5 - JCL active
;       6 - force extended error messages
;       7 - DEBUG to be turned on after LOAD
;
SFLAG$      DB    00001000B   ;System flag (FAST)
;
;
;     Machine TYPE assignment:
;     All values are in decimal
;
;      2 = TRS-80 Model 2
;      4 = TRS-80 Model 4
;      5 = TRS-80 Model 4P
;     12 = TRS-80 Model 12
;     16 = TRS-80 Model 16
;
      IF    @MOD4
TFLAG$      DB    04          ;Model 4 assignment
      ELSE
      ADISP 'ERROR: Undefined machine TYPE for TFLAG'
      ENDIF
      DB    0           ;Flag U
;
;     Video FLAG$ assignments:
;      0-3 - Set blink rate (1=fastest,7=slowest)
;      4 - display CLOCK
;      5 - cursor blink toggle bit
;      6 - Inhibit blinking cursor (user)
;      7 - Inhibit blinking cursor (system)
;
VFLAG$      DB    0           ;Blink,Slow,No clock
;
;     WRINT$ - interrupt mask register
;      0 - enable 1500 baud rising edge
;      1 - enable 1500 baud falling edge
;      2 - enable Real Time Clock INT
;      3 - enable I/O bus interrupts
;      4 - enable RS-232 transmit interrupts
;      5 - enable RS-232 receive data interrupts
;      6 - enable RS-232 error interrupt
;
WRINT$      DB    00000100B   ;WRINTMASK port image
      DS    3%0         ;Flags X,Y and Z
;
;     Contents are high-order byte of SVC table
;
      DB    SVCTAB$>8   ;MSB of SVC table
;
;     OSVER$ stores the operating system version
;
OSVER$      DB    62H         ;OS version #
;
;     Vector for config initialization
;
@ICNFG      RET               ;Initialization config
      DW    0
;
;     Chain vector for KI task processor
;
```

```
@KITSK      RET               ;Keyboard task routine
      DW    0
;
;     System File Control Block for overlays
;
SFCB$ DB    80H,0,0           ;System /SYS FCB
      DW    SBUFF$
      DB    0
      DW    0,0,0,-1,0,-1,-1
;
;     32-byte DEBUG save area
;
DBGSV$      DS    32
;
;     Job Control Language file control block
;
JFCB$ DS    3%0
      DW    SBUFF$
      DS    27
;
;     System Command Line file control block
;
CFCB$ EQU   $           ;Command Interpreter FCB
CFGFCB$     DB    'CONFIG/SYS.CCC:0',3
      DS    15
;
;     Page 1 - System Supervisor Call Table
;
SVCTAB$     EQU   $
      IF    $.NEQ.100H
      ADISP 'ERROR: SVCTBL location violation'
      ENDIF
;
;     Initial version
;
MAXCOR$     EQU   2400H+START$
MINCOR$     EQU   3000H+START$
      ORG   @BYTEIO
;
;     File positioning routines - MUST BE FIRST
;
      ADISP '<File positioning subroutines>'
;     ?
*GET  'FILPOSN:1'
;     PAGE
CORE$ DEFL  $
      ORG   CRTBGN$+13
      DB    'LS-DOS 06.02.00'
      IF    @USA
      DB    ' '
      ENDIF
      IF    @GERMAN
      DB    'D'
      ENDIF
      IF    @FRENCH
      DB    'F'
      ENDIF
      DB    '- Copyright 1984 '
      DB    'Logical Systems Inc.'
      ORG   CRTBGN$+80+14
      DB    'All Rights Reserved. '
      DB    'Licensed to                   '
      ORG   CORE$
;
;     Get the System Loader
;
      ADISP '<System Loader and associated routines>'
;     ?
*GET  'LOADER:1'
      ADISP '<System front end & task processor>'
;     ?
*GET  'TASKER:1'
      IF    $.GT.1D00H+START$
      ADISP 'ERROR: SYSRES memory overflow'
      ENDIF
CORE$ DEFL  $
      DS    1D00H-CORE$%0
      ORG   CORE$
      ORG   1D00H+START$
SBUFF$      EQU   $
      DS    256         ;Page disk I/O buffer
DIRBUF$     EQU   MAXCOR$-256 ;Another file buffer
;
;     Get the system initialization module
;
OVERLAY     EQU   $
      ADISP '<System initialization routines>'
;     ?
*GET  'SYSINIT4:1'
      ADISP '<Misc. lowcore routines>'
;     ?
*GET  'SOUND:1'
      ADISP '<Sign-on LOGO display>'
*GET  'LOGO:1'
;
      END   OVERLAY
□
```

COPYCOM/ASM

```
; COPYCOM - File for Copyright COMment block
;
      COM   '<*(C) 1982,83,84 by LSI*>'
;
      END
```

FILPOSN/ASM

```
;FILPOSN/ASM - LS-DOS 6.2
;
;      Entry for byte I/O from @GET & @PUT
;
BYTEIO       PUSH  IX
      POP   DE           ;Transfer DCB to DE
      CALL  CKOPEN@            ;Ck file open, save regs
      SET   7,(IX+1)    ;Denote byte or LRec
      LD    A,B          ;Get type code & test
      CP    2            ;  for get/put
      LD    A,C
      JR    Z,WRCHAR    ;Go on PUT
      JR    NC,IORETZ   ;Ignore if CTL
;
;      Get a byte from a file
;
RDCHAR       CALL  CKEOF1             ;Ck for end of file
      RET   NZ           ;Return if at end
      BIT   5,(IX+1)    ;If buffer not current,
      CALL  NZ,NSEC1    ;  read next sector
      RET   NZ
      CALL  BFRPOS             ;Pt to byte posn in BFR
      LD    A,(DE)             ;P/u the byte
      INC   (IX+5)             ;Inc NEXT ptr
      CALL  Z,SET5             ;Set bit 5 if zero
      CP    A            ;Set Z flag--no error
      RET
;
SET5  SET   5,(IX+1)
      RET
;
;      Write a byte to a file
;
WRCHAR       BIT   6,(IX+0)    ;Prot level is write access?
      JP    Z,RWRIT3    ;Go if not
      PUSH  AF           ;Save byte
      BIT   5,(IX+1)    ;Get next sector if
      CALL  NZ,WRCH2    ;  buffer is not current
      JR    Z,WRCH1            ;Skip if read was ok
      EX    (SP),HL            ;Pop stack but keep
      POP   HL           ;  error # in AF
      RET
;
WRCH1 CALL  BFRPOS             ;Next BFR byte posn
      POP   AF
      LD    (DE),A             ;Stuff the byte
      SET   4,(IX+1)    ;Buffer contains updated data
      INC   (IX+5)             ;Incr NEXT byte
      PUSH  AF           ;Save Z or NZ flag
      CALL  Z,SET5             ;Set bit 5 if offset 0
      CALL  CKEOF1             ;Check for EOF
      JR    NZ,ATEOFW   ;Go if there
      BIT   6,(IX+1)    ;Jump if EOF set to next
      JR    NZ,DNTSET   ;  only if at EOF
```

```
ATEOFW      LD    (IX+8),C    ;Set End Of File
      LD    (IX+12),L
      LD    (IX+13),H
DNTSET      POP   AF          ;Restore offset flag
      JR    Z,RWRIT1    ;Go to write sector if 00
IORETZ      XOR   A           ;Set Z flag--no error
      RET
;
;     WRCHR needs the next sector - if UPDATE, ck EOF
;
WRCH2 LD    A,(IX+1)    ;CK if UPD bit set
      AND   7           ;Mask for prot level
      CP    4           ;Check for UPD
      JR    NZ,NSEC1    ;Bypass EOF ck on > UPD
NXTSECT     CALL  CKEOF1            ;Ck for end of file
      RET   NZ          ;Can't extend in update mode
NSEC1 LD    A,(IX+1)    ;Read access?
      AND   7
      CP    6
      JR    NC,RWRIT3   ;"Illegal Acces..." if not
NSEC2 CALL  IOREC       ;Calc cylinder/sector
      RET   NZ
      RES   5,(IX+1)    ;Show buffer current
      LD    L,(IX+3)    ;P/u buffer address
      LD    H,(IX+4)
      CALL  @RDSEC            ;Read the sector
      JR    Z,BUMPNRN   ;Go if no error
      CP    6           ;Test for prot sector
      RET   NZ          ;Quit if error not 6
BUMPNRN     INC   (IX+10)           ;Incr the NRN ptr LSB
      JR    NZ,ZEROA@
      INC   (IX+11)           ;  and MSB if necessary
ZEROA@      XOR   A
      RET
;
;     Repositioning needs to write out the buffer
;
RWRIT@      LD    A,(IX+1)
      AND   90H         ;Test for non-sector I/O and
      CP    90H         ;  buffer contents changed
      JR    Z,RWRIT1    ;Go if conditions true
      JR    ZEROA@            ;  else no need to write
@RWRIT      CALL  CKOPEN@           ;Ck file open, save regs
RWRIT1      CALL  GETNRN            ;P/u Next Record Number
      LD    A,H         ;Ignore if rewound
      OR    L
      RET   Z
      DEC   HL          ;Dec & reset NRN
      LD    (IX+10),L
      LD    (IX+11),H
;
;     Check access protection level
;
RWRIT2      LD    A,(IX+1)    ;Get prot lvl
      AND   7
      CP    5           ;UPDATE access or better?
      JR    C,RWRIT4
```

```
RWRIT3      LD    A,25H       ;Illegal Access error code
      OR    A           ;Return NZ
      RET
;
RWRIT4      AND   4           ;If UPDATE access, then
      JR    Z,RWRIT5    ;  can't extend if at EOF
      CALL  CKEOF1
      JR    NZ,RWRIT3   ;  so show "Illegal Acces...
RWRIT5      CALL  IOREC       ;Calculate cylinder & sector
      RET   NZ
      LD    L,(IX+3)    ;P/u buffer addr
      LD    H,(IX+4)
      RES   4,(IX+1)    ;Altered buffer flag off
      SET   2,(IX+0)    ;Show modification done
      CALL  @WRSEC            ;  for directory MOD flag
      RET   NZ
VEROP LD    A,0         ;Verify operation if set
      OR    A
      CALL  NZ,@VRSEC   ;Verify if no write error
      RET   NZ          ;Return if wrt/ver error
      CALL  BUMPNRN           ;Increment NRN
;
;     Check if ERN to be set to NRN
;     Should be done for byte I/O, but not random I/O
;
      CALL  CKEOF1            ;Returns 0 if not at EOF
      DEC   A           ;Set bit 6 if retcod=0
      AND   (IX+1)            ;If IX+1, bit 6 set, then
      AND   40H         ;  don't update EOF unless at
      JR    NZ,ZEROA@   ;  or past the old EOF
YESEOF      LD    (IX+12),L   ;Update ERN
      LD    (IX+13),H
      BIT   3,(IX+1)    ;Test if ending '!'
      JP    NZ,WEOF1    ;Update direc if so
      RET
;
GETNRN      LD    L,(IX+10)   ;Xfer NRN to HL
      LD    H,(IX+11)
      RET
;
BFRPOS      LD    A,(IX+5)    ;P/u byte offset in buffer
      ADD   A,(IX+3)    ;Add to buffer LSB
      LD    E,A
      LD    A,(IX+4)    ;  and adjust buffer MSB
      ADC   A,0         ;  if needed
      LD    D,A         ;Return DE = posn
      RET
;
;     Entry to seek next record of a file
;
@SEEKSC     CALL  CKOPEN@           ;Link to FCB & ck if open
      CALL  CKEOF1            ;Ensure not > EOF
      CALL  Z,IOREC           ;Get track/sector data
      RET   NZ          ;Back on I/O error
      CALL  @SEEK       ;Issue seek to drive
      XOR   A           ;Ignore seek errors here
      RET
```

```
;
;     Entry to Skip record routine
;
@SKIP CALL  @LOC        ;Locate next record
      INC   BC          ;Step past it
;
;     Entry to Position to record routine
;
@POSN CALL  CKOPEN@
      SET   6,(IX+1)    ;Upd EOF only if NRN>EOF
      BIT   7,(IX+1)    ;Jump if sector I/O only
      JR    Z,POSN1
      LD    H,B         ;Record ptr to HL
      LD    L,C
      OR    (IX+9)            ;P/u LRL
      JR    Z,POSN1           ;Skip nxt if LRL=256
      CALL  @MUL16            ;Calc sector & offset
      LD    B,H         ;Physical sector =>BC
      LD    C,L
      LD    (IX+5),A    ;Set byte ptr
      BIT   5,(IX+1)    ;Jump if buffer does not
      JR    NZ,POSN2    ;  contain current sector
      CALL  GETNRN            ;P/u the NRN
      SCF
      SBC   HL,BC       ;Subtract with Cy
      JR    Z,$CKEOF    ;Pass on to CKEOF
POSN1 LD    (IX+5),A    ;Offset in buffer
POSN2 PUSH  BC
POSN2A      CALL  RWRIT@            ;Write current if needed
      POP   BC          ;  before moving
      RET   NZ          ;Back on write error
      LD    (IX+10),C   ;NRN
      LD    (IX+11),B
      CALL  SET5        ;Show bufr does not
$CKEOF      JP    CKEOF1            ;  contain current sector
;
;     Entry to force a physical read
;
@RREAD      CALL  CKOPEN@
      LD    C,1         ;Cause ADJUST to bump
                        ;  NRN when called
BKSP1 CALL  GETNRN            ;Get current record #
      LD    A,H         ;If file is rewound,
      OR    L           ;  then ignore the req
      JR    Z,BKSP0           ;  & force OFFSET = 0
      DEC   HL          ;Back up by 1
      CALL  ADJ2        ;RET if sector I/O only,
                        ;  else bump fwd if RREAD
                        ;  then back up if bit 5=0
      PUSH  HL          ;Will be popped into BC
      JR    POSN2A            ;Finish the job
;
;     Entry to backspace one logical record
;
@BKSP CALL  CKOPEN@
      LD    C,A         ;Keep ADJUST from bumping
      LD    B,(IX+9)    ;P/u LRL
      OR    B           ;Is it a 0?
      JR    Z,BKSP1           ;Go if so
      LD    A,(IX+5)    ;P/u next byte pointer
      SUB   B           ;Subtr one record length
BKSP0 LD    (IX+5),A
      JR    C,BKSP1           ;Go if X'd sector boundary
      XOR   A           ;  else all done
      RET
;
;     Entry to Rewind to beginning
;
@REW  CALL  CKOPEN@
      LD    B,A         ;Zero NRN
      LD    C,A
      JR    POSN1       ;Will also zero offset
;
;     Entry to Position to end-of-file
;
@PEOF CALL  CKOPEN@
      LD    C,(IX+12)   ;ERN to BC
      LD    B,(IX+13)
      OR    (IX+8)            ;P/u EOF byte
      JR    Z,POSN1           ;Go if full sector
      DEC   BC          ;Point to last record
      JR    POSN1       ;Use POSN to get end
;
;     Entry to Locate current record number
;
@LOC  CALL  CKOPEN@
      CALL  GETNRN            ;P/u NRN
      CALL  ADJUST            ;Get offset and adj NRN
LOC1  LD    E,(IX+9)    ;P/u LRL
      LD    A,E         ;Test LRL for zero
      OR    A           ;If zero, then give NRN
      JR    Z,LOC3            ;LRL=0, NRN is correct
      INC   C           ;If offset is zero,
      DEC   C           ;  then it's at 256,
      JR    Z,LOC2            ;  and we don't dec NRN
      DEC   HL
;
;     Divide the three-byte pointer (HLC) by the LRL
;
LOC2  CALL  @DIV16            ;Divide (NRN-1)/LRL
      LD    B,L         ;Save high-order result
      LD    D,H         ;Save possible overflow
      LD    H,A         ;Prepare 2nd dividend
      LD    L,C         ;P/u low order dividend
      LD    A,E         ;P/u LRL divisor again
      CALL  @DIV16
      LD    H,B         ;Xfer high order result
      OR    A           ;If remainder, we have a
      JR    Z,$+3       ;  partial record to round
      INC   HL          ;  up to next record #
      LD    A,D         ;Xfer possible overflow
LOC3  POP   BC          ;Pop RESTREG return addr
      EX    (SP),HL           ;Exchange value with BC
      PUSH  BC          ;Restore RESTREG
```

```
;
      IF    @MOD4
ORARET@     EQU   $
      ENDIF
      OR    A
      RET
;
;     Entry to Locate the End-Of-File record
;
@LOF  CALL  CKOPEN@
      LD    L,(IX+12)   ;P/u ERN
      LD    H,(IX+13)
      LD    C,(IX+8)    ;EOF byte
      JR    LOC1        ;Handle all LRLs
;
;     Entry to Write an End-Of-File mark
;
@WEOF CALL  CKOPEN@
      CALL  RWRIT@            ;Write buffer if needed
WEOF1 LD    B,(IX+7)    ;P/u DEC of FPDE
      LD    C,(IX+6)    ;P/u drive #
      CALL  @DIRRD            ;Read file's dir record
      RET   NZ          ;Back if read error
      INC   L           ;Pt to ERN offset (DIR+3)
      INC   L
      INC   L
      LD    A,(IX+8)    ;P/u EOF offset
      LD    (HL),A            ;Put in directory
      LD    DE,17       ;Pt to EOF in dir
      ADD   HL,DE
      LD    A,(IX+12)   ;P/u EOF low order byte
      LD    (HL),A            ;Put EOF in DIREC
      INC   HL
      LD    A,(IX+13)   ;P/u EOF high order byte
      LD    (HL),A
      JP    @DIRWR            ;Write dir record and return
;
;     Entry to Read a Record
;
@READ CALL  CKOPEN@
      PUSH  HL
      CALL  RWRIT@            ;Write buffer if needed
      POP   HL
      RET   NZ          ;Back on write error
      LD    B,(IX+9)    ;P/u LRL
      LD    A,B         ;If LRL=256, simply
      OR    A
      JP    Z,NXTSECT   ;  get the next sector
RDREC PUSH  HL          ;Save buffer posn
      PUSH  BC          ;Save LRL
      CALL  RDCHAR            ;Read next byte
      POP   BC
      POP   HL
      RET   NZ          ;Back on read error
      LD    (HL),A            ;Put char into buffer
      INC   HL          ;Bump buffer ptr
      DJNZ  RDREC       ;Loop for entire record
      RET
;
;     Entry to Write a Record
;
@WRITE      CALL  CKOPEN@
WRIT1 LD    (VEROP+1),A ;Turn on/off verify
      LD    B,(IX+9)    ;P/u LRL
      LD    A,B         ;Bypass if LRL=256
      OR    A
      JP    Z,RWRIT2
      PUSH  HL          ;Save some FCB values
      LD    H,(IX+5)    ;P/u buffer offset locn
      LD    L,(IX+8)    ;P/u EOF offset byte
      EX    (SP),HL           ;Put values on stack
                        ;  and recover HL
WRREC LD    A,(HL)            ;Pass the logical record
      INC   HL          ;  to the writing routine
      PUSH  HL          ;  byte by byte
      PUSH  BC
      CALL  WRCHAR
      POP   BC
      POP   HL
      JR    NZ,WRERROR  ;Exit and fix FCB
      DJNZ  WRREC       ;Loop for entire record
      EX    (SP),HL           ;Remove stored FCB info
      POP   HL          ;Recover HL
      RET
WRERROR     EX    (SP),HL           ;Get FCB values
      LD    (IX+5),H    ;  and put them back
      LD    (IX+8),L
      POP   HL          ;Restore HL
      RET               ;Go back with error
;
;     Entry to Verify after write of a record
;
@VER  CALL  CKOPEN@
      INC   A           ;Set verify byte
      JR    WRIT1
LNKFCB@     SCF               ;Init to force file open
      DB    0D2H        ;  test by JP NC,aaaa
CKOPEN@     LD    A,(DE)            ;Ignore if from LNKFCB
      RLCA              ;Test high bit of FCB
      EX    (SP),HL
      LD    (JRET$),HL  ;Save ret
      LD    (JDCB$),DE  ;Save DCB
      EX    (SP),HL
      JR    NC,NOTOPEN  ;Go if not an open FCB
      POP   AF          ;Get return
      PUSH  DE          ;DCB addr to IX
      EX    (SP),IX
      PUSH  HL          ;Save regs
      PUSH  DE
      PUSH  BC
      PUSH  HL          ;Establish Return addr
      LD    HL,RESTREG  ;  to restore registers
      EX    (SP),HL
      PUSH  AF          ;Put back ret
      XOR   A
      RET               ;Go back
;
NOTOPEN     POP   AF
      LD    A,26H       ;Set error "File Not Open
      OR    A           ;Set NZ condition
      RET
;
RESTREG     POP   BC          ;Pop back registers save
      POP   DE          ;  in CKOPEN@
      POP   HL
      POP   IX
      RET
;
;     Entry to check if at End-Of-File
;
@CKEOF      CALL  CKOPEN@
CKEOF1      CALL  GETNRN            ;P/u NRN into HL
      PUSH  HL          ;Save un-adjusted NRN
      CALL  ADJUST            ;Adjust for special cases
      LD    A,H         ;Compare high byte
      CP    (IX+13)
      JR    NZ,CKEOF2   ;Go if not equal
      LD    A,L         ;Compare low-order byte
      CP    (IX+12)
      JR    NZ,CKEOF2   ;Go if not equal
      DEC   C           ;Adjust for 00=256
      LD    A,(IX+8)    ;Compare offset byte
      DEC   A
      SUB   C           ;Set NC, NZ conditions
      CCF               ;  if past EOF
      INC   BC          ;Restore old BC value
CKEOF2      POP   HL          ;Restore unadjusted NRN
      LD    A,1DH       ;Rec # out of range code
      JR    NZ,CKEOF3   ;Go if not at EOF
      DEC   A           ;X'1C'=EOF encountered
      RET               ;Return with NZ flag
CKEOF3      RET   NC          ;Return with error
      XOR   A           ;  else set Z flag
      RET               ;Ret with no error
;
;     File positioning adjustment routines
;
ADJUST      EQU   $           ;Entry from @CKEOF and @LOC
      LD    C,(IX+5)    ;Pick up offset
ADJ2  EQU   $           ;Entry from @BKSP/@RREAD
      BIT   7,(IX+1)    ;Sector I/O only?
      RET   Z           ;No adjustment if so
      LD    A,C         ;Offset =0? (or "RREAD?")
      OR    A
      JR    Z,$+3       ;Go if zero
      INC   HL          ;Set for next record
      BIT   5,(IX+1)    ;Last byte was read?
      RET   NZ          ;Go if set
      DEC   HL          ;  else re-adjust ptr
      RET
;
```

```
;      Calculate the cylinder/sector of needed record
;
IOREC CALL  GETNRN            ;P/u record number
      CALL  @DCTBYT-5   ;Get # of sectors/gran
      AND   1FH         ;Use only bits 0-4
      INC   A           ;Adjust logical => physical
      CALL  @DIV16            ;By # of sectors/gran
      LD    (CALS5+1),A ;Sv rmndr (sector offset)
      PUSH  IX          ;Xfer FCB to HL
      EX    (SP),HL
      LD    BC,14       ;Pt to 1st extent info
      ADD   HL,BC       ;FCB+14
      POP   BC          ;Pop gran ptr HL into BC
      LD    A,5         ;Init to check 4 extents
      LD    DE,0        ;  & extended FXDE ptr
GREC1 PUSH  AF
      LD    A,(HL)            ;P/u starting cyl byte
      INC   HL          ;  & bypass if FF
      INC   A
      JR    Z,GREC2
      PUSH  HL          ;Xfer the # of grans up
      LD    H,D         ;  to but not including
      LD    L,E         ;  this extent into HL
      XOR   A           ;Subtr gran ptr from
      SBC   HL,BC       ;  cumulative figure & go
      JR    C,GREC3           ;  if not in previous ext
      POP   HL
      JR    Z,CALCSEC
GREC2 INC   HL
      POP   AF
      DEC   A
      JR    Z,GREC4           ;Jump when all quads ckd
      LD    E,(HL)            ;P/u cumulative # grans
      INC   HL          ;  up to but not
      LD    D,(HL)            ;  including this extent
      INC   HL
      JR    GREC1
GREC3 INC   H           ;Within 256 grans?
      LD    A,L         ;Xfer Low-order difference
      POP   HL          ;Rcvr # of contiguous grans
                        ;  in this extent
      JR    NZ,GREC2    ;Go if not within 256
      PUSH  DE          ;Save cumulative count
      LD    E,A         ;Xfer gran dif (neg)
      LD    A,(HL)            ;P/u # of grans
      AND   1FH         ;  in this extent
      ADD   A,E         ;Add to negative difference
      LD    A,E         ;Put negative diff into A
      POP   DE
      JR    NC,GREC2    ;Go if not in this extent
      NEG               ;Is in this extent, make
      JR    CALCSEC           ;  diff positive & use it
;
;      All current quads checked - Need directory info
;
GREC4 EQU   $
      CALL  ALLOC       ;Get # of grans
      RET   NZ           ;  into the extent
      LD    (CALS4+1),A ;  or error RET
      JR    NC,CALS3    ;Jp if record in 1st ext
      JR    CALS1       ;  else jp if in another
;
;     Calc sector in gran
;
CALCSEC     LD    (CALS4+1),A ;Stuff # grans into
      LD    B,(HL)            ;  this extent
      DEC   HL          ;P/u # contig grans &
      LD    C,(HL)            ;  rel start & start cyl
      INC   HL
      POP   AF          ;Rcvr # of quad
      CPL
      ADD   A,4
      JR    NC,CALS2    ;Jump if 1st ext or quad
      INC   A           ;If not 1st, set up to move
      RLCA              ;  matching quad to the
      RLCA              ;  first position by
      PUSH  BC          ;  shuffling the others up
      PUSH  DE
      LD    C,A         ;Get bytes to move
      LD    B,0
      EX    DE,HL       ;DE = top of last quad
      LD    HL,-4
      ADD   HL,DE       ;HL = top of next lower
      LDDR              ;Do the shuffle
      EX    DE,HL
      POP   DE
      POP   BC
CALS1 LD    (HL),B            ;Move info on matching quad
      DEC   HL          ;  into position
      LD    (HL),C
      DEC   HL
      LD    (HL),D
      DEC   HL
      LD    (HL),E
CALS2 LD    H,B         ;Xfer start & contig gran
      LD    L,C         ;Xfer start cylinder
CALS3 LD    A,H
      RLCA              ;P/u start gran on track
      RLCA
      RLCA              ;Was bits 5-7
      AND   7           ;Zero the unwanted
CALS4 ADD   A,0         ;P/u # grans into extent
      CALL  RELCYL            ;Calc 1st relative cyl
      ADD   A,L         ;Add starting cyl
      LD    D,A
      LD    A,B         ;Recover # Sectors/gran
      AND   1FH         ;  use bits 0-4
      INC   A           ;  logical => physical
      PUSH  DE          ;Calculate sector offset
      CALL  @MUL8       ;  into desired cylinder
      POP   DE          ;  for desired granule
CALS5 ADD   A,0         ;P/u # of excess sectors
      LD    E,A         ;  over even gran & add
      XOR   A           ;  to granule sector
      RET
;
;     On entry, gran needed is in BC
;
ALLOC CALL  CYL_GRN            ;Find ext cnting gran
      RET   NZ          ;Ret on error
      PUSH  HL          ;Save starting cyl & gran
      LD    H,B         ;Xfer granule needed to
      LD    L,C         ;  HL then calculate how
      XOR   A           ;  many grans into this
      SBC   HL,DE       ;  extent is the desired
      LD    A,L         ;  granule
      LD    (ALL6+1),A  ;Stuff rel gran from
      POP   HL          ;  start of extent
      PUSH  DE          ;Save granule count
      PUSH  IX          ;  to extent
      EX    (SP),HL            ;FCB pointer to HL
      LD    DE,14       ;Pt to 1st alloc in FCB
      ADD   HL,DE
      POP   DE          ;Pop starting cylinder
      LD    B,5         ;  to this extent
ALL1  LD    A,(HL)             ;P/u a cylinder
      INC   HL          ;Does starting cyl of
      CP    E           ;  needed gran alloc
      JR    NZ,ALL2            ;  appear in this extent?
      LD    A,(HL)             ;Now see if needed gran is
      XOR   D           ;  in this extent field
      AND   0E0H        ;  by checking its starting gran
      JR    Z,ALL4
ALL2  DEC   B           ;Decr the count-dwn loop
      JR    Z,ALL3             ;Done if no match
      INC   HL          ;Go to next extent
      INC   HL          ;  info in FCB
      INC   HL
      JR    ALL1
ALL3  PUSH  DE          ;Save needed extent info
      EX    DE,HL       ;Set up to shuffle extent
      LD    HL,-4       ;  info
      ADD   HL,DE
      LD    BC,12
      LDDR
      EX    DE,HL
      POP   BC
      XOR   A           ;Set Z no error
      SCF               ;Set C flag, extent not found
      JR    ALL5
ALL4  LD    (HL),D
      EX    DE,HL
      XOR   A           ;Set Z no error
ALL5  POP   DE
ALL6  LD    A,0         ;# of grans into this ext
      RET               ;Wher desired gran is
;
;     Extent is unused - need to allocate more space
;
CG06  CALL  CG07        ;Try to allocate more
      POP   BC          ;Get back desired gran
      RET   NZ          ;Return on error
                        ;Look again for gran
;
;     Find extent containing desired granule
;
CYL_GRN     PUSH  BC          ;Save desired gran #
      LD    DE,0        ;Init gran counter
      LD    B,(IX+7)    ;P/u DEC of file
CG01  LD    A,B
      LD    (STUFDEC+1),A     ;Stuf it
      LD    C,(IX+6)    ;P/u drive for file
      CALL  @DIRRD            ;Read its directory
      LD    BC,22       ;Point to 1st extent
      ADD   HL,BC       ;  of its directory
      EX    DE,HL       ;Gran count to HL
      POP   BC          ;Restore desired gran
      RET   NZ          ;Return on read error
CG02  LD    A,(DE)            ;Is this extent
      CP    0FEH        ;  allocated?
      JR    NC,CG05           ;Jump if it is not
      INC   DE          ;Point to allocation
      LD    A,(DE)            ;P/u relative gran & #
      PUSH  HL          ;  of contiguous grans
      AND   1FH         ;Keep contiguous grans
      INC   A           ;  & bump for 0 offset
      ADD   A,L         ;Add to count in HL
      LD    L,A
      JR    NC,CG03
      INC   H           ;Bump high order
CG03  PUSH  HL          ;Save gran count to
      DEC   HL          ;  end of extent
      XOR   A           ;Test if EOF if in this
      SBC   HL,BC       ;  allocation
      POP   HL
      JR    NC,CG04           ;EOF not > this alloc
      INC   DE          ;Get rid of old
      POP   AF          ;  current quantity
      JR    CG02        ;Check next extent
;
;     The EOF is within this allocation, Recover
;     the allocation data and exit
;
CG04  POP   HL          ;P/u gran count to extent
      EX    DE,HL       ;Gran count to DE
      LD    A,(HL)            ;P/u granule data
      DEC   HL
      LD    L,(HL)            ;P/u starting cylinder
      LD    H,A
      XOR   A
      RET
;
;     This extent is 1) unused, or 2) FXDE pointer
;     and the needed gran has not been found yet
;
CG05  PUSH  BC          ;Gran count to DE &
      EX    DE,HL       ;DIR ptr to HL
      JR    NZ,CG06           ;Jump if unused
      INC   HL          ;Point to DEC of FXDE
      LD    B,(HL)            ;P/u the DEC
      JR    CG01        ;  & loop
;
;     See if the drive has enough free space left
;
CG07  PUSH  BC          ;Save needed gran
      LD    C,(IX+6)    ;P/u file's drive
      CALL  @GATRD            ;Get GAT
      POP   BC          ;Recover needed gran
      RET   NZ          ;Return if GAT error
      PUSH  HL
      LD    H,B         ;Xfer the requested
      LD    L,C         ;  gran to HL &
      XOR   A           ;  subtract current gran
      SBC   HL,DE       ;Count to calculate how
      LD    B,H         ;  many excess grans
      LD    C,L         ;  are needed
      INC   BC
      POP   DE          ;Recover dir byte ptr
      INC   DE          ;Pt to next DIR byte
      LD    H,DIRBUF$>8 ;Start looking at TRK #1
      LD    A,(AFLAG$)  ;P/u Search start CYL
      LD    L,A         ;  and put it in L
      PUSH  BC          ;Save excess grans needed
      LD    A,E         ;Is this extent the 1st?
      AND   1EH         ;Jump if so, else we can
      CP    16H         ;  use it for allocation
      JR    Z,CG14
      DEC   E           ;Back up to previous
      DEC   E           ;  extent
CG12  LD    A,(DE)            ;P/u # of contig grans
      AND   1FH         ;  see if the last gran
      INC   A           ;  used can be extended
      LD    C,A         ;Is current # the max
      CP    20H         ;  an extent can hold?
      JR    Z,CG13            ;Jump if a full extent
      LD    A,(DE)            ;  (32 grans max) - else
      AND   0E0H        ;  p/u the relative
      RLCA              ;  granule offset
      RLCA
      RLCA
      ADD   A,C         ;Add the # of contiguous
      PUSH  DE          ;  granules
      CALL  RELCYL            ;Calc relative cyl needed
      LD    B,A         ;Save offset
      LD    C,E
      POP   DE
      DEC   DE          ;Backup to starting cyl
      LD    A,(DE)
      INC   DE          ;  & repoint to alloc byte
      ADD   A,B         ;Add cyls used to
      LD    L,A         ;  starting cyl
      LD    H,DIRBUF$>8 ;Is it less than max?
      CP    0CBH
      JR    NC,CG13           ;Jump if too big
      LD    A,C
      LD    B,(HL)            ;P/u the cyl's GAT
      CALL  TSTBIT            ;Test if gran is free
      JR    Z,CG21            ;Bypass if free gran
;
;     The next gran cannot be used - get another extent
;
CG13  INC   E           ;Else point to next
      INC   E           ;  extent field
      LD    A,E
      AND   1EH         ;Jump if not on the FXDE
      CP    1EH         ;  field, else we have to
      JR    NZ,CG14           ;  obtain an FXDE record
;
;     Last extent used up, get new dir rec for FXDE
;
      CALL  CG23        ;Write current GAT & HIT
      POP   BC
      RET   NZ          ;Ret if GAT/HIT error
      PUSH  BC
      CALL  NEWHIT            ;Get new HIT for FXDE
      POP   BC
      RET   NZ          ;Loop to process
      JP    CYL_GRN           ;  new extent
;
;     Extent is vacant - use it & get new allocation
;
CG14  CALL  MAXCYL            ;Get highest # cyl
      LD    (CG17+1),A  ;Stuff highest cyl
      LD    B,2
CG16  LD    A,L         ;Test last cyl used
CG17  CP    0           ;P/u max cyl
      JR    NC,CG18
      LD    A,(HL)            ;P/u a GAT byte
      INC   A
      JR    NZ,CG19           ;Go if space in this cyl
      INC   L           ;  else bump to next one
      JR    CG16        ;  & loop
CG18  LD    L,0         ;Now start from begin
      DJNZ  CG16        ;  of disk & recheck
      POP   BC
      CALL  CG23        ;Write out GAT & HIT
      RET   NZ
      LD    A,1BH       ;"disk space full"
      OR    A           ;Set error NZ
      RET
;
;     Found available space in cylinder
;
CG19  LD    A,0FFH            ;Set DIR extent to FF
      LD    (DE),A
      LD    C,0
      LD    B,(HL)            ;P/u current GAT alloc
CG20  LD    A,C
      CALL  TSTBIT            ;Find a free gran
      JR    Z,CG21            ;  & jump when found
      LD    A,(DE)            ;  else advance starting
      ADD   A,20H       ;  relative gran value
      LD    (DE),A
      INC   C           ;Bump pointer to test
      JR    CG20        ;  next gran
;
;     Next gran in line is free - allocate it
;
CG21  LD    A,C
      CALL  SETBIT            ;Show it allocated
      OR    (HL)
      LD    (HL),A
      DEC   E           ;Bump to starting cyl
      LD    A,(DE)            ;Bump by one to see if
      INC   A           ;  this alloc is the 1st
      JR    NZ,CG22           ;  one for the extent &
      LD    A,L         ;  we have to set the
                        ;  starting cylinder
      LD    (DE),A            ;Stuff starting cyl
CG22  INC   E
      LD    A,(DE)            ;Add 1 to # of contiguous
      INC   A           ;  granules
      LD    (DE),A
      POP   BC          ;Decrement needed gran
      DEC   BC          ;  count since we just
      PUSH  BC          ;  allocated one
      LD    A,B         ;Loop if we need more
      OR    C           ;  space allocated
      JP    NZ,CG12
      POP   BC
CG23  LD    C,(IX+6)    ;Else p/u the drive #
      CALL  @GATWR            ;  & write out the GAT
      RET   NZ
STUFDEC     LD    B,0         ;P/u DEC of FPDE
      JR    @DIRWR
;
;     Get new HIT for FXDE
;
NEWHIT      LD    C,(IX+6)    ;P/u drive #
      CALL  @HITRD            ;Read the HIT
      RET   NZ
      LD    A,(IX+7)    ;P/u FPDE DEC so 1st ck
      AND   1FH         ;  will be for next
      CALL  NHIT4       ;  in line
      LD    A,1EH       ;Init "Full directory...
      RET   NZ          ;Ret if no space
      LD    B,L         ;Set DEC for
      LD    A,L         ;  directory read
      LD    (NHIT3+1),A ;Stuff new DEC from HIT
      LD    D,H
      LD    E,(IX+7)    ;P/u current DEC
      LD    A,(DE)            ;Copy filespec HASH CODE
      LD    (HL),A            ;  to new DEC
      CALL  @HITWR
      CALL  Z,@DIRRD
      RET   NZ
      LD    (HL),90H    ;Show dir rec in use as
      INC   L           ;  FXDE record
      PUSH  BC          ;P/u DEC of FPDE &
      LD    A,(STUFDEC+1)     ;  stuff it into FXDE's
      LD    (HL),A            ;  DIR+1 to link back
      INC   L
      LD    B,20        ;Zero out 20 bytes
NHIT1 LD    (HL),0            ;  in the FXDE
      INC   L
      DJNZ  NHIT1
      PUSH  HL          ;Save ptr to 1st extent
      LD    B,10        ;Init to X'FF' 10 bytes
NHIT2 LD    (HL),0FFH   ;  or 5 extents
      INC   L
      DJNZ  NHIT2
      POP   DE          ;Recover ptr to 1st ext
      INC   DE          ;Pt to allocation byte
      POP   BC
      CALL  @DIRWR            ;Write FXDE back to disk
      RET   NZ          ;Return if error
      LD    A,(STUFDEC+1)     ;  else p/u DEC of FPDE
      LD    B,A
      CALL  @DIRRD            ;Read its directory
      RET   NZ          ;  & return if error
      LD    A,L
      ADD   A,1EH       ;Point to FXDE postn
      LD    L,A         ;  in FPDE
      LD    (HL),0FEH   ;Show link to FXDE
      INC   L
NHIT3 LD    (HL),0            ;Show what the FXDE DEC is
                        ;  & write the DIR back
;
;     Routine to write a directory sector
;     B => DEC of FPDE, C => logical drive number
;     HL <= will point to directory record in SBUFF$
;
@DIRWR      CALL  DIRWR       ;Permit two attempts
      RET   Z
DIRWR PUSH  DE          ;Save the regiment
      CALL  CALCDIR           ;Calc dir cyl
      LD    L,0         ;Set buffer to start
      CALL  @WRSSC            ;Write the sector
      CALL  Z,@VRSEC    ;Verify on no error
      SUB   6
      POP   DE
      RET   Z           ;Back on system sector
      CP    0FH-6       ;Write-Protected Error?
      LD    A,18        ;Set dir write error
      RET   NZ          ;  if not WP'd
      SUB   3
      RET
;
;     Find a spare Hash Index Table entry
;
NHIT4 PUSH  AF
      LD    A,7         ;Get highest # sector
      CALL  @DCTBYT           ;  on a cylinder
      PUSH  DE          ;  into register E
      LD    D,A         ;Save for Calc HEADS
      AND   1FH
      LD    E,A
      INC   E           ;& get number of HEADS
      XOR   D           ;  into register A
      RLCA
      RLCA
      RLCA              ;Bits 5-7 => 0-2
      INC   A           ;Logical => Physical
      CALL  @MUL8       ;To calc sectors/cylinder
      CALL  CKDBLBIT    ;Double if necessary
      POP   DE          ;Total sectors per cyl
      SUB   2           ;Reduce for GAT & HIT
      LD    (NHIT7+1),A ;# of directory sectors
      POP   AF          ;Get DEC init entry
      LD    L,A
      CALL  NHIT6       ;Ck if HIT slot is spare
      RET   Z           ;Return if it is spare
      LD    L,3FH
NHIT5 INC   L
NHIT6 LD    A,L
      AND   1FH
NHIT7 CP    0           ;Does value exceed
      JR    NC,NHIT9    ;  sectors/cylinder?
      LD    A,(HL)
      OR    A
      RET   Z
NHIT8 LD    A,L
      ADD   A,20H
      LD    L,A
      JR    NC,NHIT6
      CP    1FH         ;Else go to next sector
      JR    NZ,NHIT5    ;  column
NHIT9 OR    A
      RET
;
;     Test if Gran is free in GAT
;
TSTBIT      AND   7           ;Get 0 to 7
      RLCA              ;Shift to match BIT n,
      RLCA              ;  opcode
      RLCA
      OR    40H
      LD    (TBIT1+1),A ;Modify BIT instruction
TBIT1 BIT   0,B
      RET
;
;     Set gran to allocated in GAT
;
SETBIT      RLCA              ;Shift to create opcode
      RLCA              ;  to match current bit
      RLCA
      OR    0C7H
      LD    (SBIT1+1),A ;Create SET n, opcode
      XOR   A
SBIT1 SET   0,A
      RET
;
;     Routine reads/writes the Granule Allocation Table
```

```
;
@GATRD      DB    0F6H        ;Set NZ for test
@GATWR      XOR   A           ;Set Z for test
      PUSH  DE
      PUSH  HL
      PUSH  AF          ;Save flag for test
      CALL  @DIRCYL
      LD    HL,DIRBUF$
      LD    E,L         ;Set E to 0
      POP   AF          ;Recover flag for R/W
      JR    Z,GATRW1    ;Go if @GATWR
      CALL  @RDSSC
      LD    A,14H       ;Init "GAT read error"
      JR    GATRW2
GATRW1      CALL  @WRSSC            ;Protected sector write
      CALL  Z,@VRSEC    ;Verify if OK
      CP    6           ;Protected sector?
      LD    A,15H       ;Init "GAT write error"
GATRW2      POP   HL
      POP   DE
      RET
;
;     Read or write the Hash Index Table
;
@HITRD      DB    0F6H        ;Set NZ for test
@HITWR      XOR   A           ;Set Z for test
      PUSH  BC
      PUSH  DE
      PUSH  AF          ;Save flag for test
      CALL  @DIRCYL           ;D => directory cylinder
      LD    E,1         ;E => HIT sector
      LD    HL,SBUFF$   ;HL => HIT buffer area
      POP   AF          ;Recover flag for RD/WR
      JR    Z,HITRW1    ;Go if @HITWR
      CALL  @RDSSC            ;Read cyl D, sector E
      LD    A,22        ;Init "HIT read error"
      JR    HITRW2
HITRW1      CALL  @WRSSC            ;Protected sector write
      CALL  Z,@VRSEC    ;Verify the write
      CP    6           ;Protected sector?
      LD    A,23        ;"HIT write error"
HITRW2      POP   DE          ;Message for other than
      POP   BC          ;  attempt protected sector
      RET
;
;     Routine to read a directory sector
;     B => DEC ot FPDE, C => logical drive number
;     HL <= will point to directory record in SBUF$
;
@DIRRD      PUSH  DE
      CALL  CALCDIR           ;Set HL to SBUFF$
      PUSH  HL
      LD    L,0         ;Start of bfr
      CALL  @RDSSC            ;Read it
      POP   HL
      LD    A,17        ;Init to dir read error
      POP   DE
      RET
;
;     Routine to get directory access data
;     B => DEC
;     DE <= cylinder and sector needed
;     HL <= pointer to directory record in SBUFF$
;
CALCDIR     CALL  @DIRCYL           ;Get directory cyl in D
      LD    A,B         ;Calculate record start
      AND   0E0H        ;  from the DEC
      LD    L,A
      LD    H,SBUFF$>8  ;Point to buffer start
      XOR   B           ;Calculate directory
      ADD   A,2         ;  sector needed
      LD    E,A
      RET
;
;     Read system sector, D=Track, E=Sector, HL=Buffer
;
@RDSSC      CALL  READIR
      RET   Z
      PUSH  DE
      LD    DE,1        ;Pt to trk 0, sec 1
      CALL  @RDSEC            ;Read to find dir cyl
      POP   DE
      RET   NZ
      PUSH  HL
      INC   HL          ;Pt to dir trk #
      INC   HL
      LD    D,(HL)            ;P/u direc trk fr bootsec
      LD    H,9         ;Update memory table
      CALL  DCTFLD@
      LD    L,A
      LD    (HL),D
      POP   HL
READIR      CALL  @RDSEC            ;Retry dir read
      SUB   6           ;Test protected
      RET
;
@DIRCYL     LD    A,9
      CALL  @DCTBYT           ;Get the dir cylinder
      LD    D,A
      RET
;
MAXCYL      LD    A,6
      PUSH  BC
      LD    C,(IX+6)
      CALL  @DCTBYT           ;Get highest # cyl
      INC   A           ;Adjust for zero offset
      POP   BC
      RET
;
;     Multiply register E by register A
;
@MUL8 PUSH  BC          ;Mult A x E
      LD    D,A         ;Multiplier into D
      XOR   A           ;Clear accumulator
      LD    B,8         ;Init to 8 bits
MEA1  ADD   A,A         ;Bits left A
      SLA   E           ;Bits left E into C flag
      JR    NC,MEA2           ;Unless Cy flag, do not add
      ADD   A,D         ;Effective multiplication
MEA2  DJNZ  MEA1        ;Count for 8 bits
      POP   BC          ;Restore BC
      RET               ;Product is in A
;
;     Calculate relative cylinder for granule needed
;
RELCYL      LD    E,A
      CALL  @DCTBYT-5   ;Get # of grans/track
      LD    B,A         ;Hang on to this
      RLCA
      RLCA
      RLCA              ;Bits 5-7 => bits 0-2
      AND   7
      INC   A           ;Adjust from logical 0
      CALL  CKDBLBIT
;
;     Divide register E by register A
;
@DIV8 PUSH  BC
      LD    C,A         ;Divisor into C
      LD    B,8         ;Initialize for 8 bits
      XOR   A           ;Zero accumulator
DEA1  SLA   E           ;Bits left E into Carry
      RLA               ;Rotate dividend into E
      CP    C           ;Divisor > dividend?
      JR    C,DEA2            ;Yes, bypass and continue shift
      SUB   C           ;Effective division
      INC   E           ;Set rotating bit 0 of E
DEA2  DJNZ  DEA1        ;Loop for 8 bts
      LD    C,A         ;Save remainder in C
      LD    A,E         ;Quotient into A
      LD    E,C         ;Remainder into E
      POP   BC          ;Restore regs BC
      RET
;
;     Routine to double the A register if DBL bit is set
;
CKDBLBIT    EQU   $
      LD    D,A         ;Adjust for 2-sided &
      LD    A,4         ;  calculate # of cyls
      CALL  @DCTBYT
      BIT   5,A         ;Test if 2-sided
      LD    A,D
      JR    Z,$+3       ;Double the grans if 2
      ADD   A,A         ; & fall through to DIV8
      RET
      END
```

LOADER/ASM

```
;LOADER/ASM - LS-DOS 6.2
CORE$ DEFL  $
      ORG   SVCTAB$
;
;     Supervisor Call table - Page 5
;
      DW    @IPL,@KEY,@DSP,@GET           ;0-3
      DW    @PUT,@CTL,@PRT,@WHERE         ;4-7
      DW    @KBD,@KEYIN,@DSPLY,@LOGER     ;8-11
      DW    @LOGOT,@MSG,@PRINT,@VDCTL     ;12-15
      DW    @PAUSE,@PARAM,@DATE,@TIME     ;16-19
      DW    @CHNIO,@ABORT,@EXIT,SVCERR    ;20-23
      DW    @CMNDI,@CMNDR,@ERROR,@DEBUG   ;24-27
      DW    @CKTSK,@ADTSK,@RMTSK,@RPTSK   ;28-31
      DW    @KLTSK,@CKDRV,@DODIR,@RAMDIR  ;32-35
      DW    SVCERR,SVCERR,SVCERR,SVCERR   ;36-39
      DW    @DCSTAT,@SLCT,@DCINIT,@DCRES  ;40-43
      DW    @RSTOR,@STEPI,@SEEK,@RSLCT    ;44-47
      DW    @RDHDR,@RDSEC,@VRSEC,@RDTRK   ;48-51
      DW    @HDFMT,@WRSEC,@WRSSC,@WRTRK   ;52-55
      DW    @RENAME,@REMOVE,@INIT,@OPEN   ;56-59
      DW    @CLOSE,@BKSP,@CKEOF,@LOC      ;60-63
      DW    @LOF,@PEOF,@POSN,@READ        ;64-67
      DW    @REW,@RREAD,@RWRIT,@SEEKSC    ;68-71
      DW    @SKIP,@VER,@WEOF,@WRITE       ;72-75
      DW    @LOAD,@RUN,@FSPEC,@FEXT       ;76-79
      DW    @FNAME,@GTDCT,@GTDCB,@GTMOD   ;80-83
      DW    SVCERR,@RDSSC,@GATRD,@DIRRD   ;84-87
      DW    @DIRWR,@GATWR,@MUL8,@MUL16    ;88-91
      DW    SVCERR,@DIV8,@DIV16,SVCERR    ;92-95
      DW    @DECHEX,@HEXDEC,@HEX8,@HEX16  ;96-99
      DW    @HIGH$,@FLAGS,@BANK,@BREAK    ;100-103
      DW    @SOUND,@CLS,@CKBRKC,SVCERR    ;104-107
      DW    SVCERR,SVCERR,SVCERR,SVCERR   ;108-111
      DW    SVCERR,SVCERR,SVCERR,SVCERR   ;112-115
      DW    SVCERR,SVCERR,SVCERR,SVCERR   ;116-119
      DW    SVCERR,SVCERR,SVCERR,SVCERR   ;120-123
      DW    SVCERR,SVCERR,SVCERR,SVCERR   ;124-127
      ORG   CORE$
;
;     Routine to set or retrieve HIGH$/LOW$
;
@HIGH$      LD    A,H         ;Test if put or get
      OR    L
      JR    Z,GETHILO   ;Go if get
      LD    A,(CFLAG$)  ;Is HIGH$ changeable?
      RRCA
      LD    A,43        ;Init SVC parm error
      RET   C           ;Back with NZ
      INC   B           ;Test for HIGH$/LOW$
      DEC   B
      JR    NZ,PUTLO    ;Go if LOW$
      LD    (HIGH$),HL  ;Set new HIGH$
GETHI LD    HL,(HIGH$)  ;P/u the value &
```

```
      RET               ;  ret with Z flag
GETHILO     INC   B           ;Test for HIGH$/LOW$
      DEC   B
      JR    Z,GETHI
      LD    HL,(LOW$)   ;P/u LOW$
PUTLO LD    (LOW$),HL   ;Get LOW$
      XOR   A           ;Set Z flag
      RET
;
@FLAGS      LD    IY,FLGTAB$
      RET
;
@BREAK      PUSH  HL          ;Save user vector
      LD    HL,(BRKVEC$)      ;P/u current vector
      EX    (SP),HL           ;Save current & get user
      LD    (BRKVEC$),HL      ;Stuff new vector
      POP   HL          ;Recover old vector
      RET
;
@WHERE      POP   HL
      JP    (HL)
;
;     Code for these SVCs is in the system overlays
;
@CMNDR      LD    A,0A3H            ;Interpret command & RET
      RST   28H
@CMNDI      LD    A,0B3H            ;Interpret a command
      RST   28H
@FSPEC      LD    A,0C3H            ;Parse a filespec
      RST   28H
@FEXT LD    A,0D3H            ;Optional default EXT
      RST   28H
@PARAM      LD    A,0E3H            ;Parameter scanner
      RST   28H
@OPEN LD    A,94H       ;Open a file
      RST   28H
@INIT LD    A,0A4H            ;Initialize a file
      RST   28H
@GTDCB      LD    A,0B4H            ;Get a DCB vector
      RST   28H
@CKDRV      LD    A,0C4H            ;Drive available?
      RST   28H
@RENAME     LD    A,0F4H            ;Rename a file
      RST   28H
@CLOSE      LD    A,95H       ;Close a file
      RST   28H
@FNAME      LD    A,0A5H            ;Recover filespec
      RST   28H
@DBGHK      RET               ;Init DEBUG off (NOP=on)
@DEBUG      PUSH  AF
      LD    A,97H       ;Enter system Debugger
      RST   28H
EXTDBG$     DW    ORARET@           ;Hook for extended DEBUG
@REMOVE     LD    A,9CH       ;Remove a file/device
      RST   28H
@DOKEY      LD    A,0CDH            ;DO execution
      RST   28H
```

```
@RAMDIR     LD    A,09EH            ;Directory data
      RST   28H
@DODIR      LD    A,0AEH            ;Directory data
      RST   28H
@GTMOD      LD    A,0BEH            ;Get module address
      RST   28H
;
;     These SVCs handle the disk primitive requests
;
@DCSTAT     XOR   A           ;FDC status
      JR    IOFUNC
TAPDRV      LD    A,(LDRV$)   ;P/u drive #
      LD    C,A
@SLCT LD    A,1         ;Select drive
      JR    IOFUNC
@DCINIT     LD    A,2         ;FDC init
      JR    IOFUNC
@DCRES      LD    A,3         ;FDC reset
      JR    IOFUNC
@RSTOR      LD    A,4         ;Restore to cyl 0
      JR    IOFUNC
@STEPI      LD    A,5         ;Step in 1 cyl
      JR    IOFUNC
@SEEK LD    A,6         ;Seek a track/sector
      JR    IOFUNC
@RSLCT      LD    A,7         ;Re-select drive
      JR    IOFUNC
@RDHDR      LD    A,8
      JR    IOFUNC
@VRSEC      LD    A,10        ;Verify a sector
      JR    IOFUNC
@RDTRK      LD    A,11
      JR    IOFUNC
@HDFMT      LD    A,12
      JR    IOFUNC
@WRSEC      LD    A,13        ;Write standard sector
      JR    IOFUNC
@WRSSC      LD    A,14        ;Write a system sector
      JR    IOFUNC
@WRTRK      LD    A,15        ;Write a track
      JR    IOFUNC
@RDSEC      LD    A,9         ;Read a sector
;
IOFUNC      PUSH  BC          ;Save reg pair
      LD    B,A         ;Xfer the function code
;
;     Bring up bank 0
;
      PUSH  BC
      XOR   A
      LD    B,A         ;Set bank function 0,
      LD    C,A         ;  bank number 0
      CALL  @BANK       ;Bring up bank
      POP   AF          ;Perform 'EX (SP),BC'
      PUSH  BC
      PUSH  AF
      POP   BC
```

```
;
;      Continue disk I/O setup
;
       LD    A,C          ;Xfer the drive code
       LD    (LDRV$),A
       PUSH  IY
       CALL  @GTDCT             ;Get DCT address in IY
       LD    A,20H        ;Set illegal drive #
       OR    A            ;  if drive disabled
       CALL  GODOIO
       POP   IY
;
;      Bring back the old bank
;
       POP   BC
       PUSH  AF           ;Save disk I/O ret code
       LD    A,102        ;Set for @BANK
       RST   28H          ;No need to ck for error
                          ;  from @BANK
       POP   AF
       POP   BC
       RET
;
GODOIO       JP    (IY)
;
@GTDCT       PUSH  HL           ;Get I/O routine addr
       CALL  DCTFLD@            ;  into IY
       EX    (SP),HL
       POP   IY
       RET
;
;      Entry to get DCT+8 of FCB (IX) drive spec
;
D@FBYT8      LD    C,(IX+6)    ;P/u drive
;
;      Entry to get DCT+8 of Reg C drive spec
;
DCTBYT8@     EQU   $
       LD    A,8
;
;      Entry to get byte (Reg A) from DCT of Reg C drive
;       C => logical drive specification
;       A => relative byte requested from DCT
;       A <= data at position requested
;
@DCTBYT      PUSH  HL           ;Save the register pair
       LD    H,A          ;Xfer relative position
       CALL  DCTFLD@            ;Get HL pointing to
       LD    L,A          ;  DCT position
       LD    A,(HL)             ;Get the byte
       POP   HL
       RET
;
;      Entry to get HL pointing to DCT byte Reg C, Reg A
;       C => logical drive number
;       A => relative byte in DCT requested
;      HL <= start of requested DCT for the drive
;      A <= low order pointer to relative byte request
;
DCTFLD@     LD    A,C         ;Get drive spec &
       AND   7           ;  strip all but bits 0-2
       ADD   A,A         ;Times 2
       LD    L,A         ;  & saved
       ADD   A,A         ;Times 4
       ADD   A,A         ;Times 8
       ADD   A,L         ;Times 10
       ADD   A,70H       ;Add DCT offset from 0
       LD    L,A         ;Point L to DCT low order
       ADD   A,H         ;Add in rel posn desired
       LD    H,DCT$>8    ;Point H to DCT high order
       RET
;
;     Process supervisory calls <0-127>
;
SVCUSER     CP    26          ;Check for @ERROR
       JR    Z,ERRSVC    ;Skip next if so
       LD    (LSVC$),A   ;Store SVC # as Last Exctd
       EX    (SP),HL           ;P/u RET address
       LD    (SVCRET$),HL      ;  and save it
       EX    (SP),HL           ;Restore RET address
ERRSVC      PUSH  HL          ;Save HL
       RLCA              ;Multiply by 2
       LD    H,SVCTAB$>8 ;Base of Table
       LD    L,A         ;Set up the low order
       LD    A,(HL)            ;P/u table entry
       INC   L
       LD    H,(HL)
       LD    L,A         ;SVC addr is in HL
       EX    (SP),HL           ;P/u HL & stuff vector
       LD    A,C         ;Xfer for PUT type ops
       RET
;
;     RST 28H vector - System & user SVCs
;
RST28 OR    A           ;Test if bit 7 set
       JP    P,SVCUSER   ;Jump on user SVC attempt
       EX    (SP),HL           ;Discard return addr &
       PUSH  AF          ;  save HL, AF
       LD    HL,@DBGHK   ;Set up DEBUG linkage
       LD    A,(HL)
       LD    (SET@EXEC),A
       LD    (HL),0C9H
       POP   AF          ;Restore AF,HL
       POP   HL
HKRES$      CALL  CKMOD@            ;Get overlay if needed
       LD    A,0         ;P/u new overlay #
OVRLYOLD    EQU   $-1
       LD    (OVRLY$),A  ;  & update current
TRANSFR     CALL  0           ;Trnsf addr of SYSx
       PUSH  AF
       LD    A,0         ;Set to C9 if EXEC only
SET@EXEC    EQU   $-1
       LD    (@DBGHK),A
       POP   AF
       RET
;
;     DOS command overlay request
;
CKMOD@      PUSH  HL
       LD    H,A         ;Save command value
       LD    A,B
       LD    (EXOVR2+1),A      ;Set overlay #
       LD    A,H
       OR    1           ;Set for SYS6 & SYS7
       CP    89H         ;Is it either?
       LD    A,H         ;Get back the correct #
       JR    Z,EXOVR           ;Sys6/7 req? Use ISAM!
       CP    8AH         ;Sys8 also ISAM
       JR    Z,EXOVR
       LD    A,(OVRLY$)  ;P/u current overlay
       XOR   H           ;Ck if it's the one
       AND   0FH         ;  we need to execute
       LD    A,H
       LD    (OVRLYOLD),A      ;Update current tempy
       LD    HL,OVERLAY  ;Init to SYSx entry
       JR    Z,EXOVR3    ;Go exec if resident
;
;     Execute a system overlay
;
EXOVR PUSH  DE
       PUSH  BC
       AND   0FH         ;Get right nybble
       BIT   3,A         ;Check for SYS0-7
       JR    Z,EXOVR1    ;  w/o changing C flg
       ADD   A,18H       ;Adjust for Sys8-15
EXOVR1      LD    (SFCB$+7),A
       LD    B,A         ;Set DEC for directory
       LD    A,20H       ;Set bit 5 of FCB+1
       LD    (SFCB$+1),A
       SBC   HL,HL       ;Carry is clear here
       LD    (SFCB$+10),HL     ;Zero NRN
       LD    C,H         ;Init for drive 0
       CALL  @DIRRD            ;Read dir entry
       JR    NZ,EXERR    ;Go if error
       LD    A,(HL)            ;Was overlay purged?
       AND   50H         ;  or is it non-system?
       XOR   50H
       LD    A,7         ;Init "deleted error
       JR    NZ,EXERR
       LD    A,L
       ADD   A,22        ;Point to 1st extent
       LD    L,A
       LD    DE,SFCB$+14 ;Extent field in FCB
       CALL  PAT1        ;Stuff 1st two extents
EXOVR2      LD    B,0         ;P/u ISAM # or zero
       LD    E,SFCB$&0FFH
       CALL  LOADER            ;Read system overlay
EXERR POP   BC
       POP   DE
EXOVR3      LD    (TRANSFR+1),HL    ;Stuff overlay entry pt
       POP   HL
       RET   Z
       JR    SYSERR            ;Go if I/O error on read
;
;     Routine to calculate first two extents of SYS file
;
PAT1  CALL  PAT1A       ;Move first extent
       AND   1FH         ;Comput # of granules
       INC   A
       LD    (DE),A            ;And store in FCB
       INC   DE
       XOR   A
       LD    (DE),A
       INC   DE
PAT1A CALL  PAT1B       ;Move second extent
PAT1B LD    A,(HL)
       LD    (DE),A
       INC   HL
       INC   DE
       RET
;
;     System error disolay routine
;     The NOP is provided so an intercept routine vector
;       may be patched in during program development
;
SVCERR      LD    A,43        ;SVC error
       NOP
SYSERR      AND   3FH         ;Strip excess bits
       LD    HL,ERRNUM   ;Pack error number
       CALL  @HEX8       ;  into message
       LD    HL,SYSERR$
       CALL  @LOGOT            ;Log the error & ABORT
       LD    SP,STACK$   ;Reset the Stack Pointer
@ABORT      LD    HL,-1
@EXIT LD    A,93H       ;Exit to DOS
       RST   28H
;
POPERR      POP   HL          ;Pop extended error
@ERROR      PUSH  AF          ;Save the error code
       LD    A,96H       ;Display the error number
       RST   28H
;
SYSERR$     DEFM  'Error '
ERRNUM      DEFB  'xxH',CR
;
;     Routine to RUN a program
;
@RUN  PUSH  HL          ;Save register pair
       LD    HL,SFLAG$
       SET   2,(HL)            ;Turn on RUN flag bit
       CALL  @LOAD       ;Load the program module
       EX    (SP),HL           ;Put transf addr on the stk
;
;     Note: The error code is set to NOT abort. Errors
;     will be passed back to the calling module after
;     @ERROR. Note that HL will contain the error #
;
       JR    NZ,POPERR
```

```
;
;      Place the INBUF$ pointer in regiater pair BC
;
       LD     BC,INBUF$    ;Reflect buffer pointer
;
;      Get TRAADR then test if we need to go to DEBUG
;
       LD     A,(SFLAG$)
       BIT    1,A          ;Go to the program if
       RET    NZ           ;  it's EXEC only access
       BIT    7,A          ;  else test if DEBUG
       JP     NZ,@RST30    ;  is on & go to it
       RET                 ;  else go to program
;
;      This routine LOADs a Load Module Format file
;
@LOAD  LD     B,0          ;LRL=256
       LD     HL,SFLAG$
       SET    0,(HL)             ;Don't set "file open"
       LD     HL,SBUFF$    ;Set buffer to system
       CALL   @OPEN        ;Open the file
       PUSH   DE           ;Save FCB pointer
       CALL   Z,LOADER     ;Load if no OPEN error
       POP    DE           ;Restore FCB pointer
       RET    Z            ;Back if no error
       LD     L,A          ;Xfer the error code
       LD     H,0
       OR     0C0H         ;Set RETurn & abbrev
       CP     0D8H         ;Change "file not in dir
       RET    NZ           ;  to "Program not found"
       ADD    A,7
       RET
;
;      System Command File Loader
;
LOADER        LD     A,B          ;Set overlay # (0 on non-
       LD     (LDR14+1),A ;  SYStem file)
       PUSH   DE           ;Save IX & Xfer FCB to IX
       EX     (SP),IX
       LD     DE,SBUFF$+255     ;Init to end of buffer
       CALL   LDR01        ;Do the load
       POP    IX           ;Recover IX
       RET
;
;      Routine to ignore the LMF record or skip some sections
;
LDR05  CALL   LDR15        ;Get length of "Comment"
       LD     B,A          ;Init B as a counter
LDR06  CALL   LDR15        ;READ & IGNORE this many
       DJNZ   LDR06        ;  bytes, then fall through
;
;      Routine to parse LMF record types
;
LDR01  CALL   LDR15        ;Get Record Type
LDR02  CP     1            ;Start of block?
       JR     Z,LDR08
       CP     2            ;Start of TRAADR?
```

```
LDR03 JR    Z,LDR07
      CP    4           ;End of LIB member?
      JR    Z,LDR12
      CP    8           ;Begin ISAM table entry?
      JR    Z,LDR13
      CP    10          ;End of ISAM map?
      JR    Z,LDR04
      CP    20H         ;Ignore all other controls
      JR    C,LDR05
LDR04 LD    A,22H       ;Load file format error
      OR    A           ;Set NZ condition
      RET
;
;     Grab transfer address
;
LDR07 CALL  LDR15       ;Bypass 2nd X'02'
      CALL  GETADR            ;P/u transfer address
      RET               ;Ret Z or NZ
;
;     Grab load block
;
LDR08 CALL  LDR15       ;P/u block length
      LD    B,A
      CALL  GETADR            ;P/u Load address
      RET   NZ
      DEC   B           ;Adjust length for addr
      DEC   B
LDR09 CALL  LDR15       ;P/u block byte
      LD    (HL),A
      INC   HL
      DJNZ  LDR09       ;Loop until block end
      JR    LDR01
;
LDR12 POP   HL
      RET
;
;     Routine to check ISAM table match
;
LDR13 CALL  LDR15       ;Get record length
      LD    B,A
      CALL  LDR15       ;Get ISAM number
      DEC   B           ;  & decrement counter
LDR14 CP    0           ;Either ISAM# or 0
      JR    NZ,LDR06    ;Go if not a match
      CALL  GETADR            ;  else get the TRAADR
      PUSH  HL          ;  & save it
      CALL  Z,GETADR    ;Get the NRN for member
      JR    NZ,LODERR
      CALL  LDR15       ;Get the sector offset
      LD    E,A         ;Update pointer offset
      PUSH  BC
      LD    B,H         ;Xfer NRN position needed
      LD    C,L
      PUSH  DE          ;Save buffer ptr offset
      PUSH  IX
      POP   DE          ;P/u FCB in DE
      CALL  @POSN       ;Position to ISAM record
      POP   DE           ;Recover buf ptr offset
      POP   BC
      JR    NZ,LODERR
      CALL  LDR17        ;Read the sector
      JR    LDR02        ;Now go read the member
;
;     Routine to get the next file byte
;
LDR15 INC   E            ;Bump buffer pointer
      JR    Z,LDR17            ;Read sector if needed
LDR16 LD    A,(DE)             ;P/u byte from buffer
      RET
LDR17 PUSH  HL           ;Save registers
      PUSH  DE
      PUSH  BC
      CALL  NXTSECT            ;Read next record
      POP   BC
      POP   DE
      POP   HL
      JR    Z,LDR16            ;Bypass if no error
LODERR      POP   BC           ;Pop return address
      RET                ;Return NZ cond
;
;     Routine to get an address field
;
GETADR      CALL  LDR15        ;Get low order byte
      LD    L,A
      CALL  LDR15        ;Get high order byte
      LD    H,A
      CP    A            ;Set Z fl
      RET
;
;     BOOT code brings back the ROM
;
MOD3BUF     EQU   4300H
@IPL  LD    HL,BOOTCOD  ;Code to toggle in ROM
      LD    DE,MOD3BUF  ;Buffer used by ROM
      PUSH  DE          ;This is return address
      LD    BC,BOOTLEN  ;Length of BOOT sequence
      LDIR              ;Transfer boot code
      RET               ;  and Return to it
;
;     End of loader module
;
      END
```
□

TASKER/ASM

```
;TASKER/ASM - LS-DOS 6.2
;
;      Interrupt task table, IM 1
;
CORE$ DEFL  $
      ORG   TCB$
      DW    NOTASK,NOTASK,NOTASK,NOTASK
      DW    NOTASK,NOTASK,NOTASK,NOTASK
      DW    NOTASK,NOTASK,TYPTSK$,NOTASK
      ORG   CORE$
;
;      Model IV task processor
;
RST38@      EQU   $
      EX    (SP),HL
      LD    (PCSAVE$),HL      ;Save for TRACE tsk
      EX    (SP),HL
      PUSH  HL          ;Save HL for now
      PUSH  AF          ;Save AF for now
      LD    HL,NFLAG$   ;Show the system we
      SET   6,(HL)            ;  are in the TASKER
      LD    HL,LBANK$   ;P/u & save the current
      LD    A,(HL)            ;  logical bank #
      LD    (HL),0
      PUSH  AF
      LD    HL,OPREG$   ;Get current memory
      LD    A,(HL)            ;  configuration
      PUSH  AF          ;  & save it
      AND   8CH         ;Strip bits 0, 1, 4-6
      OR    3           ;Bring up regular 64K
      LD    (HL),A
      OUT   (84H),A
INTLAT      EQU   0E0H
      IN    A,(0E0H)    ;Get interrupt latch
      CPL               ;Mod IV is reverse
      LD    HL,INTIM$   ;Store state of int
      LD    (HL),A
      INC   L           ;Advance to int mask
      AND   (HL)        ;Mask the latch bits
      JR    Z,TSTBRK    ;Go if nothing interrupted
NXTVCT      INC   L           ;Ck on INTVC$
      RRA               ;Ck if device interrupted
      JR    C,ACTVTSK
NXTMSK      INC   L           ;Ck all 8 bits of mask
      OR    A           ;When finished, ck overhead
      JR    NZ,NXTVCT   ;  task routine
;
TSTBRK      CALL  KCK@        ;Test <BREAK>,<SHIFT>
      JR    NZ,BREAK?   ;Go if break
TSKEXIT     POP   AF          ;Get previous mem config
      LD    (OPREG$),A  ;  & restore RAM bank
      OUT   (84H),A
      POP   AF
      LD    (LBANK$),A
      LD    HL,NFLAG$   ;Now leaving the TASKER
      RES   6,(HL)            ;  show the system
      POP   AF          ;Restore previous regs
      POP   HL
      EI
RETINST     RET
;
;
;     Found active INTVC$
;
ACTVTSK     PUSH  AF          ;Save the regs
      PUSH  BC
      PUSH  DE
      PUSH  HL
      PUSH  IX
      LD    DE,POPREGS  ;Stack Return vector
      PUSH  DE
      LD    E,(HL)            ;P/u INTVC pointer vector
      INC   L
      LD    D,(HL)
      EX    DE,HL       ;Shift it to HL
      JP    (HL)        ;Go to service routine
;
;     Register restoral after service routine
;
POPREGS     POP   IX
      POP   HL
      POP   DE
      POP   BC
      POP   AF
      JR    NXTMSK            ;Loop to next mask bit
;
;     <BREAK> key detected
;
BREAK?      JR    NC,GOTBRK   ;Go if <BREAK> only
      PUSH  BC          ;Was <SHIFT-BREAK>?
      DI
      CALL  TAPDRV            ;Reselect drive
      POP   BC
      JR    TSKEXIT
;
;     <BREAK> during tasking - enter DEBUG? - user Break?
;
GOTBRK      LD    A,(SFLAG$)  ;Check if <BREAK> key is
      AND   10H         ;  disabled to inhibit
      JR    NZ,TSKEXIT  ;  DEBUG and BREAK vectors
      LD    HL,@DBGHK   ;Merge DEBUG flag &
      OR    (HL)        ;  hook (X'00' or X'C9')
      LD    (HL),0C9H   ;Turn off DEBUG
      INC   HL          ;Point to @DEBUG vector &
      JR    Z,EXITBRK   ;  go if DEBUG is active
;
      LD    A,(PCSAVE$+1)     ;Don't allow vectored break
      CP    MAXCOR$>8   ;  if old PC is in SYSRES
      JR    C,TSKEXIT
      LD    HL,HIGH$+1  ;  or if old PC is
      CP    (HL)        ;  above HIGH$
      JR    NC,TSKEXIT
      LD    HL,0        ;  else ck if BREAK is
BRKVEC$     EQU   $-2
      LD    A,H         ;  to be tapped by user
      OR    L
      JR    Z,TSKEXIT
EXITBRK     POP   AF          ;Discard old mem config
      POP   AF          ;Restore reg AF
      POP   AF
      EX    (SP),HL           ;P/u HL & stack vector
      EI
      RET               ;To DEBUG or BREAK vector
;
;     Real Time Clock interrupt processor
;
RTCPROC     EQU   $
      IN    A,(0ECH)    ;Clear the RTC Interrupt
      LD    A,11        ;Task 11 executes every
      CALL  RTCTASK           ;  RTC interrupt
      LD    HL,TIMSL$
      RLC   (HL)        ;Ck on the time slice
      RET   NC          ;Ignore if nothing
      LD    DE,TIMTSK$  ;  on this interrupt
      PUSH  DE          ;  else init for clocker
      LD    A,8         ;Task 8 at INT/2 if fast
      CALL  RTCTASK
      LD    A,9         ;Task 9 at INT/2 if fast
      CALL  RTCTASK
      LD    A,10        ;Task 10 at INT/2 if fast
      CALL  RTCTASK
      LD    HL,TIMER$   ;Bump the timer at INT/2
      INC   (HL)
      LD    A,(HL)            ;P/u the heart beat
      AND   7           ;For this interrupt,
RTCTASK     RLCA              ;  consider 0-7 only
      ADD   A,TCB$&0FFH ;Add offset to table
      LD    L,A
      LD    H,TCB$>8
      LD    (@RPTSK+1),HL
      LD    E,(HL)            ;P/u task vector addr
      INC   L
      LD    D,(HL)
      PUSH  DE
      POP   IX          ;Also to IX
      EX    DE,HL
      LD    E,(HL)            ;P/u task entry point
      INC   HL
      LD    D,(HL)
      EX    DE,HL
      JP    (HL)        ;Go to task
;
@KLTSK      POP   DE          ;Remove ret
      LD    A,(@RPTSK+1)      ;Pt to task tbl entry
      SUB   TCB$&0FFH
      RRCA
;
@RMTSK      LD    DE,NOTASK   ;Remove entry
```

```
;
@ADTSK      CP    12          ;Too large a task?
      RET   NC          ;Return if too big else
      RLCA              ;  add to task table
      ADD   A,TCB$&0FFH ;Add the offset
      LD    L,A         ;Estab ptr to vector
      LD    H,TCB$>8
CHGTASK     DI
      LD    (HL),E            ;Vector address to
      INC   L           ;  pointer table
      LD    (HL),D
      EI
      RET
;
NOTASK      DW    $-1         ;Current task vector
;
@RPTSK      LD    HL,0        ;P/u last task done
      LD    E,(HL)            ;P/u task vector addr
      INC   HL
      LD    D,(HL)
      EX    DE,HL
      POP   DE          ;Pop ret addr
      JR    CHGTASK
;
;     Routine to check if task slot active
;
@CKTSK      RLCA              ;Task number * 2
      ADD   A,TCB$&0FFH+1     ;Index to task table
      LD    L,A
      LD    H,TCB$>8
      LD    A,NOTASK>8  ;Check match of high
      CP    (HL)        ;  order only
      RET               ; Z or NZ result
      END
```

SYSINIT4/ASM

```
;SYSINIT4/ASM - LS-DOS 6.2
;
;     This is the initialization part of SYSRES
;
TRKREG      EQU   0F1H        ;FDC track register
KB1   EQU   0F401H            ;Keyboard row 1
KB67  EQU   0F460H            ;Keyboard rows 6&7
KB7   EQU   0F440H            ;Keyboard row 7
BOL   EQU   29          ;Beginning of line
;
      ORG   1E00H+START$
;
      DI
      LD    HL,@RSTNMI  ;Reset NMI vector to
      LD    (@NMI+1),HL ;  SYSRES's needs
      LD    HL,PAKNAM$  ;Pt to pack name
      LD    DE,2*80+CRTBGN$+30
      LD    BC,8
      LDIR              ;Move pack name to CRT
      LD    C,8         ;B contains 0 already
      INC   DE          ;Leave 2 spaces
      INC   DE
      LDIR              ;Move pack date to CRT
;
;     Initialization routines
;
      XOR   A           ;Clear out stack area
      LD    HL,STACK$+1 ;Stack start +1
CLRLOOP     DEC   L           ;Move down a byte
      LD    (HL),A            ;Now loop an fill
      JR    NZ,CLRLOOP  ;  and fill with 0's
;
      IM    1
      LD    SP,STACK$   ;Set the stack area
      XOR   A
      LD    (LBANK$),A  ;Set logical bank #
      OUT   (0E4H),A    ;Disable INTRQ & DRQ
;
      LD    HL,S1DCB$
ZERDCB      LD    (HL),A            ;Zero spare DCB area
      INC   L
      JR    NZ,ZERDCB
;
      LD    A,(MODOUT$) ;Set high speed (4 MHz)
      OUT   (0ECH),A    ;  and external bus
      LD    A,(WRINT$)
      OUT   (0E0H),A    ;Enable RTC interrupts
      LD    A,(OPREG$)  ;Set memory configuration
      LD    B,A
      LD    A,0A7H            ;Value for AUX/RAM
      LD    C,@OPREG    ;Set the memory mgt port
      OUT   (C),B       ;Bring up reg RAM
      LD    HL,-1       ;Ck for extended RAM
      LD    (HIGH$),HL
```

```
        LD      (PHIGH$),HL
;       Check the BANKS
        LD      D,(HL)                  ;Save what's in RAM
        LD      (HL),55H        ;Stuff in reg RAM
        OUT     (C),A           ;Switch in alt RAM
        LD      E,(HL)                  ;Save th byte there
        LD      (HL),A                  ;Stuff alt RAM
        OUT     (C),B           ;Switch to reg RAM
        CP      (HL)            ;See what's there now
        LD      (HL),D                  ;Restore original value
        OUT     (C),A           ;Back to reg RAM
        LD      (HL),E                  ;Restore original byte
        OUT     (C),B           ;Back to reg RAM
        LD      A,0FEH                  ;Init BAR$ for bank 0
        JR      Z,$+4           ;Bypass if only 64K
        LD      A,0F8H                  ;Init BAR$ for bank 0-2
        LD      (BAR$),A        ;Load Bank Avail RAM
        LD      (BUR$),A        ;Load Bank Used RAM
        LD      A,(FEMSK$)      ;P/u port FE mask
        OUT     (0FEH),A        ;  & set it
        DS      3%0             ;Space for a JP instr
;
;       Update DCT$ info for SYSTEM drive
;
        LD      A,(BOOTST$) ;P/u Boot step rate
        AND     3               ;Strip all but bits 0,1
        LD      B,A             ;Save tempy
        LD      HL,DCT$+3       ;Pt to DCT Step
        LD      A,(HL)                  ;P/u DCT Step
        AND     0FCH            ;Strip bits 0,1
        OR      B               ;Merge boot step fr B
        LD      (HL),A                  ;Update DCT
        IN      A,(TRKREG)      ;Update DCT with current
        LD      (DCT$+5),A      ;  track posn of head
;
        LD      DE,KIDCB$       ;Flush type,init ptrs
        LD      A,3             ;Clear type-ahead fctn
        CALL    @CTL            ;Send to *KI
        EI                      ;Interrupts on
;
;       P/u CONFIG status & set ZERO byte
;
        LD      HL,ZERO$
        LD      A,(HL)                  ;Set to NOP if SYSGENed
        LD      (HL),0                  ;Make always zero byte
        PUSH    AF              ;Save SYSGEN flag
;
;       Check if date prompt is to be suppressed
;
        LD      A,(DTPMT$)      ;No prompt for DATE?
        OR      A
;
;       Check on currency of date
;
        LD      HL,DATE$        ;Point to Year
        LD      C,(HL)                  ;  & save in reg C
        LD      (HL),0                  ;  while resetting to zero
```

```
      INC   HL          ;Bump to day
      LD    B,(HL)            ;  & save in reg B
      LD    (HL),0            ;  while resetting to zero
      INC   HL          ;Bump to Month
      LD    A,(HL)            ;  & save in reg A
      LD    (HL),0            ;  while resetting to zero
      JP    NZ,TIMIN    ;Ck time if DATE=OFF
      LD    L,0FFH&(CFGFCB$+31)     ;Reset pointer
;
      IF    @INTL
      LD    (HL),B            ;Stuff day
      DEC   HL
      LD    (HL),A            ;Stuff month
      ELSE
      LD    (HL),A            ;Stuff month
      DEC   HL
      LD    (HL),B            ;Stuff day
      ENDIF
;
      DEC   HL
      LD    (HL),C            ;Stuff Year
      EX    DE,HL       ;  & point DE to CFGFCB$+29
      DEC   A           ;Check for month range <1-12>
      CP    12          ;OK if 0-11 now
      JR    C,DATIN1
;
DATIN LD    HL,27!(21<8)      ;Set video row,col
      LD    DE,DATEPR   ;DATE? question
      LD    BC,'0'!8<8  ;Set buf len & char
      CALL  GETPARM           ;Get response
      JR    NC,DATIN    ;Jump on format error
DATIN1      LD    A,(DE)            ;Is year a leap year?
      LD    C,A         ;Save year for later
      SUB   80          ;Reduce for range test
      CP    8
      JR    NC,DATIN
      AND   3
      LD    A,28        ;Init February
      JR    NZ,NOTLEAP
      LD    HL,DATE$+3+1      ;Set leap flag
      SET   7,(HL)
      INC   A           ;Feb to 29 days
NOTLEAP     LD    HL,MAXDAY$+2      ;Set Feb max day #
      LD    (HL),A
;
      IF    @INTL
      NOP               ;Keep same length
      ELSE
      INC   DE          ;Bump to DAY
      ENDIF
      INC   DE          ;Bump to month & get it
      LD    A,(DE)
      LD    B,A         ;Save month in reg B
      DEC   A           ;Range check
      CP    12
      JR    NC,DATIN    ;Go if 0 or >12
      DEC   HL          ;Point to Jan entry
      ADD   A,L          ;Index the month
      LD    L,A
;
      IF    @INTL
      INC   DE           ;Point to day
      ELSE
      DEC   DE           ;Point to day
      ENDIF
;
      LD    A,(DE)             ;P/u day entry
      DEC   A            ;Reduce for range test
      CP    (HL)
      JR    NC,DATIN     ;Go if too large (or 0)
;
;     Range checks OK - move into DATE$
;
      LD    HL,DATE$+2
      INC   A            ;Compensate for DEC A
      LD    (HL),B             ;Stuff month
      DEC   L
      LD    (HL),A             ;Stuff day
      DEC   L
      LD    (HL),C             ;Stuff year
;
;     Date is in DATE$ - display it
;
      LD    A,C
      PUSH  AF           ;Save year for later
      AND   3            ;Check on leap year
      LD    HL,MAXDAY$+2       ;Init and adjust Feb
      LD    (HL),28            ;  as required
      JR    NZ,$+3
      INC   (HL)         ;Bump to 29
      LD    A,(DATE$+2) ;P/u month & Xfer to B
      LD    B,A
      LD    A,(DATE$+1) ;P/u day of month
;
;     Compute day of year and day of week
;
      LD    L,A          ;Start off with days
      LD    H,0          ;  in this month
      LD    DE,MAXDAY$
DAYLP LD    A,(DE)
      ADD   A,L          ;8 bit add to 16 bit
      LD    L,A
      ADC   A,H          ;Add in high order & carry
      SUB   L            ;Subtract off low order
      LD    H,A          ;Update high order
      INC   DE
      DJNZ  DAYLP
      EX    DE,HL        ;Move day of year to DE
      LD    HL,DATE$+3  ;  and store
      LD    (HL),E
      INC   HL
      LD    A,D          ;Get bit "8"
      OR    (HL)         ;  and OR it in
      LD    (HL),A             ;Then put it back
      EX    DE,HL       ;Get Day of Yr back to HL
      POP   AF          ;Pop the year & mask
      AND   7           ;Compute day of the week
      LD    E,A         ;  offset
      ADD   A,3
      RRCA
      RRCA
      AND   3
      ADD   A,E
      LD    E,A         ;And add it in
      LD    D,0         ;Add into HL
      ADD   HL,DE
      INC   HL          ;To start in right place
      LD    C,7         ;Now divide by 7 (B=0)
DIV7  SBC   HL,BC       ;Subtract weeks (7-days)
      JR    NC,DIV7           ;Until underflow
      LD    A,L
      ADD   A,8         ;Add back to get 1-7
      LD    B,A         ;Save in reg B
      RLCA              ;Shift to bits 1-3
      LD    C,A         ;Save tempy
      LD    HL,DATE$+3+1
      LD    A,(HL)            ;Pack into field
      AND   0F1H
      OR    C
      LD    (HL),A
      PUSH  BC
      LD    HL,27!(21<8)      ;Set video row,col
      LD    B,3         ;Set function code 3
      CALL  @VDCTL            ;  to position cursor
      POP   BC
      LD    HL,DAYTBL$
      CALL  SPACE4            ;Write out the DAY
      LD    A,','
      CALL  @DSP
      LD    A,' '
      CALL  @DSP
      LD    A,(DATE$+2) ;P/u month number
      LD    B,A
      LD    L,MONTBL$&0FFH    ;Reset HL for month table
      CALL  DSPMDY            ;Write out the month name
      LD    A,' '
      CALL  @DSP
      LD    A,(DATE$+1) ;P/u day
      DEC   B           ;From 0 to X'FF'
DIV10 INC   B           ;Divide by 10
      SUB   10          ;  with quotient in B
      JR    NC,DIV10
      PUSH  AF          ;Save remainder (-10)
      LD    A,B         ;P/u quotient
      ADD   A,'0'       ;Change to ASCII
      CP    '0'         ;Zero?
      CALL  NZ,@DSP           ;Display if not
      POP   AF          ;Get back remainder
      ADD   A,3AH       ;Change to ASCII
      CALL  @DSP
      LD    HL,PARTYR   ;Part of year
      CALL  @DSPLY
      LD    A,(DATE$)   ;Form last year digit
      AND   7
      ADD   A,'0'
      CALL  @DSP        ;  and display it
;
;     Prompt for time
;
TIMIN LD    A,(TMPMT$)  ;Time to be prompted
      OR    A
      JR    NZ,SELDCT   ;Skip if not
TIMIN0      LD    HL,27!(22<8)
      LD    DE,TIMEPR   ;Set prompt message
      LD    BC,'0'!(8<8)      ;Set len & separ char
      CALL  GETPARM
      JR    NC,TIMIN0   ;Loop on format error
      LD    HL,CFGFCB$+31
      LD    A,23
      CP    (HL)        ;Test hour range
      JR    C,TIMIN0
      DEC   HL
      LD    A,59
      CP    (HL)        ;Test minute range
      JR    C,TIMIN0
      DEC   HL
      CP    (HL)        ;Test the second range
      JR    C,TIMIN0
      LD    DE,TIME$    ;Move the time value
      LD    BC,3        ;  into the TIME$ field
      LDIR
;
;     Check on any AUTO command
;
SELDCT      LD    HL,INBUF$
      LD    A,(HL)            ;Pt to 1st byte of AUTO
      CP    '*'         ;<BREAK> disable?
      JR    NZ,CKDCR
      INC   HL
      LD    A,0E6H            ;Set <BREAK> bit in flag by
      LD    (STUB1+1),A ;  changing RES 4,(SFLAG$)
                        ;  to SET 4,(SFLAG$)
      JR    AUTO?
GETKB17     CALL  ENADIS_DO_RAM
      LD    A,(KB1!KB7) ;Scan row 1 & 7
      RET
CKDCR CALL  GETKB17           ;Strobe keyboard
      BIT   4,A         ;Is 'D' depressed?
      PUSH  HL          ;Save auto command pt
      LD    HL,@ABORT   ;P/u abort address
      EX    (SP),HL           ;Swap them around
      JP    NZ,@DEBUG   ;DEBUG on <D>
      POP   DE          ;Stack integrity
      CPL
      AND   1           ;No AUTO if <ENTER>
      JR    Z,NOAUT1
AUTO? LD    A,(HL)            ;Any AUTO command?
      CP    CR          ;None if equal
```

```
NOAUT1      POP   DE          ;Get back SYSGEN flag
      LD    A,D         ;  & move into reg A
      LD    DE,@EXIT    ;Where to go after boot
      LD    BC,0        ;Init BC(HL)=0 for @EXIT
      JR    Z,NOAUT           ;Go if no AUTO
      PUSH  HL          ;Save buffer pointer
      LD    HL,CURSET   ;Point to cursor setting
      INC   (HL)        ;Bump it down a line
      POP   HL          ;Recover INBUF$ pointer
      LD    DE,@CMNDI   ;Low order of @CMNDI
      PUSH  DE          ;Put on stack for RET
      LD    B,H         ;Put INBUF$ pointer on
      LD    C,L         ;  stack for @CMNDI
      LD    DE,@DSPLY   ;But do this first
NOAUT PUSH  DE          ;Put on stack for RET
      PUSH  BC          ;Either INBUF$ or 0
      LD    HL,STUB
      LD    DE,MOD3BUF+80     ;Must move out of way
      LD    BC,STUBLEN  ;  amount to move
      PUSH  DE          ;Add ret vector to stack
      LDIR              ;Move stub up
      CALL  GETKB67
      LD    DE,DCT$           ;Set up to move DCTs
      LD    HL,MOD3BUF  ;  from confined area
      LD    BC,80       ;Count fo DCTs (8*10)
      EXX               ;Keep in alternate set
      AND   82H         ;Load config if zero
      RET   NZ          ;No config > Go back
      LD    HL,21<8           ;Set to line 21
      LD    B,3         ;Position cursor
      CALL  @VDCTL
      LD    HL,CONFIG$  ;Show Sysgen message
      CALL  @DSPLY
      LD    DE,CFGFCB$  ;Set up to load config
      JP    @LOAD       ;Go to load CONFIG/SYS
;
CONFIG$     DB    '** SYSGEN **',03 ; Config DSPLY
;
GETKB67     LD    HL,KB67           ;Check <CLEAR> key
      LD    C,A
      CALL  ENADIS_DO_RAM
      LD    A,C
      OR    (HL)        ;Key down OR not SYSGENed
      RET
;
;     Final initialization code
;
STUB  LD    HL,SFLAG$
STUB1 RES   4,(HL)            ;Test or SET Break bit
                        ;  without changing Z/NZ
      JR    NZ,NOTSG    ;Go if no SYSGEN found
      LD    HL,MODOUT$  ;P/u ptr to port mask
      LD    A,(HL)            ;P/u mask byte
      OUT   (0ECH),A    ;Speed it up
      EXX               ;Set to move DCTs
      LDIR              ;Move them
      CALL  @ICNFG            ;Init config
```

```
NOTSG EQU   $
      LD    C,7
SETCYL0     EQU   $
      CALL  @GTDCT
      BIT   3,(IY+3)    ;If hard drive, don't stuff FF
      JR    NZ,NOFF           ;  & don't restore
      LD    (IY+5),0FFH ;Set in case no restore
      LD    A,(RSTOR$)  ;Do we restore the drives?
      OR    A
      CALL  Z,@RSTOR    ;Restore drives 1-7
NOFF  DEC   C
      JR    NZ,SETCYL0
      LD    HL,21<8           ;Set cursor
CURSET      EQU   $-1
      LD    B,3
      CALL  @VDCTL
;
;     Detect Model 4 or 4P and adjust TFLAG$
;     Look at 'MODEL' at 4018H. If so, MOD-4P (5)
;
;
      LD    DE,'OM'           ;Lo/Hi of 'MO' in 'MODEL'
      LD    HL,(4018H)  ;P/u 4P ROM leftover
      SBC   HL,DE       ;Check if it's 'MO'
      LD    A,4         ;Init for regular MOD 4
      JR    NZ,MOD4REG
      LD    A,5         ;Change to MOD 4P
MOD4REG     LD    (TFLAG$),A  ;Init machine type flag
;
      LD    HL,@RST38   ;Insert JP instruction to
      LD    (HL),0C3H   ;  activate task processor
      POP   HL          ;Pop INBUF$
      RET               ;To @CMD or @DSPLY,@CMNDI
      DS    12%0        ;Zero fill for future code
STUBEND     EQU   $
STUBLEN     EQU   STUBEND-STUB
;
;     Date and Time prompting
;
GETPARM     PUSH  BC          ;Save separator char
      PUSH  DE          ;Save message pointer
      LD    B,3
      CALL  @VDCTL            ;Position the cursor
      POP   HL          ;Recover message pointer
      CALL  @DSPLY            ;  & display the message
      LD    HL,OVERLAY  ;Buffer for reply
      POP   BC
      PUSH  BC          ;Use/save again separator
      CALL  @KEYIN            ;Get reply & wait a bit
      XOR   A           ;  disable test
      OR    B
      POP   BC          ;  of key prior to AUTO
      RET   Z           ;Ret with NC if no entry
      PUSH  BC
      LD    B,40H       ;Delay for wait
      CALL  @PAUSE            ;  to let finger off
      POP   BC
```

```
;
;      Routine to parse DATE entry
;
PARSDAT      LD    DE,CFGFCB$+31     ;Point to end of buffer
      LD    B,3          ;Process 3 fields
PRSD1 PUSH  DE           ;Save pointer
;
;      Routine to parse a digit pair
;
      CALL  PRSD3        ;Get a digit
      JR    NC,PRSD2     ;Jump if bad digit
      LD    E,A          ;Multiply by 10
      RLCA
      RLCA
      ADD   A,E
      RLCA
      LD    E,A
      CALL  PRSD3        ;Get another digit
      JR    NC,PRSD2     ;Jump on bad digit
      ADD   A,E          ;Accumulate new digit
      LD    E,A          ;Save 2-digit value
      SCF                ;Show valid
      LD    A,E          ;Xfer field value
PRSD2 POP   DE           ;Recover pointer
      RET   NC           ;Ret if bad digit pair
      LD    (DE),A             ;Else stuff the value
      DEC   B            ;Loop countdown
      SCF
      RET   Z            ;Ret when through
      DEC   DE           ;Point to preceding field
      LD    A,(HL)             ;Ck for valid separator
      INC   HL           ;Bump pointer
      CP    ':'          ;Check for colon ':'
      JR    Z,PRSD1            ;  loop if match
      CP    C            ;Separator char required
      JR    NC,PRSD4     ;Exit if bad char
      JR    PRSD1        ;  else loop now
PRSD3 LD    A,(HL)             ;P/u a digit &
      INC   HL           ;  convert to binary
      SUB   30H
PRSD4 CP    10
      RET
;
;      Routine to display month or day of week
;
SPACE4       PUSH  HL            ;Print 4 SPACEs
      LD    HL,SPACE4$   ;  point to string
      CALL  @DSPLY
      POP   HL
DSPMDY       DEC   B             ;Point to Bth entry
      LD    A,L          ;  in table
      ADD   A,B
      ADD   A,B
      ADD   A,B
      LD    L,A
      LD    B,3          ;Print 3 characters
DSPM1 LD    A,(HL)
```

```
      INC   HL
      CALL  @DSP
      DJNZ  DSPM1
      RET
PARTYR      DB    ', 198',30,3
;
      IF    @INTL
DATEPR      DB    30,'Date DD/MM/YY ? ',3
      ELSE
DATEPR      DB    30,'Date MM/DD/YY ? ',3
      ENDIF
;
TIMEPR      DB    30,'Time HH:MM:SS ? ',3
SPACE4$     DB    '   ',03,03 ;3 (or 4) space string
      DS    32%00       ;Space for future messages
      END
```

SOUND/ASM

```
;SOUND/ASM - LS-DOS 6.2
;
;      Contains IPL, PAUSE, SOUND, and DECHEX routines
;      Will be loaded into lowcore area along with SYSRES
;
;*MOD
SNDPORT     EQU   90H
      ORG   STACK$
      DW    00          ;Stack guard
;
;      Pause routine
;
@PAUSE      PUSH  BC          ;Save the count
;     SRL   B           ;Adjust for WAIT states
;     RR    C
      LD    A,(SFLAG$)  ;If system (FAST)
      BIT   3,A         ;  then double it
      CALL  NZ,CDLOOP   ;Call if FAST
      POP   BC          ;Restore the count
CDLOOP      DEC   BC          ;CountDown Loop
      LD    A,B
      OR    C           ;Loop until C=0
      JR    NZ,CDLOOP   ;  and B=0
      RET               ;Return (or do second loop)
;
;      @SOUND SVC-104 - Operates sound generator
;      B => sound function
;      Bits 0-2 <0-7> = note # (0 highest)
;      Bits 3-7 <0-31> = relative sound duration
;      All registers are preserved except A
;      Z flag set on exit
;      To ensure sound quality, interrupts are disabled
;
@SOUND      PUSH  BC          ;Save registers
      PUSH  HL
      LD    A,B         ;P/u sound data
      AND   7           ;  strip bits 3-7
      RLCA              ;Adjust for 2-byte fields
      LD    HL,SNDTAB   ;  in sound data table,
      LD    C,A         ;  use as LSB of ptr
      LD    A,B         ;Pick up duration data
      LD    B,0         ;Index into tone table
      ADD   HL,BC       ;  to get note-on/off
      LD    C,(HL)            ;P/u note on/off data
      INC   HL
      LD    L,(HL)            ;P/u note duration
      RRCA              ;Rotate sound duration
      RRCA              ;  into bits 0-4
      RRCA
      AND   1FH         ;Strip off sound #
      INC   A           ;Adjust for offset 0
      LD    H,A         ;Set sound counter
      LD    A,(SFLAG$)  ;If fast, double values
      AND   00001000B
      JR    Z,$A1
      SLA   H
      SLA   L
      SLA   C           ;Values * 2
$A1   DI                ;Don't interrupt timing
$A2   PUSH  HL          ;Save note duration
$A3   LD    B,C         ;Play the tone
      LD    A,1         ;Hold output high
      OUT   (SNDPORT),A ;  for count of (B)
      DJNZ  $
      LD    B,C         ;Hold output low
      INC   A           ;Bit 0 is latch bit =>0
      OUT   (SNDPORT),A
      DJNZ  $           ;Countdown (B)
      DEC   L           ;Decrement the duration
      JR    NZ,$A3
      POP   HL          ;Get sound/note durations
      DEC   H           ;Count down the sound
      JR    NZ,$A2            ;  duration counter
      EI                ;Restore interrupts
      POP   HL
      POP   BC          ;Restore regs
      RET
;
;     Note table
;
SNDOFF      EQU   180         ;Sound duration offset
TONER EQU   28
SNDTAB      DB    108-TONER   ;Note 0 (highest)
      DB    0-SNDOFF
      DB    114-TONER
      DB    252-SNDOFF
      DB    120-TONER
      DB    248-SNDOFF
      DB    126-TONER
      DB    244-SNDOFF
      DB    135-TONER
      DB    240-SNDOFF
      DB    142-TONER
      DB    236-SNDOFF
      DB    149-TONER
      DB    232-SNDOFF
      DB    156-TONER   ;Note 7 (lowest)
      DB    228-SNDOFF
SNDLEN      EQU   $-@SOUND
;
;     Process decimal adjustment
;
@DECHEX     LD    BC,0        ;Init value to zero
DEC1  LD    A,(HL)            ;P/u a char
      SUB   30H         ;Convert to binary
      RET   C           ;Return if < "0"
      CP    10          ;Ck for bad decimal
      RET   NC          ;Ret if not 0-9
      PUSH  BC          ;Exchange BC & HL
      EX    (SP),HL           ;  & save HL on stack
      ADD   HL,HL       ;Multiply by 10
      ADD   HL,HL
      ADD   HL,BC
      ADD   HL,HL
      LD    B,0         ;Merge in new digit
      LD    C,A         ;New digit to C
      ADD   HL,BC       ;  & add it in
      LD    B,H         ;Current value to BC
      LD    C,L
      POP   HL          ;Recover HL pointer
      INC   HL
      JR    DEC1        ;Loop
;
;     Special Boot code to be moved to 4300H by @IPL
;
BOOTCOD     DI                ;Boot stub for @IPL
      XOR   A           ;  to move to 4300H
      OUT   (@OPREG),A
      RST   0
BOOTLEN     EQU   $-BOOTCOD
;
      END
```

LOGO/ASM

```
;RSLOGOB/ASM      3-D RS LOGO used on 6.2.0 - 1/20/84
*LIST OFF
      ORG   0F957H
      DEFB  130,175
      DEFS  27%191
      DEFB  159,161,132,144,128,'tm'
      ORG   0F9A9H
      DEFB  139
      DEFS  7%191
      DEFS  11%143
      DEFB  175
      DEFS  6%191
      DEFB  135,152,161,134,152
      DEFB  161,132
      ORG   0F9FAH
      DEFB  130,175
      DEFS  5%191
      DEFS  5%188
      DEFB  128,168
      DEFS  4%188
      DEFB  190
      DEFS  4%191
      DEFB  159,161,134
      DEFB  152,161,134,152,129
      ORG   0FA4CH
      DEFB  139
      DEFS  9%191
      DEFB  128,170
      DEFS  8%191
      DEFB  135,152,161
      DEFB  134,152,161,134
      ORG   0FA9DH
      DEFB  130,175
      DEFS  7%191
      DEFB  128,170
      DEFS  6%191
      DEFB  159,161,134,152,161,134,152
      DEFB  129
      ORG   0FAEFH
      DEFB  139
      DEFS  6%191
      DEFB  176,186
      DEFS  5%191
      DEFB  135,152,161,134,152,161,134
      ORG   0FB40H
      DEFB  130,175
      DEFS  9%191
      DEFB  159,161,134,152,161,134,152
      DEFB  129
      ORG   0FB92H
      DEFB  171
      DEFS  7%143
      DEFB  151,168,129
      DEFB  150,168,129,150
      ORG   0FBE2H
      DEFB  186
      DEFS  7%188
      DEFB  181,138,144,165,138,144,165
      ORG   0FC30H
      DEFB  160,190
      DEFS  9%191
      DEFB  189,146,164,137,146,164,137,144
      ORG   0FC7FH
      DEFB  184,191,191,135
      DEFS  7%131
      DEFB  139,191,191
      DEFB  180,137,146,164,137,146,164
      ORG   0FCCDH
      DEFB  160,190,191,191,129
      DEFB  160,190
      DEFS  5%191
      DEFB  189,176,178,191,191,189,146,164,137
      DEFB  146,164,137,144
      ORG   0FD1CH
      DEFB  184
      DEFS  4%191
      DEFB  128,170
      DEFS  13%191
      DEFB  180,137,146,164
      DEFB  137,146,164
      ORG   0FD6AH
      DEFB  160,190
      DEFS  5%191
      DEFB  180,128
      DEFB  139
      DEFS  5%143
      DEFB  135,128,184
      DEFS  5%191
      DEFB  189,146
      DEFB  164,137,146,164,137,144
      ORG   0FDB9H
      DEFB  184
      DEFS  8%191
      DEFB  189
      DEFS  7%188
      DEFB  190
      DEFS  8%191
      DEFB  180,137,146,164,137,146,132
      ORG   0FE07H
      DEFB  160
      DEFB  190
      DEFS  27%191
      DEFB  189,146,132,129
*LIST ON
      END
□
```

SYS0/EQU

```
;SYS0/EQU - Equates from cross reference of Sysres
      ADISP '<SYS0/EQU>'
;
$A1   EQU   03B7H
$A2   EQU   03B8H
$A3   EQU   03B9H
$CKEOF      EQU   1470H
@$SYS EQU   08F0H
@@1   DEFL  0000H
@@2   DEFL  0000H
@@3   DEFL  0000H
@@4   DEFL  0000H
@ABORT      EQU   1B08H
@ADTSK      EQU   1CDAH
@BANK EQU   0877H
@BKSP EQU   1486H
@BREAK      EQU   196FH
@BYTEIO     EQU   1300H
@CHNIO      EQU   0689H
@CKBRKC     EQU   0553H
@CKDRV      EQU   1993H
@CKEOF      EQU   158FH
@CKTSK      EQU   1CF5H
@CLOSE      EQU   1999H
@CLS  EQU   0545H
@CMNDI      EQU   197EH
@CMNDR      EQU   197BH
@CTL  EQU   0623H
@DATE EQU   07A8H
@DBGHK      EQU   199FH
@DCINIT     EQU   19C0H
@DCRES      EQU   19C4H
@DCSTAT     EQU   19B5H
@DCTBYT     EQU   1A2BH
@DEBUG      EQU   19A0H
@DECHEX     EQU   03E1H
@DIRCYL     EQU   18F7H
@DIRRD      EQU   18BBH
@DIRWR      EQU   1803H
@DIV16      EQU   06E3H
@DIV8 EQU   1927H
@DODIR      EQU   19AFH
@DOKEY      EQU   19A9H
@DSP  EQU   0642H
@DSPLY      EQU   052DH
@ERROR      EQU   1B0FH
@EXIT EQU   1B0BH
@FEXT EQU   1984H
@FLAGS      EQU   196AH
@FNAME      EQU   199CH
@FRENCH     EQU   0000H
@FSPEC      EQU   1981H
@GATRD      EQU   1874H
@GATWR      EQU   1875H
```

```
@GERMAN     EQU   0000H
@GET  EQU   0638H
@GTDCB      EQU   1990H
@GTDCT      EQU   1A1EH
@GTMOD      EQU   19B2H
@HDFMT      EQU   19E4H
@HEX16      EQU   07BDH
@HEX8 EQU   07C2H
@HEXDEC     EQU   06F6H
@HIGH$      EQU   1948H
@HITRD      EQU   1897H
@HITWR      EQU   1898H
@HZ50 EQU   0000H
@ICNFG      EQU   0086H
@INIT EQU   198DH
@INTL EQU   0000H
@IPL  EQU   1BF2H
@JCL  EQU   0630H
@KBD  EQU   0635H
@KEY  EQU   0628H
@KEYIN      EQU   0585H
@KITSK      EQU   0089H
@KLTSK      EQU   1CD0H
@LOAD EQU   1B38H
@LOC  EQU   14B3H
@LOF  EQU   14DEH
@LOGER      EQU   0503H
@LOGOT      EQU   0500H
@MOD2 EQU   0000H
@MOD4 EQU   0FFFFH
@MSG  EQU   0530H
@MUL16      EQU   06C9H
@MUL8 EQU   190AH
@NMI  EQU   0066H
@OPEN EQU   198AH
@OPREG      EQU   0084H
@PARAM      EQU   1987H
@PAUSE      EQU   0382H
@PEOF EQU   14A2H
@POSN EQU   1434H
@PRINT      EQU   0528H
@PRT  EQU   063DH
@PUT  EQU   0645H
@RAMDIR     EQU   19ACH
@RDHDR      EQU   19D8H
@RDSEC      EQU   19F4H
@RDSSC      EQU   18D8H
@RDTRK      EQU   19E0H
@READ EQU   1513H
@REMOVE     EQU   19A6H
@RENAME     EQU   1996H
@REW  EQU   149BH
@RMTSK      EQU   1CD7H
@RPTSK      EQU   1CEBH
@RREAD      EQU   1473H
@RSLCT      EQU   19D4H
@RST00      EQU   0000H
```

```
@RST08      EQU   0008H
@RST10      EQU   0010H
@RST18      EQU   0018H
@RST20      EQU   0020H
@RST28      EQU   0028H
@RST30      EQU   0030H
@RST38      EQU   0038H
@RSTNMI     EQU   0FE9H
@RSTOR      EQU   19C8H
@RSTREG     EQU   0680H
@RUN  EQU   1B1DH
@RWRIT      EQU   13ADH
@SEEK EQU   19D0H
@SEEKSC     EQU   1421H
@SKIP EQU   1430H
@SLCT EQU   19BCH
@SOUND      EQU   0392H
@STEPI      EQU   19CCH
@TIME EQU   078DH
@USA  EQU   0FFFFH
@VDCTL      EQU   0B99H
@VDCTL3     EQU   0D38H
@VER  EQU   1560H
@VRSEC      EQU   19DCH
@WEOF EQU   14ECH
@WHERE      EQU   1979H
@WRITE      EQU   1531H
@WRSEC      EQU   19E8H
@WRSSC      EQU   19ECH
@WRTRK      EQU   19F0H
@_VDCTL     EQU   0D42H
ADDR_2_ROWCOL     EQU   0DF1H
AFLAG$      EQU   006AH
AUTO? EQU   1FF1H
BAR$  EQU   0201H
BOOTST$     EQU   439DH
BREAK?      EQU   1C60H
BRKVEC$     EQU   1C88H
BUR$  EQU   0200H
CASHK$      EQU   0A7BH
CFCB$ EQU   00E0H
CFGFCB$     EQU   00E0H
CFLAG$      EQU   006CH
CKMOD@      EQU   1A7FH
CKOPEN@     EQU   1568H
CONFIG$     EQU   203FH
CORE$ DEFL  1CFFH
CORE$ DEFL  1BFFH
CORE$ DEFL  1948H
CORE$ DEFL  0300H
CRTBGN$     EQU   0F800H
CYL_GRN     EQU   16AEH
D@FBYT8     EQU   1A26H
DATE$ EQU   0033H
DAYTBL$     EQU   04C7H
DBGSV$      EQU   00A0H
DCBKL$      EQU   0031H
```

```
DCT$  EQU   0470H
DCTBYT8@    EQU   1A29H
DCTFLD@     EQU   1A34H
DFLAG$      EQU   006DH
DIRBUF$     EQU   2300H
DIS_DO_RAM  EQU   0846H
DODATA$     EQU   0B94H
DODCB$      EQU   0210H
DO_CONTROL  EQU   0C44H
DO_DSPCHAR  EQU   0CB8H
DO_INVERT_DIS     EQU   0C8CH
DO_INVERT_ENA     EQU   0C89H
DO_INVERT_OFF     EQU   0C9BH
DO_MASK     EQU   0000H
DO_RET      EQU   0BCBH
DO_RETI     EQU   0BCCH
DO_SCROLL   EQU   0CCEH
DO_TABS     EQU   0BEAH
DSKTYP$     EQU   04C0H
DTPMT$      EQU   04C2H
DVREND$     EQU   0FF4H
DVRHI$      EQU   0206H
EFLAG$      EQU   006EH
ENADIS_DO_RAM     EQU   0817H
EXTDBG$     EQU   19A4H
FDDINT$     EQU   000EH
FEMSK$      EQU   006FH
FLGTAB$     EQU   006AH
GET_@_ROWCOL      EQU   0DAEH
HERTZ$      EQU   0750H
HIGH$ EQU   040EH
HKRES$      EQU   1A6CH
IFLAG$      EQU   0072H
INBUF$      EQU   0420H
INTIM$      EQU   003CH
INTMSK$     EQU   003DH
INTVC$      EQU   003EH
JCLCB$      EQU   0203H
JDCB$ EQU   0024H
JFCB$ EQU   00C0H
JLDCB$      EQU   0230H
JRET$ EQU   0026H
KCK@  EQU   07D6H
KFLAG$      EQU   0074H
KIDATA$     EQU   08FCH
KIDCB$      EQU   0208H
LBANK$      EQU   0202H
LDRV$ EQU   0023H
LFLAG$      EQU   0075H
LNKFCB@     EQU   1566H
LOW$  EQU   001EH
LSVC$ EQU   000DH
MAXCOR$     EQU   2400H
MAXDAY$     EQU   0401H
MINCOR$     EQU   3000H
MODOUT$     EQU   0076H
MONTBL$     EQU   04DCH
```

```
NFLAG$      EQU   0077H
OPREG$      EQU   0078H
OPREG_SV_AREA     EQU   086EH
OPREG_SV_PTR      EQU   0835H
ORARET@     EQU   14DCH
OSRLS$      EQU   003BH
OSVER$      EQU   0085H
OVRLY$      EQU   0069H
PAKNAM$     EQU   0410H
PAUSE@      EQU   0382H
PCSAVE$     EQU   07AFH
PDRV$ EQU   001BH
PHIGH$      EQU   001CH
PRDCB$      EQU   0218H
PUTA@DE     EQU   0DCDH
PUT_@ EQU   0DCAH
PUT_@_ROWCOL      EQU   0DC6H
RFLAG$      EQU   007BH
ROWCOL_2_ADDR     EQU   0DD0H
RST38@      EQU   1BFFH
RSTOR$      EQU   04C4H
RWRIT@      EQU   13A2H
S1DCB$      EQU   0238H
SBUFF$      EQU   1D00H
SET@EXEC    EQU   1A79H
SET_SCROLL  EQU   0CF3H
SFCB$ EQU   008CH
SFLAG$      EQU   007CH
SIDCB$      EQU   0220H
SODCB$      EQU   0228H
SPACE4$     EQU   2142H
STACK$      EQU   0380H
START$      EQU   0000H
SVCRET$     EQU   000BH
SVCTAB$     EQU   0100H
SYSERR$     EQU   1B13H
TCB$  EQU   004EH
TFLAG$      EQU   007DH
TIME$ EQU   002DH
TIMER$      EQU   002CH
TIMSL$      EQU   002BH
TIMTSK$     EQU   0713H
TMPMT$      EQU   04C3H
TRACE_INT   EQU   07B1H
TYPHK$      EQU   0A8FH
TYPTSK$     EQU   0B26H
USTOR$      EQU   0013H
VFLAG$      EQU   007FH
WRINT$      EQU   0080H
ZERO$ EQU   0401H
ZEROA@      EQU   13A0H
□
```

SYS1/ASM

```
;SYS1/ASM - LS-DOS 6.2
      ADISP '<SYS1 - LS-DOS 6.2>'
;
LD___A      EQU   3AH         ;LD A,(nnnn)
;
@SMALL      EQU   0           ;Switch for "SMALL" or
                        ;  "FULL" library
;
LIBA  EQU   8000H
LIBB  EQU   0A000H            ;Set bit 5
LIBC  EQU   0C000H            ;Set bit 6
LF    EQU   10
CR    EQU   13
*LIST OFF               ;Get SYS0/EQU
*REF  'SYS0/EQU:1'
*LIST ON
*GET  'COPYCOM:1'       ;Copyright message
;
      ORG   1E00H
;
SYS1  JR    SYS1BGN           ;Hop around pointer
      DW    LIBTBL$           ;LIBTBL pointer
SYS1BGN     AND   70H         ;Strip all but ept
      RET   Z           ;Back on zero entry
      CP    10H         ;Ck for @EXIT
      JR    Z,CMD
      CP    40H         ;Ck for FSPEC
      JP    Z,FSPEC
      CP    50H         ;Ck for FEXT
      JP    Z,FEXT
      CP    60H         ;Ck for PARAM
      JP    Z,PARAM
      CP    70H         ;Ck for vacant entry
      RET   Z
;
;     Entry code for CMNDI (30) and CMNDR (20) SVCs
;
      LD    DE,INBUF$   ;Move 79 characters
      PUSH  DE          ;  from (HL) to buffer
      LD    BC,79
      LDIR
      EX    DE,HL       ;Terminate with ETX
      LD    (HL),3
      POP   HL          ;Recover buffer start
      CP    30H         ;Ck entry for CMNDI
      JR    Z,CMD30           ;Go on CMNDI
      CALL  @CKBRKC           ;Clear the Break bit
      LD    A,(CFLAG$)
      OR    2           ;Set CMNDR bit
      LD    (CFLAG$),A  ;Put it back
      JP    CMD20       ;  & go to CMNDR
;
;     Entry for @EXIT & @CMNDI
;
```

```
CMD30 CALL  CLEANUP           ;Reset Break, stack, etc.
      JR    CMD3A
;
CMD   CALL  CLEANUP           ;Reset Break, stack, etc.
      JR    CMDCONT
;
CLEANUP     EQU   $
      DI                ;Stop for a moment
      LD    HL,0        ;Reset vectored BREAK
      CALL  @BREAK            ;  to system
      POP   HL          ;P/u local RETurn
      LD    SP,STACK$   ;Reset stack pointer
      LD    BC,@EXIT    ;Establish Return addr
      PUSH  BC
      PUSH  HL          ;Put back local return
      LD    A,(SFLAG$)  ;DEGUB to be on or off?
      RLCA
      LD    A,0C9H            ;Bit 7, 1=on, 0=off
      JR    NC,DBGOFF   ;Go if OFF
      XOR   A           ;  else reset to on
DBGOFF      LD    (@DBGHK),A
      LD    HL,KFLAG$   ;Point to KFLAG$
      LD    A,11111001B ;Reset pause and enter
      AND   (HL)        ;Merge together
      LD    (HL),A
      LD    HL,SFLAG$   ;Point to System flag
      LD    A,11111000B ;Reset bits 0-2
      AND   (HL)        ;Merge with old
      LD    (HL),A
      LD    HL,2FFFH    ;Reset LOW$
      LD    (LOW$),HL
;
;     Reset video RAM handler pointer
;
      LD    HL,OPREG_SV_AREA
      LD    (OPREG_SV_PTR),HL
      LD    A,(CFLAG$)  ;P/u CFLAG
      AND   20H         ;Leave only bit 5
      LD    (CFLAG$),A  ;  and put it back
      LD    HL,INBUF$   ;Point to command line
      PUSH  HL          ;Xfer start
      POP   BC          ;  to BC
      EI
      CALL  @CKBRKC           ;Check and clear BREAK
      RET               ;Local cleanup done
;
CMDCONT     LD    A,(EFLAG$)  ;P/u ECI flag
      OR    A           ;Check if set
      JR    Z,CMD1A           ;Go if normal
      OR    10001111B   ;Set for SYS13 but
                        ;  leave user entry code
      RST   28H
;
CMD1A LD    HL,RDYMSG$  ;Display ready message
      CALL  @DSPLY
CMD2  LD    HL,CFLAG$   ;Let the world know we
      SET   2,(HL)            ;  are in the command
      PUSH  HL          ;  interpreter
      LD    HL,INBUF$   ;Get 79 chars max
      LD    BC,79<8           ;No fill char for now
      CALL  @KEYIN
      EX    (SP),HL           ;Turn off the interpreter
      RES   2,(HL)            ;  bit & re-get the buffer
      POP   HL
      JR    C,CMD       ;Jump on <BREAK>
;
;     Entry from @EXIT & @CMNDI
;
CMD3A EQU   $
      LD    A,(HL)            ;Check for comment
      CP    '.'         ;If so go before CR
      JR    Z,CMD20           ;  is displayed
;
      LD    A,CR        ;Do a line feed on
      CALL  @DSP        ;  CMNDI and @EXIT
;
;     Entry from @CMNDR plus the above
;
;     Always bring in bank 0
;
CMD20 XOR   A           ;Prepare for bank0
      LD    B,A         ;Set function and
      LD    C,A         ;  bank number to 0
      CALL  @BANK       ;Invoke bank 0
;
;     Process the command entry
;
      CALL  @LOGER            ;Log the entry
      LD    DE,CFCB$    ;Point to command FCB
      LD    A,(HL)            ;Jump on comment
      CP    '.'
      JR    Z,COMMENT
      CP    '*'         ;Check if alternate CMD
      JR    NZ,CKNOEXC  ;  processor needed
      PUSH  HL
      POP   BC          ;Get buffer in BC
      INC   HL          ;Move HL past '*'
      LD    A,0FFH            ;Set up for SYS13 entry
      RST   28H         ;  # 7, and do it
CKNOEXC     SUB   '!'         ;Test for program force
      JR    NZ,NOEXC
      INC   HL          ;Bump past the '!'
NOEXC LD    (TSTEXC+1),A
      CALL  FSPEC       ;Fetch command spec
      JR    NZ,WHAT           ;Jump on error
      PUSH  HL          ;Save terminator pointer
TSTEXC      LD    A,0         ;Test if prog force
      OR    A
      JR    Z,NOTLIB    ;Jump if starting "!"
      LD    BC,LIBTBL$  ;Pt to tbl of LIB cmds
      CALL  @FNDPRM           ;Check for a match
      JR    Z,CMD4            ;Jump if it is
NOTLIB      LD    HL,DFTEXT   ;Else assume prg file, so
      CALL  FEXT        ;  default 'EXT' to CMD
      POP   HL           ;Rcvr terminator pointer
      LD    A,(CFLAG$)  ;Ck LIB only execution
      AND   10H          ;CFLAG$ bit 4
      JP    Z,@RUN             ;The program else WHAT(?)
;
;     Process non-entry
;
WHAT  LD    HL,-1        ;Set to show abort
      RET
;
;     Process "dot" comment
;
COMMENT     LD    A,(SFLAG$)  ;Ret if <DO> in effect
      BIT   5,A          ;  else get another
      JP    Z,CMD2             ;  input line
      LD    HL,0         ;Set for no error
      RET
;
;     Process LIB command
;
CMD4  POP   HL           ;Rcvr terminator pointer
      LD    A,0C9H             ;Turn off DEBUG
      LD    (@DBGHK),A
      LD    A,D          ;Test bit 7 of high
      RLCA               ;  order LIB address
      PUSH  DE           ;Ret to address of
      RET   NC           ;  vector if bit 7 = 0
      POP   DE
      LD    B,E          ;Else put overlay # in
      RLCA               ;Calculate needed library
      RLCA               ;  by rotating 7-5 into
      ADD   A,84H        ;  2-0 & adding RST base
      RST   28H
;
;     BOOT code brings back the ROM
;
BOOTIT      XOR   A            ;SVC 0 => @IPL
      RST   28H
;
;     LIBRARY look-up table starts here
;
LIBTBL$     EQU   $            ;Start of library table
;
      IF    @SMALL
;
;     Use this table for SMALL (OEM) library
;
; DB 'APPEND'
; DW LIBA!31H
      DB    'ATTRIB'
      DW    LIBB!51H
      DB    'AUTO  '
      DW    LIBB!11H
; DB 'BOOT  '
; DW BOOTIT
; DB 'BUILD '
; DW LIBB!33H
; DB 'CAT   '
; DW LIBA!20H
; DB 'CLS   '
; DW LIBA!24H
      DB    'COPY  '
      DW    LIBA!32H
; DB 'CREATE'
; DW LIBB!13H
      DB    'DATE  '
      DW    LIBB!15H
; DB 'DEBUG '
; DW LIBB!14H
; DB 'DEVICE'
; DW LIBA!61H
      DB    'DIR   '
      DW    LIBA!21H
      DB    'DO    '
      DW    LIBA!91H
; DB 'DUMP  '
; DW LIBB!71H
      DB    'FILTER'
      DW    LIBA!66H
      DB    'FORMS '
      DW    LIBC!0B1H
; DB 'FREE  '
; DW LIBB!22H
; DB 'LIB   '
; DW LIBA!19H
; DB 'LINK  '
; DW LIBA!62H
; DB 'LIST  '
; DW LIBA!41H
; DB 'LOAD  '
; DW LIBA!81H
; DB 'MEMORY'
; DW LIBA!1EH
; DB 'PURGE '
; DW LIBB!72H
      DB    'REMOVE'
      DW    LIBA!18H
; DB 'RENAME'
; DW LIBA!53H
; DB 'RESET '
; DW LIBA!63H
; DB 'ROUTE '
; DW LIBA!64H
; DB 'RUN   '
; DW LIBA!82H
      DB    'SET   '
      DW    LIBA!65H
; DB 'SETCOM'
; DW LIBC!0B2H
; DB 'SETKI '
; DW LIBC!0B3H
; DB 'SPOOL '
; DW LIBC!0A2H
      DB    'SYSGEN'
```

```
        DW      LIBC!1CH
        DB      'SYSTEM'
        DW      LIBC!0A1H
        DB      'TIME  '
        DW      LIBB!16H
; DB 'TOF   '
; DW LIBA!25H
        DB      'VERIFY'
        DW      LIBB!1BH
        DB      0           ;Patch 'K' here for KILL
        DB      'ILL  '
        DW      LIBA!18H
        NOP
;
;
        ELSE
;
;       This table for FULL library
;
        DB      'APPEND'
        DW      LIBA!31H
        DB      'ATTRIB'
        DW      LIBB!51H
        DB      'AUTO  '
        DW      LIBB!11H
        DB      'BOOT  '
        DW      BOOTIT
        DB      'BUILD '
        DW      LIBB!33H
        DB      'CAT   '
        DW      LIBA!20H
        DB      'CLS   '
        DW      LIBA!24H
        DB      'COPY  '
        DW      LIBA!32H
        DB      'CREATE'
        DW      LIBB!13H
        DB      'DATE  '
        DW      LIBB!15H
        DB      'DEBUG '
        DW      LIBB!14H
        DB      'DEVICE'
        DW      LIBA!61H
        DB      'DIR   '
        DW      LIBA!21H
        DB      'DO    '
        DW      LIBA!91H
        DB      'DUMP  '
        DW      LIBB!71H
        DB      'FILTER'
        DW      LIBA!66H
        DB      'FORMS '
        DW      LIBC!0B1H
        DB      'FREE  '
        DW      LIBB!22H
        DB      'LIB   '
        DW      LIBA!19H
```

```
      DB    'LINK  '
      DW    LIBA!62H
      DB    'LIST  '
      DW    LIBA!41H
      DB    'LOAD  '
      DW    LIBA!81H
      DB    'MEMORY'
      DW    LIBA!1EH
      DB    'PURGE '
      DW    LIBB!72H
      DB    'REMOVE'
      DW    LIBA!18H
      DB    'RENAME'
      DW    LIBA!53H
      DB    'RESET '
      DW    LIBA!63H
      DB    'ROUTE '
      DW    LIBA!64H
      DB    'RUN   '
      DW    LIBA!82H
      DB    'SET   '
      DW    LIBA!65H
      DB    'SETCOM'
      DW    LIBC!0B2H
      DB    'SETKI '
      DW    LIBC!0B3H
      DB    'SPOOL '
      DW    LIBC!0A2H
      DB    'SYSGEN'
      DW    LIBC!1CH
      DB    'SYSTEM'
      DW    LIBC!0A1H
      DB    'TIME  '
      DW    LIBB!16H
      DB    'TOF   '
      DW    LIBA!25H
      DB    'VERIFY'
      DW    LIBB!1BH
      DB    0            ;Patch 'K' here for KILL
      DB    'ILL  '
      DW    LIBA!18H
      NOP
;
      ENDIF
;
;
;     Routine to fetch a filespec/devicespec
;
FSPEC PUSH  DE           ;Save pointer to DCB
      CALL  @PARSER            ;Parse expected command
      JR    NZ,FSP5            ;NZ=not file, ck for device
      CP    '/'          ;EXT separator?
      JR    NZ,FSP1
      LD    (DE),A             ;File extent coming,
      INC   DE           ;  get it
      LD    B,3          ;EXT is 3-chars maximum
      CALL  @PAR1
```

```
FSP1  CP    '.'          ;PASSWORD entered?
      JR    NZ,FSP2
      LD    (DE),A             ;Password coming,
      INC   DE           ;  get it also
      CALL  @PARSER
      JR    NZ,FSP6            ;Return if error
FSP2  CP    ':'          ;Drive entered?
      JR    NZ,FSP3
      LD    (DE),A             ;A one-byte drive
      INC   DE           ;  has been had
      LD    B,1
      CALL  @PAR1
      JR    NZ,FSP6            ;Return if error
FSP3  CP    '!'          ;Update EOF always?
      JR    NZ,FSP4
      LD    (DE),A             ;Yes slow but accurate
      INC   DE           ;Incr buffer pointers
      INC   HL
      LD    A,(HL)
FSP4  LD    C,A          ;Save separator char
      LD    A,3
      LD    (DE),A             ;Stuff an ETX
      XOR   A
      LD    A,C          ;P/u separator
      POP   DE           ;P/u start of DCB
      PUSH  DE
      LD    BC,PREPTBL  ;Ck on prepositions
      CALL  @FNDPRM
      POP   DE           ;Can use TO, ON,
      JR    Z,FSPEC            ;  OVER, USING
      XOR   A
      RET                ;Return with Z flag
FSP5  CP    '*'          ;Ck on device spec
      JR    NZ,FSP6            ;Jump if not device
      LD    (DE),A             ;  else stuff the '*'
      INC   DE
      LD    B,2          ;Xfer two char device
      CALL  @PAR1
      JR    Z,FSP4             ;Terminate buffer
FSP6  POP   DE
      RET
;
;     Preposition table
;
PREPTBL     DB    'TO    '
      DW    SBUFF$
      DB    'ON    '
      DW    SBUFF$
      DB    'OVER  '
      DW    SBUFF$
      DB    'USING '
      DW    SBUFF$
      NOP
;
;     Fetch default file extension
;
FEXT  PUSH  DE           ;Save FCB pointer
```

```
      PUSH  HL          ;Save EXT default pointer
      EX    DE,HL       ;Exchange pointers
      INC   HL
      LD    B,9         ;Init for 9-char test
FEX1  LD    A,(HL)            ;Ret if extension start
      CP    '/'         ;  is found
      JR    Z,FEX3
      JR    C,FEX4            ;Jump on other separator
      CP    ':'         ;Jump on digit 0-9
      JR    C,FEX2
      CP    'A'         ;Jump on special char
      JR    C,FEX4
FEX2  INC   HL          ;Advance past A-Z,0-9
      DJNZ  FEX1
FEX3  POP   HL          ;User entered file EXT
      POP   DE          ;FCB start
      RET
;
;     Use default extension
;
FEX4  LD    BC,15       ;Point to position past
      ADD   HL,BC       ;  the filespec
      LD    D,H
      LD    E,L
      INC   DE          ;Make room for '/EXT'
      INC   DE          ;  which is 4 chars
      INC   DE
      INC   DE
      INC   BC          ;Now move 16 bytes
      LDDR
      POP   HL          ;Recover pointer to EXT
      INC   HL          ;Point to 3rd char
      INC   HL
      LD    C,3         ;Move in 3 chars
      LDDR
      LD    A,'/'       ;Put in the slash
      LD    (DE),A
      POP   DE          ;Point back to FCB
      RET
;
;     Get the code for the @PARAM SVC
;
*GET  'PARAM:1'
;
DFTEXT      DB    'CMD'       ;Default extension
      IF    @MOD2
RDYMSG$     DB    LF,14,'LS-DOS Ready',CR
      ELSE
RDYMSG$     DB    LF,14,'TRSDOS Ready',CR
      ENDIF
LAST  EQU   $
      IF    $.GT.DIRBUF$
      ADISP 'ERROR: Module too big'
      ENDIF
      ORG   MAXCOR$-2
      DW    LAST-SYS1   ;Size of overlay
      END   SYS1
```

PARAM/ASM

```
;PARAM/ASM - LS-DOS 6.2
;
;      Parse a field
;      (HL) => command line
;      (DE) => FCB area
;      (HL) <= 1st byte past non-<A-Z, a-z, 0-9>
;             except 13, 3, "("
;      Z   <=  found valid field
;      NZ  <=  found invalid field
;
@PARSER      LD    B,8          ;Set length
@PAR1 LD    A,B
      LD    (PAR6+1),A  ;Stuff length for test
      INC   B
PAR2  LD    A,(HL)
      CP    3           ;ETX
      JR    Z,PAR5
      CP    CR          ;<ENTER>?
      JR    Z,PAR5
      CP    '('         ;Begin of parm?
      JR    Z,PAR5
      INC   HL          ;Bump pointer to next
      CALL  TST09AZ           ;Test if 0-9, A-Z
      JR    NC,PAR3           ;Go if one of the above
      CP    'a'         ;Check on lower case
      JR    C,PAR5            ;Jump on non-alpha
      CP    'z'+1       ;Is it <a-z>
      JR    NC,PAR5           ;Jump on non-alpha
      RES   5,A         ;Convert lower to upper
PAR3  DEC   B           ;Count down
      JR    Z,PAR4
      LD    (DE),A            ;Xfer the char
      XOR   A           ;Show at least 1 valid
      LD    (PAR6+1),A  ;Char was detected
      INC   DE          ;Bump FCB pointer
      JR    PAR2        ;Loop
PAR4  INC   B           ;Here on max chars ck'd
      JR    PAR2
PAR5  LD    C,A         ;Save separator
      LD    A,3         ;Stuff ETX
      LD    (DE),A
;
;      Skip over spaces
;
      LD    A,C         ;Was separator a space?
      CP    ' '
      JR    NZ,PAR6           ;Don't skip if not
PAR5A CP    (HL)        ;Next char a space?
      INC   HL
      JR    Z,PAR5A           ;Loop until not
      DEC   HL          ;Back up to last non-space
;
;      Return status of field validity
;
```

```
PAR6  LD    A,0         ;Set Z flag if at least
      OR    A           ;  1 valid char detected
      LD    A,C         ;Recover separator char
      RET
;
;     Test if 0-9 of A-Z
;
TST09AZ     CP    '0'         ;Special character?
      RET   C           ;Go if not in range
      CP    '9'+1       ;Jump on digit 0-9
      JR    C,EXITC           ;Go if 0-9 & make NC
      CP    'A'         ;Jump on special char
      RET   C           ;Go it 3B-40
      CP    'Z'+1       ;Jump on A-Z
EXITC CCF               ;Switch flag of result
      RET
;
;     Find parameter in table
;     (HL) => pointer to line
;     (DE) => pointer to buffer area
;     (BC) => pointer to parameter table
;     (BC) <= pointer to possible response byte
;     (DE) <= returns parameter vector address
;       Z <= set if found
;      NZ <= if NOT FOUND in table
;
@FNDPRM     PUSH  HL
      LD    H,B         ;Xfer table addr
      LD    L,C
      LD    A,(HL)            ;P/u 1st byte of table
      RLCA              ;  & test for enhanced
      PUSH  AF          ;  table format
      JR    NC,FND1
      INC   HL          ;Bump past indicator
FND1  POP   AF          ;Old or enhanced format?
      PUSH  AF
      LD    A,5         ;Init for old lengths
      LD    BC,2!(1<8)
      JR    NC,FND1A    ;Branch if old format
      LD    A,(HL)            ;  else get parm length
      AND   0FH         ;Strip flags
      DEC   A           ;Adjust for length-1
      INC   B           ;Update offset to address
      INC   HL          ;Bump past TYPE byte
FND1A LD    (FND3A+1),A ;Stuff the lengths
      ADD   A,B
      LD    (FND5A+1),A
      ADD   A,C
      LD    (FND2+1),A
      LD    A,(DE)            ;P/u command line byte
      CP    (HL)        ;Match 1st char of table?
      JR    Z,FND3            ;Jump if 1st char matches
FND2  LD    BC,8        ;  else bypass that entry
      ADD   HL,BC
      LD    A,(HL)            ;Test for table end
      OR    A
      JR    NZ,FND1           ;Loop if more
      POP   HL          ;Clean flag from stack
      POP   HL          ;Recover saved reg &
      INC   A           ;  set NZ for not found
      RET
FND3  POP   AF          ;Ck old or new table
      PUSH  AF
      JR    NC,FND3A    ;Go if old format table
      DEC   HL          ;Ck if type byte permits
      BIT   4,(HL)            ;  single char abbrev
      INC   HL
      JR    Z,FND3A           ;Go on no abbrev
      INC   DE          ;Make sure the next char
      LD    A,(DE)            ;  is not in the range
      DEC   DE          ;  <0-9,A-Z> before
      CALL  TST09AZ           ;  assuming abbrev
      JR    C,FND5A           ;Go on 1-char abbrevs
FND3A LD    B,5         ;5 more chars to match
      PUSH  HL
      PUSH  DE
      LD    A,B         ;Don't if trailing length
      OR    A           ;  is zero
      JR    Z,FND5
FND4  INC   DE
      INC   HL
      LD    A,(DE)
      CP    3           ;ETX?
      JR    Z,FND7
      CP    CR          ;Jump on <ENTER>
      JR    Z,FND7
      CP    (HL)        ;Match?
      JR    NZ,FND6           ;Jump if not
      DJNZ  FND4        ;  else loop
FND5  POP   DE          ;Parm matched
      POP   HL          ;Recover begin of parm
FND5A LD    BC,6        ;Point to address field
      ADD   HL,BC
      LD    C,L         ;Save the response-byte
      LD    B,H         ;  pointer in BC
      DEC   BC
      LD    E,(HL)            ;P/u parm table address
      INC   HL
      LD    D,(HL)
      POP   AF          ;If not enhanced, change
      JR    C,$+4       ;  pointer to bucket
      LD    B,SBUFF$>8  ;  so we don't alter user
      POP   HL          ;Recover line position
      XOR   A           ;Show found
      RET
FND6  CALL  TST09AZ           ;Ck if 0-9, A-Z
      JR    NC,FND8           ;Go if in the range of above
FND7  LD    A,(HL)            ;Loop if table has
      CP    ' '         ;  trailing spaces
      JR    Z,FND5
FND8  POP   DE
      POP   HL
      JR    FND2
;
```

```
;     PARAM routine
;      (HL) => param line
;      (DE) => parm table
;      (DE) <= returns table address value
;         C <= # of parm
;         Z = Okay
;        NZ = Parm Error
;
PARAM0      INC   HL          ;Bump the pointer
PARAM LD    A,(HL)            ;  and P/u char
      CP    CR
      RET   Z           ;Return on <ENTER>
      CP    ' '
      JR    Z,PARAM0    ;Loop on space
      CP    '('
      JR    NZ,PARAM5   ;Jump if not left parenthesis
      LD    A,(DE)            ;Check if enhanced table
      RLCA
      JR    NC,PARAM1
      PUSH  DE          ;Save pointer to start
      INC   DE          ;Point to 1st TYPE byte
      PUSH  HL          ;Save this position
;
$?1   LD    A,(DE)            ;P/u TYPE byte
      AND   0FH
      JR    Z,$?2       ;Exit on end of table
      LD    L,A         ;Point to response byte
      LD    H,0
      INC   L
      ADD   HL,DE
      LD    (HL),0            ;Zero the response
      INC   HL          ;Bump to the next TYPE
      INC   HL
      INC   HL
      EX    DE,HL       ;Table pointer back to DE
      JR    $?1         ;Loop thru all response bytes
;
$?2   POP   HL          ;Recover reg
      POP   DE          ;  & start of parm table
PARAM1      PUSH  DE
      LD    B,15        ;Maximum 15-character field
      LD    DE,SBUFF$   ;Point to buffer region
      INC   HL          ;Bypass the '('
      CALL  @PAR1       ;Get the field
      DEC   HL          ;Back up to separator
      POP   DE
      JR    NZ,ERROUT   ;Return if bad field
      CP    CR          ;If separator was a CR,
      JR    NZ,$+3            ;  we need to counteract
      INC   HL          ;  the DEC HL above
      PUSH  DE
      LD    B,D         ;Table pointer to BC
      LD    C,E
      LD    DE,SBUFF$   ;Parm in table?
      CALL  @FNDPRM
      PUSH  BC          ;Save response pointer
      JR    Z,PARAM3    ;Jump if found in table
;
;     Parameter not in table - NZ condition
;
PARAM2      POP   DE          ;Pop response pointer
      POP   DE          ;Pop parm table pointer
ERROUT      LD    A,44        ;Set up PARM ERROR
      RET
;
;     Parameter found in table - parse the value
;
PARAM3      LD    A,(HL)            ;Test for assignment
      CP    '='
      JR    Z,ASSIGN    ;Jump if parm=value
      LD    BC,-1       ;  else set symbol TRUE
PARMSW      EX    (SP),HL           ;Get response byte
      SET   6,(HL)            ;Turn on FLAG-SWITCH
;
;     Valid parm argument parsed into reg BC
;
PARAM4      EX    DE,HL       ;Address pointer to HL
      LD    (HL),C            ;Stuff low-order value
      INC   HL
      LD    (HL),B            ;Stuff high-order value
      POP   HL          ;Recover parm line ptr
      POP   DE          ;Recover parm table ptr
      LD    A,(HL)
      CP    ','         ;Comma separator?
      JR    Z,PARAM1
      CP    CR
      JR    Z,PARAM5
      CP    ')'         ;Closing parenthesis?
      JR    NZ,ERROUT   ;No, leave with ERROR
      INC   HL          ;Bump line pointer
PARAM5      XOR   A           ;Show all Okay
      RET
;
;     Parameter assignment statement
;
ASSIGN      INC   HL          ;Advance token past '='
      LD    A,(HL)
      CP    '"'         ;Double quote string?
      JR    Z,STRING
      CP    'A'         ;Ck on digit or
      JR    C,ASS3            ;  special character
      RES   5,A         ;Strip if lower case
      CP    'X'         ;Hexadecimal?
      JR    Z,ASS1
      CALL  ONOFF       ;Ck on Y, N, ON, OFF
      JR    Z,PARMSW    ;Set FLAG-SWITCH if Okay
      JR    PARAM2            ;  else error exit
ASS1  INC   HL
      CALL  HEXVAL            ;Ck on hex format
      JR    NZ,PARAM2   ;Error if bad format
      JR    ASS3A       ;  else bypass & set resp
;
;     Which is the parameter, numeric or flag?
;
```

```
ASS3  CP    '0'         ;Parameter=number ?
      PUSH  AF          ;CF = 0 if number
      CALL  @DECHEX           ;Cvt # @ HL to bin in DE
      POP   AF
ASS3A EX    (SP),HL           ;Get response pointer
      JR    NC,ASS4           ;Show numeric if CF=0
      SET   6,(HL)            ;  otherwise show switch
      DB    LD___A            ;Skip next instruction
ASS4  SET   7,(HL)            ;Set Numeric Response bit
      JR    PARAM4
;
;     Parameter string entry
;
STRING      INC   HL          ;Bypass '"'
      LD    B,H         ;Save starting address
      LD    C,L
STR1  LD    A,(HL)            ;P/u a char
      CP    20H
      JR    C,PARAM2    ;Exit on control char
      INC   HL          ;Bump pointer
      CP    '"'         ;Closing double quote
      JR    NZ,STR1
      PUSH  HL          ;Save current pointer
      SBC   HL,BC       ;Calculate length of str
      LD    A,L
      DEC   A           ;Adjust for INC HL
      CP    32          ;If len > 31, set to 0
      JR    C,$+3
      XOR   A
      POP   HL          ;Recover pointer
      EX    (SP),HL           ;Get response byte
      OR    20H         ;Set FLAG-STRING
      LD    (HL),A
      JR    PARAM4
;
;     Check for YES, NO, ON, OFF switches
;
ONOFF LD    BC,0        ;Init to FALSE
      SUB   'Y'         ;Is it yes?
      JR    Z,ONO1            ;Jump on YES
      ADD   A,'Y'-'N'   ;Is it no?
      JR    Z,ONO2            ;Jump on NO
      DEC   A           ;Is it 'O'n or 'O'ff?
      RET   NZ          ;Return if not on/off
      INC   HL          ;Bump pointer to next
      LD    A,(HL)            ;  character & p/u
      RES   5,A         ;Set l/c to Upper case
      CP    'F'
      JR    Z,ONO2            ;Jump on off
      CP    'N'
      RET   NZ          ;Return if neither
ONO1  LD    BC,-1       ;Init to true
ONO2  INC   HL          ;Ignore the trailing part
      LD    A,(HL)            ;  of word until closing
      CP    ')'         ;  ")" or comma separator
      RET   Z           ;  or CR
      CP    CR
      RET   Z
      CP    ','
      RET   Z
      JR    ONO2
;
;     Process hexadecimal assignment
;
HEXVAL      LD    BC,0        ;Init value to zero
      LD    A,(HL)            ;P/u a char
      CP    '&'+1       ;Must be single quote ("'")
      RET   NZ          ;Return if not
HEX1  INC   HL          ;Bump past it
      LD    A,(HL)            ;P/u possible hex digit
      SUB   30H         ;Begin conversion
      JR    C,HEX2            ;Jump if < "0"
      CP    10          ;Ck for 0-9
      JR    C,HEX3            ;Jump if digit is 0-9
      RES   5,A         ;Strip l/c if present
      SUB   7           ;else ck A-F
      CP    16
      JR    C,HEX3            ;Jump if A-F
HEX2  LD    A,(HL)            ;Test for closing quote
      CP    '&'+1       ;Compare to "'"
      INC   HL          ;Bump pointer
      RET   Z           ;Ret if closing quote
      DEC   HL          ;  else backup, set OK,
      XOR   A           ;  then return
      RET
HEX3  PUSH  BC          ;Exchange BC & HL
      EX    (SP),HL           ;  and save HL
      ADD   HL,HL       ;Multiply by 16
      ADD   HL,HL
      ADD   HL,HL
      ADD   HL,HL
      LD    B,H         ;Merge new digit
      ADD   A,L
      LD    C,A
      POP   HL          ;Recover pointer
      JR    HEX1        ;Loop
      END
```
□

SYS2/ASM

```
;SYS2/ASM - LS-DOS 6.2
      ADISP '<SYS2 - LS-DOS 6.2>'
;
; This SYS module performs the following functions:
; . OPENs an exitsting File or Device
; . INITs a new file
; . Checks availability of a specific drive
; . Hashes an 11-byte field (file name & ext)
; . Hashes an 8-byte field (password)
; . Renames a filespec/devspec
; . Gets the address of a Device Control Block
;
CR    EQU   13
*LIST OFF               ;Get SYS0/EQU
*REF  'SYS0/EQU:1'
*LIST ON
*GET  'COPYCOM:1'       ;Copyright message
;
      ORG   1E00H
;
SYS2  AND   70H         ;Strip all but entry
      RET   Z           ;Back on zero entry
      CP    10H         ;Check for OPEN
      JP    Z,OPEN
      CP    20H         ;Check for INIT
      JP    Z,INIT
      CP    70H         ;Check for rename
      JP    Z,RENAME
      CP    30H         ;Get a DCB?
      JR    Z,GTDCB
      CP    40H         ;Drive availability?
      JR    Z,CKDRV
      CP    60H         ;Check password hash
      JR    Z,HASHPSWD
;
;     Routine to hash a file name
;
HASHNAME    EQU   $
      LD    B,11        ;Init for 11 chars
      XOR   A           ;Clear for start
HNAME1      XOR   (HL)        ;Modulo 2 addition
      INC   HL          ;Bump to next character
      RLCA              ;Rotate bit structure
      DJNZ  HNAME1            ;  & loop for field len
      OR    A           ;Do not permit a zero
      JR    NZ,HNAME2   ;  hash code
      INC   A
HNAME2      LD    (FILEHASH),A      ;Stuff code for later
      RET
;
;     Hash a password
;
HASHPSWD    EQU   $
      LD    HL,7        ;Hashing will be from
      ADD   HL,DE       ;  right to left so
      EX    DE,HL       ;  point to low-order
      LD    HL,-1       ;Init shift reg to 1's
      LD    B,8         ;Init for 8-char string
HPSWD1      LD    A,(DE)            ;P/u the next byte
      PUSH  DE          ;  & save the pointer
      LD    D,A
      LD    E,H
      LD    A,L         ;Modulo 2 add bits 0-2
      AND   7           ;  to bits 4-6 of the
      RRCA              ;  16-bit shift register
      RRCA
      RRCA
      XOR   L
      LD    L,A         ;Shift shift-regitser
      LD    H,0         ;  left by 4-bits to
      ADD   HL,HL       ;  isolate bits 4-7
      ADD   HL,HL
      ADD   HL,HL
      ADD   HL,HL
      XOR   H           ;Mod 2 add SR bits 4-7
      XOR   D           ;Mod 2 add new byte
      LD    D,A         ;Save tempy for high-order
      LD    A,L
      ADD   HL,HL
      XOR   H
      XOR   E
      LD    E,A
      EX    DE,HL       ;SR result to HL
      POP   DE          ;P/u pointer to string
      DEC   DE          ;  & point to next byte
      DJNZ  HPSWD1            ;Loop for field length
      XOR   A           ;Set Z
      RET
;
;     Routine to locate a Device Control Block
;
GETDCB      LD    E,(IX+1)    ;P/u the 2-character
      LD    D,(IX+2)    ;  device name
GTDCB LD    HL,KIDCB$   ;Point to 1st DCB
DEV1  PUSH  HL
      LD    A,L         ;Point to device
      ADD   A,6         ;  name field
      LD    L,A
      LD    A,(HL)            ;P/u 1st char of name
      INC   L           ;Point to 2nd char
      CP    E           ;Compare 1st for match
      JR    NZ,DEV2           ;No match? then loop
      LD    A,(HL)            ;1st matches, does 2nd?
      CP    D
      JR    NZ,DEV2           ;Loop if no match
      POP   HL          ;Get start of DCB
      RET
DEV2  POP   AF          ;Pop last DCB start
      INC   L           ;Inc to start of next DCB
      JR    NZ,DEV1           ;Bypass if not at end
;
```

```
;      Device not found in tables
;
      LD    A,8         ;"device not available"
      OR    A
      RET
;
;      Check a drive for availability
;
CKDRV PUSH  IY          ;We use IY in Disk I/O
      CALL  @GTDCT            ;Get driver routine addr
      LD    A,(IY+0)    ;P/u drive vector
      CP    0C3H        ;Ck for enabled
      JP    NZ,CKDRV5   ;Bypass if disabled
      PUSH  HL
      PUSH  DE
      BIT   3,(IY+3)    ;Test for HARD drive
      JR    NZ,CKDRV1A  ;If so bypass range check
      LD    A,(IY+6)    ;Make sure the current
      CP    (IY+5)            ;  cylinder is in range
      JR    NC,CKDRV1   ;Go if in range
      CALL  @RSTOR            ;Restore drive
      JP    NZ,CKDR7A   ;Go if error
;
CKDRV1      LD    D,(IY+5)    ;P/u current track
      LD    E,0         ;Set for sector 0
      CALL  @SEEK       ;Send track info to FDC
      JR    NZ,CKDR7A   ;Go if error
CKDRV1A     CALL  @RSLCT            ;Wait until not busy
      JR    NZ,CKDR7A   ;Not there - ret NZ
      BIT   3,(IY+3)    ;If hard drive, bypass
      JR    NZ,CKDR3A   ;  GAT data update
      BIT   4,(IY+4)    ;If "ALIEN" bypass
      JR    NZ,CKDR2B   ;  test of index pulses
      IF    @MOD4
      LD    A,(FDDINT$) ;Check 'SMOOTH' state
      OR    A
      LD    A,09        ;Set MSB of countdown
      JR    Z,INTRON    ;Go if not SMOOTH
      SRL   A           ;Divide the count by two
      DI
      ENDIF
      IF    @MOD2
      LD    A,20
      ENDIF
INTRON      LD    (CDCNT+1),A ;Store in 'LD H' instruction
      LD    HL,0020H    ;Set up count (short)
;
;      Test for diskette in drive and rotating
;
CKDR1 CALL  INDEX       ;Test index pulse
      JR    NZ,CKDR1    ;Jump on index
      BIT   7,(IY+4)    ;Check CKDRV inhibit bit
      JR    NZ,CKDR2B   ;If on skip index test
CDCNT LD    H,00H       ;CKDRV counter (long)
                        ;Count set from above
CKDR2 CALL  INDEX       ;Test index pulse
      JR    Z,CKDR2           ;Jump on no index
      IF    @MOD4
      EI                ;Okay for INTs now
      ENDIF
      LD    HL,0020H    ;Index off wait (short)
CKDR2A      CALL  INDEX
      JR    NZ,CKDR2A   ;Jump on index
;
;     Diskette is rotating
;
CKDR2B      PUSH  AF          ;Save FDC status
      CALL  @DIRCYL           ;Get directory track in D
      LD    HL,SBUFF$   ;Point to HIT buffer
      LD    E,L         ;Sector 0 for GAT
      CALL  @RDSSC            ;Read the GAT
      JR    NZ,CKDR7    ;Jump on error
      LD    HL,(SBUFF$+0CCH)  ;P/u excess tracks
      LD    A,22H       ;Add offset
      ADD   A,L
      LD    (IY+6),A    ;Max track # to DCT
      RES   5,(IY+4)    ;Set to side 0
      BIT   5,H         ;Test double sided
      JR    Z,CKDR3           ;Jump if only single
      SET   5,(IY+4)    ;Set for side 2
CKDR3 POP   AF          ;Recover FDC status
CKDR3A      RLCA              ;Shift write prot to 7
      OR    (IY+3)            ;Merge Software WP bit
      AND   10000000B   ;Strip all but bit 7
      LD    (OPNCB9+1),A      ;Save WP status for OPNCB
      ADD   A,A         ;Write protect to C flag
;
CKDR4 EQU   $
      EI
      POP   DE
      POP   HL
CKDRV5      POP   IY
      RET
INDEX LD    A,H
      OR    L
      JR    Z,CKDR7
      DEC   HL
      CALL  @RSLCT            ;Check for index pulse
      BIT   1,A         ;Test index
      RET
CKDR7 POP   AF
;
CKDR7A      OR    A           ;Set NZ ret
      JR    CKDR4       ;  and exit
;
;     OPEN a device
;     Device Control Blocks are from X'0208' - X'02FF'
;
DEVOPEN     CALL  GETDCB            ;Find the DCB named
      RET   NZ          ;  in the IX pointer
;
;     Found the needed Device Control Block
;
DEV4  LD    B,H         ;Xfer dcb vector to BC
```

```
      LD    C,L
      PUSH  IX          ;User DCB to HL
      POP   HL
      LD    (HL),10H    ;Show routed
      INC   HL
      LD    (HL),C            ;Stuff dcb vector
      INC   HL
      LD    (HL),B
      INC   HL
      XOR   A           ;Zero next 3 bytes
      LD    (HL),A
      INC   HL
      LD    (HL),A
      INC   HL
      LD    (HL),A
      INC   HL
      LD    (HL),E            ;Stuff dcb name
      INC   HL
      LD    (HL),D
      RET
;
;
;     OPEN a file
;      HL => the address of a 256-byte buffer
;      DE => the address of a 32-byte FCB
;       B => the logical record length (LREC)
;
OPEN  CALL  LNKFCB@           ;Set up link to DCB
OPEN1 LD    A,(SFLAG$)  ;Stuff current sysflag
      LD    (OPEN14+1),A      ;  to chack later then
      AND   11111000B   ;  remove bits 0,1,2
      LD    (SFLAG$),A
      LD    A,(IX+0)
      CP    '*'         ;If name starts with '*'
      JR    Z,DEVOPEN   ;  it is a device spec
      LD    A,B         ;P/u LRL requested
      LD    (LREC$),A
      LD    (OPNCB4+1),HL     ;Stuff disk I/O buffer
      PUSH  IX          ;Transfer the filespec
      POP   HL          ;  into the system
      CALL  XFRSPEC           ;  buffer area
      RET   NZ          ;Return if bad name
      LD    HL,NAME$EXT ;Point to name/ext field
      CALL  HASHNAME    ;  & hash it (11 chars)
      LD    DE,PSWDBUF  ;Point to the password
      CALL  HASHPSWD    ;  & hash it
      LD    (PW$HASH1),HL     ;Stuff owner password
      LD    (PW$HASH2),HL     ;Stuff user pasword
OPEN2 LD    A,0         ;P/u drive <FF-07>
      LD    C,A
      INC   A           ;Jump if :dr entered
      JR    NZ,OPEN3
      LD    C,A
OPEN3 CALL  CKDRV       ;Drive available?
      JR    NZ,OPEN6    ;Jump if not
      CALL  @HITRD            ;Get hash index table
      RET   NZ          ;Return if read error
```

```
;
;     Compare hashed filename/ext with each entry
;     in the HIT to see if file is on this drive
;
OPEN4 LD    A,(HL)            ;Bypass HIT entry if
      OR    A           ;  unused
      JR    Z,OPEN5
      PUSH  HL          ;Not vacant
      LD    HL,FILEHASH ;Point to DEC
      CP    (HL)        ;Compare with HIT entry
      POP   HL
      JR    Z,OPEN9           ;Jump if a match else
OPEN5 INC   L           ;  bump to next entry
      JR    NZ,OPEN4    ;Loop until 256 bytes
;
;     File not on this drive
;
OPEN6 CALL  TESTDRV           ;Bump drive if we can
      JR    C,OPEN3           ;Loop if another to test
OPEN7 LD    A,24        ;File not found error
      OR    A           ;Set NZ
      RET
TESTDRV     LD    A,(OPEN2+1) ;If drive still X'FF',
      INC   A           ;  then advance to next
      OR    A           ;Reset Carry for ret w/o
      RET   NZ          ;  affecting Z/NZ result
      INC   C           ;Bump drive counter
      LD    A,C
      CP    8           ;Loop end, 8 DRIVES MAXIMUM
      RET
;
;     Although the HIT entry matched, the filename/ext
;     did not (due to a collision). Continue to scan
;     the rest of the Hash Index Table.
;
OPEN8 POP   BC          ;Remove ret address and
      POP   HL          ;  excess registers
      POP   BC
      CALL  @HITRD            ;Re-read the HIT
      POP   HL
      RET   NZ          ;Go on I/O Error
      JR    OPEN5
;
;     The hashed name matches, read the directory
;
OPEN9 PUSH  HL
      PUSH  BC
      LD    B,L         ;Set up the Directory
      CALL  @DIRRD            ;  Entry Code
      JR    Z,OPEN10    ;Jump if no error
      POP   BC          ;  else pop returns
      POP   HL
      RET               ;  & exit NZ
;
;     Verify that directory entry is this file
;
OPEN10      PUSH  HL
```

```
      PUSH  BC          ;Save drive (reg C)
;
;     If bit 7 is set, in denotes an extended
;     directory entry which does not include
;     the filename. Go to the next HIT entry if set
;
      BIT   7,(HL)            ;Test for FXDE
      JR    NZ,OPEN8    ;Jump if extended
      BIT   4,(HL)            ;If DIR record spare,
      JR    Z,OPEN8           ;  continue to search
      LD    A,5         ;Point to filename/ext
      ADD   A,L         ;  field in directory
      LD    L,A
      LD    DE,NAME$EXT ;Point to entered name
      LD    B,11        ;Init to check 11 chars
OPEN11      LD    A,(DE)            ;Verify a match
      CP    (HL)        ;  or no match
      JR    NZ,OPEN8    ;Go to next HIT entry
      INC   HL          ;  if no match; else bump
      INC   DE          ;  pointers & loop
      DJNZ  OPEN11
      POP   BC          ;Matches! get drive #
      LD    A,C         ;  & stuff it
      LD    (OPEN2+1),A
      POP   HL
      POP   AF
      POP   AF
      PUSH  BC          ;Save DEC and drive
      PUSH  HL          ;Save ptr to dir record
      LD    A,(HL)            ;P/u 1st byte of dir rec
      LD    (DIR$INIT),A      ;Stuff it
      AND   00000111B   ;Strip all but protection
      LD    C,A
      LD    B,0
      LD    A,16        ;Point to update password
      ADD   A,L
      LD    L,A
      LD    DE,(PW$HASH2)     ;P/u password hash
      LD    A,(HL)            ;P/u owner pswd low-order
      INC   HL
      PUSH  HL
      LD    H,(HL)            ;P/u owner pswd high-order
      LD    L,A
      LD    A,(NFLAG$)  ;P/u NFLAG$
      BIT   7,A         ;Check network active bit
      JR    Z,USEPWD
      LD    D,H
      LD    E,L
USEPWD      XOR   A           ;Compare password entry
      SBC   HL,DE       ;  with owner password
      POP   HL
WASMAT      JR    Z,OPEN16    ;Grant access if match
      LD    A,C         ;Recover protection
      CP    7           ;Abort if "no access"
      JR    Z,OPEN12
      INC   HL          ;  else point to user
      LD    B,C         ;  password & Xfer prot lvl
      LD    A,(HL)            ;P/u user pswd low-order
      INC   HL
      LD    H,(HL)            ;P/u user pswd high-order
      LD    L,A
      XOR   A           ;Check for a match
      SBC   HL,DE
      JR    Z,OPEN13    ;Jump if match
;
;     File is password protected - abort
;
OPEN12      POP   HL
      POP   BC
      LD    A,25        ;"file access denied due to...
      OR    A           ;Set NZ for error
      RET
;
;     Check if prot is EXECute only
;
OPEN13      LD    A,C
      CP    6           ;Check for EXEC ONLY
      JR    NZ,OPEN16   ;Jump if not
OPEN14      LD    B,0         ;P/u SFLAG$ entry state
      BIT   2,B         ;Did RUN request open?
      JR    Z,OPEN15    ;Bypass if not from RUN
      LD    HL,SFLAG$
      SET   1,(HL)            ;Show RUN & EXEC file
      LD    A,5         ;Set READ access for now
OPEN15      LD    HL,SET@EXEC ;Set RST vector to turn
      LD    (HL),0C9H   ;  off DEBUG
OPEN16      LD    (OPNCB1+1),A      ;Stuff access level
      POP   HL          ;Ptr to direc record
      POP   BC          ;P/u DEC and drive
;
;     Routine to open up the FCB from the directory
;     HL => directory record in SBUFF$
;     BC => DEC and drive used for directory read/write
;     IX => pointer to File Control Block
;
OPNCB PUSH  IY          ;Save IY
      PUSH  HL          ;Transfer direc record
      POP   IY          ;  ptr to IY
      PUSH  BC          ;Save DEC and drive
      CALL  OPNCB0            ;Create the opened FCB
      POP   BC
      LD    HL,OPEN14+1 ;If from LOAD, don't do
      BIT   0,(HL)            ;  any further checks
      JR    Z,OPNEX1
      XOR   A
OPNEX POP   IY
      RET
OPNEX1      BIT   5,(IY+1)    ;If file already open
      JR    Z,OPNCB8    ;  then set read-only
      POP   IY          ;  & return "file open...
OPNEX2      LD    A,(IX+1)    ;P/u current attributes
      AND   11111000B   ;Mask off current prot
      OR    5           ;  & replace with READ
      LD    (IX+1),A    ;Reset acces to READ
      LD    A,41        ;Set "file already open"
      RET
;
;     If access level is > READ, set file open flag in
;     the directory & note close authority in the FCB
;
OPNCB8      LD    A,(IX+1)    ;P/u FCB access level
      AND   00000111B   ;Mask off other junk
      CP    5           ;Ck READ, EXEC, NONE
      JR    NC,OPNCB10  ;Go if one of the above
OPNCB9      LD    A,0         ;P/u CKDRV status
      RLCA              ;Was drive write prot?
      JR    C,FRCREAD   ;C flag = Wr Prot
      SET   5,(IY+1)    ;Set file open in direc
      LD    A,(NFLAG$)  ;P/u Network flag
      BIT   0,A         ;Check for function ON
      CALL  NZ,@DIRWR   ;Write the directory
      JR    NZ,OPNEX
      SET   6,(IX+0)    ;Set close authority
;
;     Check if passed LRL matches directory
;
OPNCB10     LD    A,(IX+9)    ;P/u LRL from FCB
      CP    (IY+4)            ;  compare with directory
      LD    A,42        ;Init "LRL open fault
      JR    OPNEX
;
;     Disk write protected - Change access to READ
;
FRCREAD     CALL  OPNEX2            ;Change access to READ
      JR    OPNCB10
;
;     This routine creates the open file control block
;
OPNCB0      EX    DE,HL
      PUSH  IX          ;Transfer FCB pointer
      POP   HL
      LD    A,(DE)            ;Get DIR+0
      AND   00100000B   ;Keep "PDS" bit & show
      OR    10000000B   ;  FCB as open
      LD    (HL),A            ;Shove into FCB+0
      INC   HL
      LD    A,(LREC$)   ;P/u LRL
      OR    A           ;Test for 0 (is 256)
OPNCB1      LD    A,0         ;Now start byte 2 with
      JR    Z,OPNCB2    ;  that set by "OPEN16"
      OR    10000000B   ;Show sector or byte I/O
OPNCB2      OR    00100000B   ;Show buffer is empty
;
;     Set bit 3 if filespec ended in an
;     exclamation point. This causes the
;     directory to be updated on EVERY
;     file write where the EOF is extended
;
OPNCB3      OR    0
      LD    (HL),A            ;Init FCB+1
      INC   HL
      XOR   A
      LD    (HL),A            ;Init FCB+2 with 0
      INC   HL
      PUSH  DE          ;Put address of disk I/O
OPNCB4      LD    DE,0        ;  buf into FCB+3 & FCB+4
      LD    (HL),E
      INC   HL
      LD    (HL),D
      INC   HL
      POP   DE          ;FCB+5 with 0 for
      LD    (HL),A            ;  low order next
      INC   HL
      LD    (HL),C            ;FCB+6 with drive
      INC   HL
      LD    (HL),B            ;FCB+7 with DEC
      INC   HL
      INC   DE          ;Point to DIR EOF byte
      INC   DE
      INC   DE
      LD    A,(DE)            ;P/u DIR low order EOF
      LD    (HL),A            ;  & stuff into FCB+8
      INC   HL
      INC   DE
      LD    A,(LREC$)   ;P/u LRL & stuff
      LD    (HL),A            ;  into FCB+9
      INC   HL
      XOR   A
      LD    (HL),A            ;Init FCB+10 & FCB+11
      INC   HL          ;  with zero for NRN
      LD    (HL),A
      INC   HL
      SET   4,E         ;Point to file EOF
      LD    BC,2        ;Move ERN
      EX    DE,HL
      LDIR              ;  and zero BC reg
      EX    DE,HL
      LD    A,5         ;Max 5 extents
      PUSH  AF
OPNCB5      LD    A,(DE)            ;Move starting track
      LD    (HL),A
      INC   HL
      INC   DE
      LD    A,(DE)            ;Move grans & offset
      LD    (HL),A
      INC   HL
      AND   00011111B   ;Strip out grans
      INC   A           ;Bump for 0 offset
;
;     Add reg A to reg pair BC
;
      ADD   A,C         ;Add previous count
      LD    C,A         ;Update C
      JR    NC,$+3            ;Go if no carry to B
      INC   B
      POP   AF          ;Recover counter
      DEC   A           ;Decrement loop
      RET   Z           ;Done if moved in 5
      PUSH  AF
      INC   DE
      LD    A,(DE)            ;Test for end of extents
      CP    0FEH        ;Extent in use?
      JR    NC,OPNCB6   ;Jump if not
      LD    (HL),C            ;Stuff # of cumulative
      INC   HL          ;  grans to this
      LD    (HL),B            ;  allocation into FCB
      INC   HL
      JR    OPNCB5            ;Loop for next
;
;     Unused extents - Put X'FFFF' in remaining fields
;
OPNCB6      POP   AF          ;Recover counter
      RLCA              ;Make times 4 and
      RLCA              ;  fill remaining
      LD    B,A         ;  extent bytes with
OPNCB7      LD    (HL),0FFH   ;  0FFH
      INC   HL
      DJNZ  OPNCB7
      RET
;
;     INIT a file
;      HL => the address of a 256-byte buffer
;      DE => the address of a 32-byte FCB
;       B => the logical record length (LREC)
;
INIT  CALL  LNKFCB@           ;Link to FCB
      LD    (OPNCB1+1),A      ;Start FCB+1 with 0
      PUSH  HL
      LD    HL,SFLAG$   ;Reset called by RUN bit
      RES   2,(HL)
      POP   HL
      CALL  OPEN1       ;Can we "OPEN" the file?
      RET   Z           ;Return if file existing
      CP    24          ;Return if error not
      RET   NZ          ;  "file not found"
      LD    A,10H       ;Set dir rec to show
      LD    (DIR$INIT),A      ;  assigned
      LD    A,(OPEN2+1) ;P/u the drive entry
      LD    C,A
      INC   A           ;Jump if a drive entry
      PUSH  AF
      JR    NZ,INIT1    ;  was made
      LD    C,A
INIT1 POP   AF          ;Stack integrity
      CALL  CKDRV       ;Is this drive available?
      JR    NZ,INIT2    ;Jump if not
      JR    C,INIT2           ;  or if write protected
      CALL  @HITRD            ;Read Hash Index Table
      RET   NZ          ;Return if read error
      CALL  SPRHIT            ;Locate spare entry
      JR    Z,INIT4           ;Jump if space
      XOR   A           ;Set status of CKDRV=Z
INIT2 PUSH  AF          ;Save last CKDRV status
      CALL  TESTDRV
      JR    C,INIT1           ;Loop if not at end
      LD    A,(OPEN2+1) ;If drive spec not entered
      INC   A           ;  then "directory full
      JR    NZ,INIT2A
      POP   AF          ;Stack integrity
      JR    ERR26
INIT2A      POP   AF          ;If no drive then
      JR    NZ,ERR32    ;  "illegal drive... else
      JR    C,ERR15           ;If Cy then "write protected
ERR26 LD    A,26        ;  else "directory space full
      DB    1           ;Mask with LD BC,nnnn
ERR15 LD    A,15        ;  if fall through
      DB    1           ;Mask .
ERR32 LD    A,32        ;
      OR    A           ;Set NZ for error
      RET
;
;     Found a spare HIT entry position
;
INIT4 LD    B,L         ;Save DEC
      LD    A,(FILEHASH)      ;P/u filespec hash
      LD    (HL),A            ;  & store in HIT
      CALL  @HITWR            ;Write updated HIT
      CALL  Z,@DIRRD    ;Read that dir record
      RET   NZ          ;Return if read error
      PUSH  HL
      PUSH  BC
      EX    DE,HL
      LD    BC,5        ;Move 1st 5 bytes into
      LD    HL,DIR$INIT ;  directory record
      LDIR
      LD    C,17        ;Move filename & password
      LD    HL,NAME$EXT ;  info into directory
      LDIR
      EX    DE,HL
      LD    B,10        ;Put X'FFFF' into 5 extents
INIT5 CALL  OPNCB7            ;4 for the ext's & 1 for
      POP   BC          ;  starting info
      CALL  @DIRWR            ;Write updated directory
      POP   HL
      RET   NZ          ;Return if write error
      CALL  OPNCB       ;  else open the FCB
      SCF               ;Indicate new file by C fl
      RET
;
;     Xfer the filespec to system buffer area
;
XFRSPEC     LD    B,19
      LD    DE,PSWDBUF
      LD    A,20H       ;Blank out the filename
XSPEC1      LD    (DE),A            ;  field in system buffer
      INC   DE
      DJNZ  XSPEC1
      LD    A,0FFH            ;Set drive to X'FF' for
      LD    (OPEN2+1),A ;  checking user entry
      LD    E,NAME$EXT&0FFH   ;Xfer filename
      CALL  XSPEC8
      LD    C,A
      LD    A,B
      SUB   8           ;Any valid chars found?
      JR    NZ,XSPEC3   ;Jump if valid name
;
;     Filename was invalid format
;
      OR    19          ;"illegal file name"
      RET
;
;     Continue to check file spec
;
XSPEC3      LD    A,C
      CP    '/'         ;Ext entered?
      LD    E,FILE$EXT&0FFH
      LD    B,3
      CALL  Z,XSPEC8A   ;Xfer the extension
      CP    '.'         ;Password entered?
      LD    E,PSWDBUF&0FFH
      CALL  Z,XSPEC8    ;Xfer the password
      CP    ':'         ;Drive entered?
      JR    NZ,XSPEC6
      LD    A,(HL)            ;P/u drive #
      SUB   '0'         ;Convert to binary
      LD    (OPEN2+1),A ;Stuff drive #
      AND   0F8H        ;Must be <0-7>
      LD    A,32        ;"illegal drive #"
      RET   NZ          ;Return error if out
      INC   HL          ;  of range
      LD    A,(HL)            ;Does filespec end in
XSPEC6      SUB   21H         ;  exclamation point?
      LD    A,8         ;Init to set bit 3 of
      JR    Z,XSPEC7    ;  FCB+1 & jump if "!"
      XOR   A           ;  else reset if not
XSPEC7      LD    (OPNCB3+1),A
      RET
;
;     ?
;
XSPEC8      LD    B,8
XSPEC8A     LD    A,(HL)            ;P/u a filespec character
      INC   HL          ;  & 1st test for A-Z
      JR    XSPEC10
XSPEC9      LD    A,(HL)            ;P/u a filespec character
      INC   HL          ;Advance to next one
      CP    '0'         ;Check for 0-9
      RET   C
      CP    '9'+1
      JR    C,XSPEC11
XSPEC10     CP    'A'         ;Check for A-Z
      RET   C
      CP    'Z'+1
      RET   NC
XSPEC11     LD    (DE),A            ;Character if valid
      INC   DE          ;Advance to next one
      DJNZ  XSPEC9            ;  & loop
      LD    A,(HL)            ;P/u following character
      INC   HL
      RET
;
;     Routine to find a spare HIT entry
;     Calculate the number of directory sectors
;     = (#sectors x #heads) - 2 for GAT & HIT
;
SPRHIT      EQU   $
      LD    A,7         ;Get highest # sector
      CALL  @DCTBYT
      PUSH  DE
      LD    D,A         ;Store heads & sectors
      AND   00011111B   ;Rake off # sectors
      LD    E,A         ;  & stuff into E
      INC   E           ;Adjust for 0 offset
      XOR   D           ;Recover # heads
      RLCA              ;  into bits 0-2
      RLCA
      RLCA
      INC   A           ;Adjust for 0 offset
      CALL  @MUL8       ;Multiply sectors x heads
      LD    E,A         ;Now check if double-sided
      LD    A,4
      CALL  @DCTBYT
      BIT   5,A         ;Set if 2-sided
      LD    A,E
      JR    Z,ONESID    ;Go if not set else
      ADD   A,A         ;  double the value
ONESID      POP   DE
      SUB   2           ;Reduce for GAT & HIT
      LD    (GSH3+1),A  ;Stuff for compare
;
;     Search across rows
;
      LD    L,27H       ;Try to use a HIT
      CALL  GSHLOOP           ;  past the SYS slots
      RET   Z           ;Return if spare found
;
      LD    L,1         ;Start after DIR slot
GSHLOOP     INC   L           ;Step to next
      JR    NZ,GSHTRY   ;Go it not done yet
      OR    H           ;Set NZ flag
      RET               ;Return failure
GSHTRY      LD    A,L         ;Skip unused parts
      AND   1FH
GSH3  CP    0           ;Cp with # of dir sectors
      LD    A,L
      JR    C,GSHOK           ;Go if NOT unused
      OR    1FH         ;Force to end of row
      LD    L,A
      JR    GSHLOOP           ;Loop back & ck for end
GSHOK LD    A,(HL)            ;P/u HIT byte
      OR    A           ;Free?
      RET   Z           ;Done if so
      JR    GSHLOOP           ;Try next
;
;     Routine to rename a filespec/devspec
;
```

```
REN0  LD    A,18H
      LD    (WASMAT),A
      OR    A           ;Denote "file not in dir
      RET               ;Ret w NZ condition
RENAME      CALL  LNKFCB@           ;Save regs & link to IX
      LD    A,(IX+0)    ;If a device, use the
      SUB   '*'         ;  "device" routine
      JR    Z,RENDEV
      CP    'R'!80H-'*' ;Special open condition?
      JR    Z,REN0            ;Go if so
      PUSH  HL          ;Save new pointer
      LD    HL,SFLAG$   ;Set don't test flags
      SET   0,(HL)
      CALL  OPEN1       ;Open the "old" spec
      POP   HL
      RET   NZ          ;Exit on error
      LD    A,(IX+1)    ;Make sure user has
      AND   7           ;  permission to rename
      CP    3
      JR    C,REN1
      LD    A,25H       ;"illegal acces...
      OR    A
      RET
;
;     User has acces to rename - locate drivespec
;
REN1  PUSH  HL          ;Save start
REN2  LD    A,(HL)            ;P/u char of new spec
      INC   HL
      CP    CR
      JR    Z,REN3            ;Go on ENTER
      CP    3
      JR    Z,REN3            ;Go on ETX
      CP    ':'
      JR    NZ,REN2           ;Loop on colon
REN3  DEC   HL          ;Back up to where the
      LD    (HL),':'    ;  colon should go
      INC   HL          ;  & force the drivespec
      LD    A,(IX+6)    ;  to the same as "old"
      LD    C,A         ;Keep drivespec in C
      AND   7
      ADD   A,'0'       ;Make it an ASCII digit
      LD    (HL),A
      INC   HL
      LD    (HL),CR
      LD    B,(IX+7)    ;Get DEC
      POP   IX          ;Put "new" FCB into IX
      PUSH  BC          ;  & save DEC on drive
      LD    HL,SFLAG$   ;Set don't test flags
      SET   0,(HL)
      CALL  OPEN1       ;Open the "new" spec
      POP   BC
      JR    NZ,REN4           ;Should error here
REN3A LD    A,19        ;  or else return
      OR    A           ;  if "new" is existing
      RET               ;  & we opened it
REN4  CP    24          ;If not "file not found"
```

```
      RET   NZ          ;  then is error
      CALL  @DIRRD            ;Read "old"'s directory
      RET   NZ
      PUSH  BC          ;Save drive spec
      LD    D,H         ;Xfer buffer high order
      LD    A,L
      ADD   A,5         ;Pt to filename field
      LD    E,A         ;Set buffer low order
      LD    HL,NAME$EXT ;Point to where the
      LD    BC,11       ;  new name is stored
      LDIR              ;Move in new name
      POP   BC
      CALL  @DIRWR            ;Rewrite the directory
      CALL  Z,@HITRD    ;Read the HIT
      RET   NZ
      LD    D,H         ;Set the buffer high order
      LD    E,B         ;Set the exact HIT low order
      LD    HL,NAME$EXT ;This doesn't change C fl
      CALL  HASHNAME    ;Hash the new name
      LD    (DE),A            ;Stuff code into HIT
      JP    @HITWR            ;Rewrite & exit
;
;     Routine to rename a device
;
RENDEV      PUSH  HL          ;Save new pointer
      CALL  GETDCB            ;Locate "old" in tables
      POP   IX          ;Recover pointer to "new"
      RET   NZ          ;Back if not in tables
      LD    A,L
      CP    DCBKL$            ;Ck if protected device
      LD    A,40        ;"Protected system device
      RET   C
      LD    A,(IX+0)    ;"new" must be a device
      CP    '*'
      JR    NZ,REN3A    ;"illegal file name...
      PUSH  HL          ;Save address of "old"
      CALL  GETDCB            ;Ck if "new" is unused
      POP   HL          ;Rcvr address of "old"
      JR    Z,REN3A
      LD    BC,6        ;Point to name field
      ADD   HL,BC       ;  of "old" device
      LD    (HL),E            ;Stuff new name into
      INC   HL          ;  Device Control Block
      LD    (HL),D
      XOR   A           ;Set Z-flag
      RET
;
;     Parameter storage area
;
FILEHASH    DS    1
PSWDBUF     DS    8
NAME$EXT    DS    8
FILE$EXT    DS    3
PW$HASH1    DS    2
PW$HASH2    DS    2
      DW    0           ;ERN init
DIR$INIT    DB    0,0,0,0
```

```
LREC$ DS    1
LAST  EQU   $
      IF    $.GT.DIRBUF$
      ADISP 'ERROR:  Module is too large'
      ENDIF
      ORG   MAXCOR$-2
      DW    LAST-SYS2   ;Overlay length
;
      END   SYS2
```

SYS3/ASM

```
;SYS3/ASM - LS-DOS 6.2
      ADISP '<SYS3 - LS-DOS 6.2>'
;
*LIST OFF               ;Get SYS0/EQU
*REF  'SYS0/EQU:1'
*LIST ON
LF    EQU   10
CR    EQU   13
;
*GET  'COPYCOM:1'       ;Copyright message
;
      ORG   1E00H
;
SYS3  AND   70H
      RET   Z           ;Back on zero entry
      CP    10H
      JR    Z,CLOSE           ;Jump if close
      CP    20H
      JP    Z,FNAME           ;Jump if filespec recover
      RET
CLOSE LD    A,(DE)            ;Test for device
      BIT   7,A
      JP    Z,CLOSDEV   ;Jump if closing device
      CALL  CKOPEN@           ;Test for open file
      LD    C,(IX+6)    ;P/u drive #
;
;     Special MINI check drive routine
;
      PUSH  IY          ;Save IY
      CALL  @GTDCT            ;Pick up DCT for drive
CKAGN CALL  @RSLCT            ;Wait until not busy
      JP    NZ,HOLDUP   ;Go to error handler
      BIT   3,(IY+3)    ;If hard drive, bypass
      JR    NZ,SAWBLK
      BIT   4,(IY+4)    ;If "ALIEN" bypass
      JR    NZ,SAWBLK
      BIT   7,(IY+4)    ;Ck if CKDRV inhibit
      JR    NZ,SAWBLK   ;Go if so
;
;     Test for diskette in drive (no index)
;
      PUSH  DE
      LD    D,(IY+5)    ;P/u current track
      LD    E,0         ;Set sector to 0
      CALL  @SEEK       ;Do a command
      POP   DE
      LD    B,30H       ;Set up count (short)
BLACK CALL  @RSLCT            ;Check for index pulse
      BIT   1,A         ;Test index
      JR    Z,SAWBLK    ;Saw black, seems OK
      DJNZ  BLACK
      JP    HOLDUP            ;Close fault handler
;
;     Diskette is there, let's continue
```

```
;
SAWBLK      POP   IY          ;Restore IY
      LD    B,(IX+7)    ;P/u DEC of FPDE
      CALL  @DIRRD            ;Read the directory
      RET   NZ          ;Quit if error there
      BIT   4,(HL)            ;Ck for killed file
      RET   Z           ;Quit if killed file
      PUSH  HL
      PUSH  BC
      CALL  RWRIT@            ;Write last buffer?
      POP   BC
      POP   HL
      RET   NZ          ;Ret on I/O error
      BIT   6,(IX+0)    ;If user does not have
      JP    Z,RCVN0           ;  close authority...
      INC   L           ;  else reset possible
      RES   5,(HL)            ;  file open bit in DIR+1
      INC   L           ;Determine if the EOF
      INC   L           ;  byte has been changed
      LD    A,(IX+8)    ;P/u EOF byte offset
      PUSH  HL          ;Save ptr to DIR+3
      CP    (HL)
      JR    NZ,CLOS1    ;Go if moved
      LD    A,11H
      ADD   A,L
      LD    L,A
      LD    A,(IX+12)   ;P/u low-order ERN
      CP    (HL)
      JR    NZ,CLOS1    ;Go if moved
      INC   L
      LD    A,(IX+13)   ;P/u high-order ERN
      CP    (HL)
      JR    NZ,CLOS1    ;Go if moved
      POP   AF
      JR    CLOS2       ;Didn't move
;
;     Routine to change a 3-byte EOF marker
;
CLOS1 POP   HL          ;Pop DIR+3
      LD    A,(IX+8)    ;Xfer the EOF offset
      LD    (HL),A
      LD    A,11H
      ADD   A,L
      LD    L,A
      LD    A,(IX+12)   ;  and the ERN from the FB
      LD    (HL),A
      INC   L
      LD    A,(IX+13)   ;  to the DIR entry
      LD    (HL),A
      BIT   2,(IX+0)    ;If the file was updated
      JR    NZ,CLOS3    ;  then update MOD date
      JR    CLOS5       ;  else don't
;
;     Three-byte EOF marker did not change
;
CLOS2 BIT   2,(IX+0)    ;If file was updated
      JR    NZ,CLOS3    ;  then update MOD date
      BIT   6,(IX+0)    ;If close authority then
      JR    NZ,CLOS5    ;  write back the DIR
      JR    CLOS6       ;  else continue
;
;     Routine to insert packed date into entry
;
CLOS3 PUSH  HL          ;Save ptr to DIR+21
      LD    A,L         ;Pt to start of dir rec
      AND   0E0H
      LD    L,A
      INC   L           ;Pt to DIR+1
      SET   6,(HL)            ;Set the MOD flag
      LD    DE,DATE$    ;Point to the year
      LD    A,(DE)            ;If year = 0, then date
      OR    A           ;  is 00/00/00
      JR    Z,$+4
      SUB   80          ;Offset from 1980
      PUSH  BC
      LD    B,A         ;Year-80 -> regB
      INC   DE          ;Point to day
      LD    A,(DE)            ;Shift day into 3-7 &
      RLCA              ;  merge the year into
      RLCA              ;  the lo-order bits
      RLCA
      OR    B
      INC   L
      LD    (HL),A            ;Store day/year
      DEC   L
      INC   DE          ;Point to month
      LD    A,(DE)
      LD    B,A
      LD    A,(HL)            ;P/u dir byte
      AND   0F0H        ;Strip old month
      OR    B           ;Merge month &
      LD    (HL),A            ;  update the field
      POP   BC
CLOS4 POP   HL          ;Rcvr DIR+21
CLOS5 PUSH  HL
      CALL  @DIRWR            ;Write back DIR entry
      POP   HL
      RET   NZ
CLOS6 INC   L           ;Pt to DIR+22 which is
      PUSH  HL          ;  the 1st extent
      LD    A,L
      SUB   15H         ;Back up to DIR+1
      LD    L,A
      BIT   7,(HL)            ;Test if created
      POP   HL
      JP    NZ,RCVN0    ;Bypass if created
      LD    DE,0        ;Init gran counter
CLOS7 LD    A,(HL)            ;P/u cyl indicator
      INC   L           ;Pt to gran alloc
      CP    0FEH        ;Extent in use?
      JR    NC,CLOS8    ;Jump if spare or FXDE
      LD    A,(HL)            ;P/u granule allocation
      INC   L           ;Pt to next extent
      AND   1FH         ;Strip off # of grans &
      INC   A           ;  adjust for zero offset
      ADD   A,E         ;Accumulate the number of
      LD    E,A         ;  grans in this extent
      JR    NC,CLOS7    ;Any previous quantity
      INC   D
      JR    CLOS7
CLOS8 JR    NZ,CLOS9    ;Found all grans in this
      LD    B,(HL)            ;  extent, ck for FXDE
      CALL  @DIRRD
      RET   NZ
      LD    A,L         ;Point to extents in FXDE
      ADD   A,16H
      LD    L,A
      JR    CLOS7       ;Go to continue count
;
;     Routine to determine need to deallocate
;
CLOS9 PUSH  HL          ;Save ptr to last extent
      LD    L,(IX+12)   ;P/u ending record #
      LD    H,(IX+13)
      LD    A,8         ;Get # sectors/gran
      CALL  @DCTBYT
      AND   1FH         ;Remove other data
      PUSH  AF          ;Save the #
      ADD   A,L         ;Round up to next
      LD    L,A         ;  higher gran
      JR    NC,CLOS10
      INC   H
CLOS10      POP   AF          ;Rcvr # sectors/gran
      INC   A           ;Adjust for division
      CALL  @DIV16            ;Calculate # grans in use
      XOR   A           ;Subtract the # of grans
      EX    DE,HL       ;  used from the # of
      SBC   HL,DE       ;  grans allocated in the
      EX    DE,HL       ;  directory, and move DE
      POP   HL          ;Rcvr ptr to last extent
      JP    Z,RCVN0           ;Jump if same quantity
      JP    C,RCVN0           ;Jump if now more
;
;     Need to deallocate space
;
      CALL  @GATRD            ;Read GAT
      RET   NZ
      JR    BAKUP       ;B/u to last used extent
CLOS11      PUSH  DE          ;Sv count of excess grans
      LD    A,(HL)            ;P/u alloc info
      AND   0E0H        ;Get starting relative
      RLCA              ;  gran into reg-E
      RLCA
      RLCA
      LD    E,A
      LD    A,(HL)            ;# of contiguous grans
      AND   1FH         ;Remove unneeded data
      ADD   A,E         ;Calculate ending
      LD    E,A         ;  relative gran #
      LD    A,8         ;P/u the # of grans
      CALL  @DCTBYT           ;  per cylinder
```

```
        RLCA
        RLCA
        RLCA
        AND   7            ;Move into bits 0-2
        INC   A            ;Adjust for zero offset
        LD    D,A          ;Save count
        LD    A,4
        CALL  @DCTBYT
        BIT   5,A          ;2-sided disk?
        LD    A,D          ;Rcvr count
        JR    Z,$+3        ;Bypass if 1-sided
        RLCA               ;Double count
        CALL  @DIV8        ;A=quotient, E=remainder
        DEC   L            ;Pt to starting cylinder
        ADD   A,(HL)             ;Bump cyl pointer by how
        LD    D,A          ;  many excessive cyls to
        PUSH  HL           ;  start from the rear
        PUSH  BC
        LD    H,DIRBUF$>8 ;Pt to that cyl's GAT
        LD    L,D
        LD    B,(HL)             ;P/u the GAT allocation
        LD    A,E
        CALL  CALCBIT            ;Deallocate a gran
        LD    (HL),B             ;Replace GAT byte
        POP   BC
        POP   HL
        INC   L            ;Repoint to alloc info
        DEC   (HL)         ;Reduce by 1 gran
        LD    A,(HL)             ;Get info on contig gran
        INC   A            ;Adj for zero offset
        AND   1FH          ;Mask off unneeded
        POP   DE           ;Rcvr excess gran count
        DEC   DE           ;  and count down
        JR    NZ,CLOS12    ;Go if extent still used
BAKUP   LD    (HL),0FFH    ;  else extent is spare
        DEC   L
        LD    (HL),0FFH
        DEC   L
        LD    A,L          ;Check if backed all the
        AND   1FH          ;  way thru this entry
        CP    15H
        JR    NZ,CLOS12    ;Go if not
        XOR   L            ;Deallocate this FXDE
        LD    L,A
        BIT   7,(HL)             ;Was it the FPDE?
        JR    Z,CLOS12     ;Bypass if FPDE
        LD    (HL),0             ;Show dir is spare
        CALL  @DIRWR             ;Write back
        RET   NZ
        LD    A,B          ;P/u deallocated DEC
        AND   0E0H
        INC   A            ;Pt to DIR+1
        LD    L,A
        LD    A,(HL)             ;P/u previous DEC
        LD    (STUFDEC+1),A      ;Save in opcode ahead
        CALL  @HITRD             ;Read the HIT
        RET   NZ
```

```
      LD    L,B         ;Point to deallocated HIT
      LD    (HL),0            ;Deallocate space in HIT
      CALL  @HITWR            ;Write back
      RET   NZ
STUFDEC     LD    B,0         ;P/u previous DEC
      CALL  @DIRRD            ;Read its dir entry
      RET   NZ
      LD    A,B
      OR    1FH         ;Pt to end of entry
      LD    L,A
      LD    (HL),0FFH   ;Erase pointer
      DEC   L           ;  to deallocated FXDE
      LD    (HL),0FFH
      DEC   L           ;Point to previous extent
      PUSH  HL          ;Save pointer
      CALL  @DIRWR            ;Write back
      POP   HL
      RET   NZ
CLOS12      LD    A,D         ;Loop if still more to
      OR    E           ;  deallocate
      JP    NZ,CLOS11
      CALL  @DIRWR
      JR    Z,CLOS13    ;Go if no write error
      CP    15          ;"write protected
      RET   NZ          ;Bad if not
      JR    RCVN0
;
CLOS13      CALL  @GATWR            ;Write back the altered GAT
      RET   NZ
;
;     Routine starts to recover file spec
;
RCVN0 LD    A,(IX+7)    ;P/u DEC of FPDE
      LD    C,(IX+6)    ;P/u drive
      XOR   B           ;Check if its directory
      AND   1FH         ;  record is resident
      LD    B,(IX+7)    ;P/u DEC of FPDE
      CALL  NZ,@DIRRD   ;Get FPDE dir if needed
      RET   NZ
      PUSH  IX          ;Transfer FCB to DE
      POP   DE
RCVNAM      LD    A,C
      AND   7           ;Convert drive to ASCII
      OR    '0'
      LD    (RCVN5+1),A
      LD    H,SBUFF$>8  ;Pt to DIR+5 (name)
      LD    A,B
      AND   0E0H
      OR    5
      LD    L,A
      PUSH  HL          ;Save name start posn
      LD    B,8         ;Init 8 chars max
RCVN1 LD    A,(HL)            ;Move filename from
      CP    ' '         ;  direc to FCB
      JR    Z,RCVN2
      LD    (DE),A
      INC   HL
      INC   DE
      DJNZ  RCVN1       ;Loop up to 8
RCVN2 POP   HL
      LD    A,L
      ADD   A,8         ;Pt to extension
      LD    L,A
      LD    A,(HL)
      CP    ' '
      JR    Z,RCVN4           ;Jump if none
      LD    A,'/'
      LD    (DE),A            ;Stuff separator into FCB
      INC   DE
      LD    B,3         ;Init 3-char extension
RCVN3 LD    A,(HL)            ;Stuff the ext
      CP    ' '         ;  into FCB
      JR    Z,RCVN4
      LD    (DE),A
      INC   HL
      INC   DE
      DJNZ  RCVN3
RCVN4 LD    A,':'       ;Stuff drive indicator
      LD    (DE),A
      INC   DE
RCVN5 LD    A,0         ;P/u drive in ASCII
      LD    (DE),A            ;  & stuff it
      INC   DE
      LD    A,03H       ;Close FCB with ETX
      LD    (DE),A
      XOR   A           ;Set Z for no error
      RET
;
;     Routine to recover the filespec
;
FNAME PUSH  HL
      PUSH  DE
;
;     Calculate the number of directory sectors
;     = (#sectors x #heads) - 2 for GAT & HIT
;
      LD    A,7         ;Get highest # sector
      CALL  @DCTBYT
      LD    D,A         ;Store heads & sectors
      AND   1FH         ;Mask for # sectors
      LD    E,A         ;  & stuff into E
      INC   E           ;Bump for 0 offset
      XOR   D           ;Rcvr # heads, destroy # secs
      RLCA              ;Rotate into bits 0-2
      RLCA
      RLCA
      INC   A           ;Bump for 0 offset
      CALL  @MUL8       ;Multiply sectors x heads
      LD    E,A         ;Now check double bit
      LD    A,4
      CALL  @DCTBYT
      BIT   5,A         ;2-sided if set
      LD    A,E
      JR    Z,ONESID    ;Go if not set
      ADD   A,A         ;  else double value
ONESID      SUB   2           ;Reduce for GAT & HIT
      LD    D,A
      LD    A,B
      AND   1FH         ;Calc req sector #
      CP    D
      JR    C,FNAM1
      LD    A,16        ;"Illegal logical file #
      OR    A
      JR    FNAM2
FNAM1 POP   DE          ;Reget Cyl/Sec
      PUSH  DE
      CALL  @DIRRD
      CALL  Z,RCVNAM    ;Rcvr the filespec
FNAM2 POP   DE
      POP   HL
      RET
;
;     Close a logical device
;
CLOSDEV     CP    10H         ;If not open device,
      LD    A,38        ;  return "file not open...
      RET   NZ
      CALL  LNKFCB@           ;Link to FCB
      LD    C,(IX+6)    ;Get device name
      LD    B,(IX+7)
      LD    (IX+0),'*'  ;Stuff device indicator
      LD    (IX+1),C    ;Stuff 1st char of name
      LD    (IX+2),B    ;Stuff 2nd char of name
      LD    (IX+3),3    ;Terminate with ETX
      XOR   A
      RET
;
;     Calculate GAT bit to deallocate
;
CALCBIT     AND   7           ;Make binary bit # into
      RLCA              ;  the proper RES
      RLCA              ;  opcode
      RLCA
      OR    80H
      LD    (CALC1+1),A
CALC1 RES   0,B         ;Reset bit in GAT
      RET
;
;     User removed disk with an open file
;
HOLDUP      PUSH  HL
      PUSH  DE
      LD    HL,HOLDUP$  ;Pt to message
      CALL  @DSPLY            ;Display to console
      CALL  @CKBRKC           ;Clear out break bit
WAITING     CALL  @KBD        ;Scan the keyboard
      JR    NZ,WAITING  ;Keep looking
      CP    CR          ;Check for <ENTER>
      JR    Z,TRYNOW
      CALL  @CKBRKC           ;Check for a break
      JR    Z,WAITING
```

```
ABRT  POP   DE
      POP   HL
      POP   IY           ;Restore from above
      LD    A,32         ;Show illegal drive #
      OR    A            ;Set NZ condition
      RET                ;Go back now
TRYNOW      POP   DE
      POP   HL
      JP    CKAGN        ;Try checking again
HOLDUP$     DEFB  LF,'** CLOSE FAULT **  Drive not ready, '
      DEFB  '<ENTER> to retry, <BREAK> to abort',CR
LAST  EQU   $
      IF    $.GT.DIRBUF$
      ADISP 'ERROR: Module too big'
      ENDIF
      ORG   MAXCOR$-2
      DW    LAST-SYS3    ;Overlay length
;
      END   SYS3
```

SYS4/ASM

```
;SYS4/ASM - LS-DOS 6.2
      ADISP '<SYS4 - LS-DOS 6.2>'
LF    EQU   10
CR    EQU   13
*LIST OFF               ;Get SYS0/EQU
*REF  'SYS0/EQU:1'
*LIST ON
*GET  'COPYCOM:1'       ;Copyright message
;
      ORG   1E00H
;
SYS4  JP    BEGIN
;
;     Sentence table - Must be totally within one page
;
MSG0  DB    1,2+80H
;           no error
MSG1  DB    4,2,5,6,9+80H
;           parity error during header read
MSG2  DB    8,2,5,9+80H
;           seek error during read
MSG3  DB    11,7,5,9+80H
;           lost data during read
MSG4  DB    4,2,5,9+80H
;           parity error during read
MSG5  DB    7,27,12,44,5,9+80H
;           data record not found during read
MSG6  DB    13,9,15,7,27+80H
;           attempted to read system data record
MSG7  DB    13,9,14,7,27+80H
;           attempted to read locked/deleted data record
MSG8  DB    42,12,51+0C0H
;           device not available
MSG9  DB    4,2,5,6,10+80H
;           parity error during header write
MSG10 DB    8,2,5,10+80H
;           seek error during write
MSG11 DB    11,7,5,10+80H
;           lost data during write
MSG12 DB    4,2,5,10+80H
;           parity error during write
MSG13 DB    7,27,12,44,5,10+80H
;           data record not found during write
MSG14 DB    10,21,18,19,48+80H
;           write fault on disk drive
MSG15 DB    10,22,19+80H
;           write protected disk
MSG16 DB    23,24,26,25+80H
;           illegal logical file number
MSG17 DB    16,9,2+80H
;           directory read error
MSG18 DB    16,10,2+80H
;           directory write error
MSG19 DB    23,26,41+0C0H
```

```
;            illegal file name
MSG20 DB    34,9,2+80H
;            gat read error
MSG21 DB    34,10,2+80H
;            gat write error
MSG22 DB    35,9,2+80H
;            hit read error
MSG23 DB    35,10,2+80H
;            hit write error
MSG24 DB    26,12,45,16+0C0H
;            file not in directory
MSG25 DB    26,46,49+0C0H
;            file access denied
MSG26 DB    1,16,39,51+0C0H
;            directory space full
MSG27 DB    19,39,47+0C0H
;            disk space full
MSG28 DB    28,29,26,32+80H
;            end of file encountered
MSG29 DB    27,25,30,29,31+80H
;            record number out of range
MSG30 DB    16,47,52,26+80H
;            directory full - can't extend file
MSG31 DB    50,12,44+0C0H
;            program not found
MSG32 DB    23,48,25+0C0H
;            illegal drive number
MSG33 DB    1,42,39,51+0C0H
;            no device space available
MSG34 DB    38,26,43,2+80H;
;            load file format error
MSG35 DB    17,21+80H
;            memory fault
MSG36 DB    13,38,9,40,17+80H
;            attempted to load read only memory
MSG37 DB    23,46,13,22,26+80H
;            illegal access attempted to protected file
MSG38 DB    26,12,53+0C0H
;            file not open
MSG39 DB    42,45,54+80H
;            device in use
MSG40 DB    22,15,42+80H
;            protected system device
MSG41 DB    26,57,53+0C0H
;            file already open
MSG42 DB    24,27,58,53,21+0C0H
;            logical record length open fault
MSG43 DB    56,20,2+80H
;            SVC parameter error
MSG44 DB    20,2+80H
;            Parameter error
MSG45 DB    37,2,33+80H
;            unknown error code
BEGIN AND   70H         ;What's the entry?
      RET   Z           ;Back on zero
      PUSH  AF
      LD    A,(LSVC$)   ;Grab the last SVC
      LD    (SVSVC+1),A ;  and store for later
      POP   AF
      LD    (EXTEND+1),HL     ;Value if extended error
      EX    (SP),HL           ;Grab return address
      LD    (ERR7+1),HL ;  & stuff it
      POP   HL
      POP   AF          ;Pop off the error code
      EX    (SP),HL           ;Get user ret address
      LD    (USRET+1),HL      ;  for long dsply
      EX    (SP),HL
      PUSH  HL          ;Save regs
      PUSH  DE
      PUSH  BC
      LD    HL,(SVCRET$)      ;Grab last SVC return
      LD    (SVRET+1),HL      ;  and save for display
      LD    B,A
      LD    A,(SFLAG$)  ;Test expanded-error flag
      AND   01000000B   ;  flag bit in system flag
      XOR   B
      AND   B
      LD    B,A         ;Xfer the result to B
      PUSH  AF          ;  & save for later
      AND   3FH         ;Strip all but error #
      LD    C,A         ;Place error code -> C
      LD    HL,CFLAG$   ;If system error suppress
      BIT   6,(HL)            ;  flag is set, don't
      JP    NZ,ERR6A    ;  display error message
      BIT   7,(HL)            ;If error-to-buffer is
      JR    NZ,ERR0           ;  set, put to user bufr
      LD    DE,SBUFF$
      JR    ERR0A       ;Branch around force
ERR0  SET   6,B         ;Force buffer to abbrev
      POP   AF
      SET   6,A
      PUSH  AF
ERR0A BIT   6,B         ;Expanded error display?
      LD    B,0
      JR    NZ,ERR2           ;Jump if abbreviated
      PUSH  BC
      LD    HL,ERRMSG   ;Pt to "< ERRCOD =...
      LD    C,MLEN            ;  & move to buffer
      LDIR
      POP   BC
      EX    DE,HL       ;Buffer ptr to HL
      LD    A,C         ;Error code to Accum
      LD    (HL),2FH    ;Init for digit conv
ERR1  INC   (HL)        ;Bump ASCII digit
      SUB   10          ;  count by 10
      JR    NC,ERR1           ;Keep bumping 10's digit
      INC   L           ;Bump buffer ptr
      ADD   A,'0'+10    ;Convert rmdr to unit's
      LD    (HL),A            ;  & place in buffer
      INC   L           ;Bump to next pos
      LD    (HL),','    ;Stuff a comma & bump
      INC   L
      LD    (HL),' '    ;  & a space
      INC   L
      EX    DE,HL       ;Buffer ptr back to DE
      PUSH  BC
      LD    HL,ERRMSG1  ;"Returns to X'"
      LD    BC,M1LEN
      LDIR
      EX    DE,HL       ;HL back to buffer
USRET LD    DE,$-$            ;User ret address
      CALL  @HEX16
      LD    A,27H       ;"'"
      LD    (HL),A
      INC   HL
      LD    (HL),LF           ;End with a linefeed
      INC   HL
      POP   BC
      BIT   6,C         ;Extended error?
      JR    NZ,ERR6           ;Go if not
      LD    (HL),'*'    ;Make long msg look nice
      INC   HL
      LD    (HL),'*'
      INC   HL
      LD    (HL),' '
      INC   HL
ERR6  EX    DE,HL       ;DE back to nxt buff line
ERR2  LD    A,C
      CP    63          ;"Extended error"?
      JR    NZ,ERR2A
;
;     Do extended error only
;
      PUSH  DE          ;Save buffer ptr
EXTEND      LD    DE,$-$            ;Ext'd err value from HL
      LD    HL,EXT$ERR+26
      CALL  @HEX16
      LD    HL,EXT$ERR  ;Point to error msg
      POP   DE          ;Recover buffer
      PUSH  HL          ;Save msg start
      PUSH  BC
      LD    BC,M2LEN    ;Len of error
      LDIR              ;Move into buffer
      POP   BC
      LD    HL,CFLAG$   ;See if to user buffer
      BIT   7,(HL)
      RES   7,(HL)            ;Dont logot if so
      POP   HL
      CALL  Z,@LOGOT
      JR    ERR6A       ;  and exit
;
;     Do regular (non-extended) error
;
ERR2A LD    A,45        ;If error code is > 43,
      CP    C           ;  then set to 44 (max)
      PUSH  DE          ;Save ptr to 1st char
      JR    NC,ERR3
      LD    C,A
ERR3  LD    HL,CODTAB   ;Pt to start of code
      ADD   HL,BC       ;  address table & index
      LD    L,(HL)            ;P/u lo-order vector
      LD    H,MSG0>8    ;Set hi-order vector
;
;     HL now points to sentence table
;
ERR5  LD    A,(HL)            ;P/u word offset
      AND   3FH         ;  & strip any flags
      LD    B,A         ;Xfer word # to reg B
      PUSH  HL          ;Save sentence pointer
      LD    HL,WORDS    ;Dictionary start
LP1   LD    A,(HL)            ;Scan through the table
      RLCA              ;  counting words (bit 7
      INC   HL          ;  denotes word end)
      JR    NC,LP1            ;  until requested word
      DEC   B           ;  is reached
      JR    NZ,LP1
;
;     Found start of a desired word
;
LP2   LD    A,(HL)            ;Transfer word until
      RLCA              ;  bit 7 set (last char)
      SRL   A           ;  while resetting bit-7
      LD    (DE),A            ;Stuff letter of word
      INC   HL          ;  & bump pointers
      INC   DE
      JR    NC,LP2
      LD    A,' '       ;Move a space into buffer
      LD    (DE),A
      INC   DE
      POP   HL          ;Rcvr ptr to sentence
      LD    A,(HL)            ;P/u this word byte
      INC   HL
      RLCA              ;Was this the last word?
      JR    NC,ERR5           ;Loop if still more to go
      EX    (SP),HL           ;Get ptr to 1st char
      LD    A,(HL)
      RES   5,A         ;Set it to Upper-Case
      LD    (HL),A
      POP   HL          ;Get back sentence ptr
      POP   AF          ;Rcvr error code
      PUSH  AF
      PUSH  HL          ;Save sentence ptr
      LD    A,CR
      LD    (DE),A            ;Stuff end-of-line
      LD    HL,CFLAG$   ;If to user buffer,
      BIT   7,(HL)            ;  then don't LOGOT
      RES   7,(HL)
      LD    HL,SBUFF$   ;Display the line
      CALL  Z,@LOGOT
      POP   HL
      POP   AF          ;Rcvr word index
      PUSH  AF
      BIT   6,A         ;Test if a disk error
      CALL  Z,DSPSPEC   ;Get filespec if it is
ERR6A POP   AF
      POP   BC
      POP   DE
      POP   HL
      OR    A           ;Ret to user if bit 7
ERR7  JP    M,0         ;  of error code is set
      JP    @ABORT            ;  else abort
;
;     Routine to display the filespec
;
DSPSPEC     PUSH  IX
      LD    IX,(JDCB$)  ;P/u FCB vector
      DEC   HL
      BIT   6,(HL)
      JR    NZ,DSPC2
      LD    C,(IX+6)    ;Device 1st char or drive
      LD    B,(IX+7)    ;Device 2nd char or drive
      BIT   7,(IX+0)    ;Test if file or device
      JR    NZ,RCVSPEC  ;Jump if it is a file
      LD    HL,OPN$DCB
DSPC1 LD    A,C         ;Possible devspec, 1st char
      CP    'A'
      JR    C,DCBUNK    ;C=do unknown
      CP    'Z'+1
      JR    NC,DCBUNK   ;Again, go if bunk
      LD    A,B         ;Check 2nd character
      CP    '0'
      JR    C,DCBUNK
      CP    'Z'+1
      JR    NC,DCBUNK
      LD    (OPN$DCB+18),BC   ;Stuff the device name
DSPC1A      EQU   $-2
      POP   IX
      JR    RSPC6       ;Go display it
;
DCBUNK      LD    HL,UNK$TYP
      POP   IX
      JR    RSPC6
;
DSPC2 LD    C,(IX+1)    ;P/u 1st char or vector
      LD    B,(IX+2)    ;P/u 2nd char or vector
      LD    A,(IX+0)
      LD    HL,DEV$NAM
      LD    (DSPC1A),HL ;Change dsply message
      LD    HL,DEV$EQ
      CP    '*'         ;IF '*', go to device
      JR    Z,DSPC1
      PUSH  IX          ;  else assume a file
      POP   HL
      LD    DE,FILE$EQ+7      ;Init "<file=...
      LD    B,24        ;Max filespec
DSPC3 LD    A,(HL)            ;P/u filespec char
      CP    3           ;ETX?
      JR    Z,DSPC3A
      CP    CR          ;EOL?
      JR    Z,DSPC3A
      OR    A
      JR    Z,DSPC3A    ;Zero ok terminator too
      CALL  CHKASC            ;Check if an ASCII char
      JR    C,DCBUNK    ;  and abort if not
      LD    (DE),A
      INC   DE
      INC   HL
      DJNZ  DSPC3       ;Loop until end
DSPC3A      LD    HL,FILE$EQ
      JR    RSPC5
;
;     Routine to get recover the filespec
;
RCVSPEC     LD    A,C
      ADD   A,30H       ;Conv drive # to decimal
      CP    '0'         ;Valid drive?
      JR    C,DCBUNK
      CP    '8'
      JR    NC,DCBUNK
      LD    (OPN$FCB+16),A
      LD    A,B         ;DEC into Accum
      LD    HL,OPN$FCB+23     ;Pt into msg string
      CALL  @HEX8       ;  and convert it
      EX    DE,HL       ;DE back to buff end
      LD    HL,OPN$FCB
      INC   DE
RSPC5 LD    A,CR        ;Close with EOL
      LD    (DE),A
      POP   IX
RSPC6 CALL  @LOGOT            ;Log it
;
;     Build the SVC info line
;
      LD    DE,LILBUF   ;Tempy for hexdec
SVSVC LD    A,$-$       ;P/u stored last SVC
      LD    L,A
      LD    H,0         ;  into HL for conv
      CALL  @HEXDEC
      LD    DE,SVC$NUM+11
      CALL  EDEC
      LD    A,3         ;Then put ETX
      LD    (DE),A
;
      LD    HL,SVC$RET+16     ;Now, do last svc return
SVRET LD    DE,$-$
      CALL  @HEX16
      LD    HL,SVC$NUM
      CALL  @LOGOT
      LD    HL,SVC$RET
      JP    @LOGOT            ;Log it
;
;     Routine to check for vaild chars
;
CHKASC      LD    A,(HL)            ;Xfer until 1st space
      CP    '.'
      RET   C           ;Cy flg on ret = Bad Char
      CP    ':'+1
      JR    NC,CKASC1
      JR    CKASC2
CKASC1      CP    'A'
      RET   C
      CP    'Z'+1
```

```
CKASC2      CCF
      RET
;
EDEC  LD    HL,LILBUF   ;Pt to conved decimal num
ED1   LD    A,(HL)
      OR    A
      RET   Z
      CP    ' '
      INC   HL
      JR    Z,ED1
      LD    (DE),A            ;Store valid digit
      INC   DE
      JR    ED1
;
;
;
EXT$ERR     DB    '** Extended error, HL = X',27H,'xxxx',27H,CR
M2LEN EQU   $-EXT$ERR
ERRMSG      DB    LF,'** Error code = '
MLEN  EQU   $-ERRMSG
ERRMSG1     DB    'Returns to X',27H
M1LEN EQU   $-ERRMSG1
DEV$EQ      DB    'Device = *'
DEV$NAM     DB    'XX',CR
FILE$EQ     DB    'File = NNNNNNNN/EEE.PPPPPPPP:D',CR
OPN$FCB     DB    'Open FCB, Drive=n, DEC=   ',CR
OPN$DCB     DB    'Open DCB, Device=*xx',CR
UNK$TYP     DB    'Unknown FCB/DCB',CR
SVC$NUM     DB    'Last SVC = nnn',3
SVC$RET     DB    ', Returned to X',27H,'xxxx',27H,CR
;
LILBUF      DS    5
      DB    0
;
;     Table points to low-order bytes of messages
;
CODTAB      DB    MSG0&0FFH,MSG1&0FFH,MSG2&0FFH,MSG3&0FFH
      DB    MSG4&0FFH,MSG5&0FFH,MSG6&0FFH
      DB    MSG7&0FFH,MSG8&0FFH,MSG9&0FFH
      DB    MSG10&0FFH,MSG11&0FFH,MSG12&0FFH,MSG13&0FFH
      DB    MSG14&0FFH,MSG15&0FFH,MSG16&0FFH,MSG17&0FFH
      DB    MSG18&0FFH,MSG19&0FFH,MSG20&0FFH,MSG21&0FFH
      DB    MSG22&0FFH,MSG23&0FFH,MSG24&0FFH,MSG25&0FFH
      DB    MSG26&0FFH,MSG27&0FFH,MSG28&0FFH,MSG29&0FFH
      DB    MSG30&0FFH,MSG31&0FFH,MSG32&0FFH,MSG33&0FFH
      DB    MSG34&0FFH,MSG35&0FFH,MSG36&0FFH,MSG37&0FFH
      DB    MSG38&0FFH,MSG39&0FFH,MSG40&0FFH,MSG41&0FFH
      DB    MSG42&0FFH,MSG43&0FFH,MSG44&0FFH,MSG45&0FFH
;
;     Word dictionary
;
WORDS DB    'R'!80H                 ;Start table with bit 7
      DB    'n','o'!80H       ;1
      DB    'erro','r'!80H          ;2
      DB    'o'!80H                 ;3 extra word
      DB    'parit','y'!80H         ;4
      DB    'durin','g'!80H         ;5
      DB    'heade','r'!80H         ;6
      DB    'dat','a'!80H           ;7
      DB    'see','k'!80H           ;8
      DB    'rea','d'!80H           ;9
      DB    'writ','e'!80H          ;10
      DB    'los','t'!80H           ;11
      DB    'no','t'!80H            ;12
      DB    'attempted t','o'!80H   ;13
      DB    'locked/delete','d'!80H ;14
      DB    'syste','m'!80H         ;15
      DB    'director','y'!80H      ;16
      DB    'memor','y'!80H         ;17
      DB    'o','n'!80H       ;18
      DB    'dis','k'!80H           ;19
      DB    'paramete','r'!80H      ;20
      DB    'faul','t'!80H          ;21
      DB    'protecte','d'!80H      ;22
      DB    'illega','l'!80H  ;23
      DB    'logica','l'!80H  ;24
      DB    'numbe','r'!80H         ;25
      DB    'fil','e'!80H           ;26
      DB    'recor','d'!80H         ;27
      DB    'en','d'!80H            ;28
      DB    'o','f'!80H       ;29
      DB    'ou','t'!80H            ;30
      DB    'rang','e'!80H          ;31
      DB    'encountere','d'!80H    ;32
      DB    'cod','e'!80H           ;33
      DB    'GA','T'!80H            ;34
      DB    'HI','T'!80H            ;35
      DB    'y'!80H                 ;36
      DB    'unknow','n'!80H  ;37
      DB    'loa','d'!80H           ;38
      DB    'spac','e'!80H          ;39
      DB    'onl','y'!80H           ;40
      DB    'nam','e'!80H           ;41
      DB    'devic','e'!80H         ;42
      DB    'forma','t'!80H         ;43
      DB    'foun','d'!80H          ;44
      DB    'i','n'!80H       ;45
      DB    'acces','s'!80H         ;46
      DB    'ful','l'!80H           ;47
      DB    'driv','e'!80H          ;48
      DB    'denie','d'!80H         ;49
      DB    'progra','m'!80H  ;50
      DB    'availabl','e'!80H      ;51
      DB    '- can''t exten','d'!80H      ;52
      DB    'ope','n'!80H           ;53
      DB    'us','e'!80H            ;54
      DB    'o','r'!80H       ;55
      DB    'SV','C'!80H            ;56
      DB    'alread','y'!80H  ;57
      DB    'lengt','h'!80H         ;58
LAST  EQU   $
      IF    $.GT.DIRBUF$
      ADISP 'ERROR: Module too big'
      ENDIF
      ORG   MAXCOR$-2
      DW    LAST-SYS4   ;Overlay length
;
      END   SYS4
□
```

SYS5/ASM

```
;SYS5/ASM - LS-DOS 6.2
      ADISP '<SYS5 - LS-DOS 6.2>'
*LIST OFF                 ;Get SYS0/EQU
*REF  'SYS0/EQU:1'
*LIST ON
*GET  'COPYCOM:1'         ;Copyright message
;
*GET  'SYS5A:1'
;
      END   SYS5
```

SYS5A/ASM

```
;SYS5A/ASM - LS-DOS 6.2
;
      ORG   0A0H
;
;     References to save area in lowcore
;
SAVONE      DS    1
SAVTWO      DS    1
      DS    1           ;Space for saved byte (1)
NXTADR      DS    2
NXTBYT      DS    1
DSPADR      DS    2
AFREG DS    2           ;AF  register save area
      DS    2           ;BC
      DS    2           ;DE
HLREG DS    2           ;HL
      DS    8           ;AF', BC', DE', HL'
IXREG DS    2           ;IX
IYREG DS    2           ;IY
SPREG DS    1           ;SP
REGSAV      DS    1
PCREG DS    2           ;PC
;
      ORG   1E00H
;
SYS5  AND   70H         ;If entry = 0, return
      RET   Z
      POP   AF          ;Discard return to SYS0
      POP   AF          ;Get original reg-AF
      PUSH  AF
      PUSH  IY          ;Save remaining regs
      PUSH  IX
      EX    AF,AF'
      EXX
      PUSH  HL
      PUSH  DE
      PUSH  BC
      PUSH  AF
      EX    AF,AF'
      EXX
      PUSH  HL
      PUSH  DE
      PUSH  BC
      PUSH  AF
      LD    HL,0
      ADD   HL,SP       ;Place SP address into HL
      LD    DE,AFREG
      LD    BC,24       ;Move the 24 bytes saved
      LDIR
      LD    (SPREG),HL
      LD    SP,HL
      LD    HL,(PCREG)
      DEC   HL
      LD    A,(HL)            ;P/u the byte at PC
      CP    0F7H        ;  & check for breakpoint
      JR    NZ,$?1            ;Go if not a breakpoint
      LD    (PCREG),HL
;
;     This next routine picks up the data stored in the
;     instruction storage areas used to hold the
;     address & byte of the inserted RST's used to
;     control the single step mode. If the address
;     save area is zero, the an RST was not inserted.
;     Two areas are needed because DEBUG inserts
;     RST 48's at both CALL origin & destination.
;
$?1   LD    HL,SAVONE
      LD    B,2         ;Set up loop for 2 areas
$?2   XOR   A           ;Clear register A & flags
      LD    E,(HL)            ;P/u the next 2 bytes
      LD    (HL),A            ;  (where an address
      INC   HL          ;  would be stored) while
      LD    D,(HL)            ;  simultaneously setting
      LD    (HL),A            ;  the save area to zero
      INC   HL
      LD    A,E         ;Ck if the area was zero
      OR    D
      JR    Z,$?3       ;If zero, no RST entry
      LD    A,(DE)            ;Address save <> zero,
      CP    0F7H        ;  ck byte for RST 30H
      JR    NZ,$?3
      LD    A,(HL)            ; Was RST 30H, restore
      LD    (DE),A            ;  the program byte
$?3   INC   HL
      DJNZ  $?2         ;Loop thru 2 save areas
CMND  LD    SP,(SPREG)  ;Set up the stack
      CALL  WRREGS            ;  & display normal CRT
      LD    HL,16<8!0   ;Move cursor to 16,0
      LD    B,3         ;Command
      LD    A,15        ;Svc @VDCTL
      RST   28H         ;Set cursor
      CALL  INPUT@            ;Get command
      CP    'g'         ;Goto AAAA,(BBBB(,CCCC))
      JP    Z,CMD_G
      LD    HL,CMND           ;Set up a return branch
      PUSH  HL
      CP    's'         ;Set CRT to full screen?
      JR    Z,CMD_S
      CP    ';'         ;Inc CRT one page?
      JR    Z,CMD_INC
      CP    '-'         ;Dec CRT one page?
      JR    Z,CMD_DEC
      CP    'o'         ;Out to DOS
      JR    Z,CMD_O
      CP    'c'         ;Single step with CALL?
      JR    Z,CMD_C
      CP    'd'         ;Display AAAA <space>
      JR    Z,CMD_D
      CP    'i'         ;Single step?
CMD_C JP    Z,CMD_CI
      CP    'a'         ;ASCII modify memory?
      JP    Z,CMD_AH
      CP    'h'         ;Hex modify memory AAAA?
      JP    Z,CMD_AH
      CP    'r'         ;Modify reg pair RP DDDD?
      JP    Z,CMD_R
      CP    'u'         ;Dynamic display update?
      JR    Z,CMD_U
      CP    'x'         ;Display register format?
      JP    NZ,BLOCK    ;Try extra commands
;
;     Command X - Normal display mode
;
CMD_X XOR   A
CMD_S LD    (SAVTWO),A  ;Show not full screen
      RET
;
;     Command U - continuously update display
;
CMD_U CALL  @KBD        ;Scan keyboard
      OR    A           ;Character entered?
      RET   NZ          ;Return to CMND if so
      CALL  WRREGS            ;  else refresh display
      JR    CMD_U       ;  & loop
;
;     Command D - Display memory at address NNNN
;
CMD_D CALL  HEXIN@
      RET   Z           ;Ret to CMMD if no char
      JR    $?6         ;  else set DSPADR to
                        ;  new address in HL
;
;     Command ; - Increment memory display one block
;
CMD_INC     LD    BC,64       ;Init for 64-byte block
$?4   LD    HL,(DSPADR) ;P/u current display addr
      LD    A,(SAVTWO)  ; =0 -> Normal display addr
                        ;<>0 -> Full disp mode
      OR    A
      JR    Z,$?5
      LD    C,0         ;Zero out low order to
                        ;  provide inc or dec of
                        ;  256 bytes (full disp)
      LD    A,B         ;B=00 -> inc 1 page,
      OR    A           ;  make BC = 256
      JR    NZ,$?5            ;B=FF -> Dec 1 page,
      INC   B           ;  just add
$?5   ADD   HL,BC       ;HL now points to
$?6   LD    (DSPADR),HL ;  new display address
      RET
;
;     Command - - Decrement memory display 1 block
;
CMD_DEC     LD    BC,0FFC0H   ;Init to 64-byte dec
      JR    $?4
;
;     Command O - Exit to DOS
;
```

```
CMD_O CALL  INPUT@            ;Fetch valid terminator
      RET   NC          ;Back if bad char
      JP    @EXIT       ;Else exit to DOS
;
;     Register display routine
;
WRREGS:
      LD    A,1CH       ;Home the cursor
      CALL  @DSP
      IF    @MOD4
      LD    A,15        ;Turn off the cursor
      CALL  @DSP
      ENDIF
      LD    A,(SAVTWO)  ;0 = Normal display mode
      OR    A           ;<> 0 = Full display mode
      JR    NZ,FULDSP   ;No reg display if FULL
      LD    HL,AFREG    ;Pt to register save area
      PUSH  HL
      LD    HL,REGTBL   ;Pt to reg symbol table
      LD    B,12        ;Init for 12 registers
$?8   CALL  WR3BYT            ;Write 3-character symbol
      EX    (SP),HL           ;Exchange reg save ptr
      LD    E,(HL)            ;Place reg value -> DE
      INC   HL
      LD    D,(HL)
      INC   HL          ;Place next reg save
      PUSH  HL          ;  pointer on the stack
      EX    DE,HL       ;Reg value -> HL
      LD    A,'='
      CALL  @DSP
      CALL  WRSPA@
      LD    A,H         ;Write hi-order byte
      CALL  WRHEX
      LD    A,L         ;Write lo-order byte
      CALL  WRHEX
      LD    A,B         ;Get loop counter &
      AND   0BH         ;  ck if 12 => AF pair
      CP    08H         ;  or if 8 => AF' pair
      JR    NZ,NOFLG    ;Bypass if not flag reg
      LD    C,L         ;Transfer 'F' reg to C &
      PUSH  BC          ;  save the loop counter
      LD    HL,FLGTBL   ;Pt to flag syMbol table
      LD    B,8         ;Init for 8 bits
$?9   SLA   C           ;Shift a bit into carry
      LD    A,(HL)            ;P/u flag table character
      JR    C,$?10            ;Use table char if bit on
      LD    A,'-'       ;  else use a dash
$?10  CALL  @DSP
      INC   HL          ;Next flag table char
      DJNZ  $?9         ;Loop for 8 flag bits
      POP   BC          ;Get main loop counter
      LD    A,61+0C0H   ;Tab 60 to put cursor
      CALL  @DSP        ;  on next line
      JR    $?11
NOFLG CALL  WRMEM
$?11  POP   HL          ;Get next reg save ptr
      EX    (SP),HL           ;Excg with next reg symbol
      DJNZ  $?8         ;Loop end
      POP   HL          ;Get reg save ptr (fini)
      LD    HL,(DSPADR) ;P/u memory disp address
      LD    B,4         ;Init for 4 lines
$?12  LD    A,6+0C0H    ;Tab 6 spaces
      CALL  @DSP
      CALL  WR2HEX@           ;Write the memory address
      CALL  WRSPA@            ;Write a space
      CALL  WRMEM       ;Write a line of memory
      DJNZ  $?12        ;Loop until 4 or 16
      LD    A,1FH       ;Clear to end-of-frame
      JP    @DSP
FULDSP      LD    HL,(DSPADR) ;P/u display address
      LD    L,0         ;Round to multiple of 256
      LD    B,16        ;Init for 16 lines
      JR    $?12
;
;     Register symbol table
;
REGTBL      DB    'af bc de hl af''bc''de''hl''ix iy sp pc '
;
;     Flag register bit symbol table
;
FLGTBL      DB    'SZ1H1PNC'
;
;     Command G - Go to memory address NNNN,
;       with optional breakpoints
;
CMD_G LD    B,2         ;Init for maximum of
      LD    DE,NXTBYT   ;  two breakpoints
      CALL  HEXIN@            ;Get exec address
      JR    Z,$?13            ;Go on end
      LD    (PCREG),HL  ;  else save new start
$?13  JR    C,$?14            ;Go if <ENTER> used
      CALL  HEXIN@            ;Get a breakpoint
      PUSH  AF
      CALL  NZ,$?17           ;Set if brkpt entered
      POP   AF
      DJNZ  $?13
$?14:
      XOR   A
      LD    (@DBGHK),A  ;Init DEBUG on
;
;     This next section of code picks up the register
;     save arrea, pushes the save area onto the stack,
;     the pops out into the correct reg assignments.
;
$?15  LD    HL,REGSAV   ;End of reg save area
      LD    B,11        ;Init for 11 regs
$?16  LD    D,(HL)
      DEC   HL
      LD    E,(HL)
      DEC   HL
      PUSH  DE
      DJNZ  $?16
      POP   AF          ;Now pop the registers
      POP   BC
      POP   DE
      POP   HL
      EX    AF,AF'
      EXX
      POP   AF
      POP   BC
      POP   DE
      POP   HL
      EX    AF,AF'
      EXX
      POP   IX
      POP   IY
      POP   HL
      LD    SP,HL
      LD    HL,(PCREG)  ;Init the branch address
      PUSH  HL
      LD    HL,(HLREG)
      RET               ;Go to branch
;
;     This next routine will insert an RST 48 inst into
;     the target of a single-step or breakpoint
;     providing the target address is a RAM location.
;     If it is, the target byte and its address are
;     saved in one of the instruction save areas.
;     If the target address is ROM or nonexistent, a
;     branch to command INPUT routine is taken instead
;     of the pending operation.
;
$?17  LD    A,(HL)            ;Save byte of next inst
      LD    (DE),A
      DEC   DE
      LD    A,0F7H            ;Insert RST 48 into
      LD    (HL),A            ;  next INST address
      CP    (HL)        ;Ck if RAM/ROM/no memory
      JP    NZ,$?1            ;Go to command if not RAM
      LD    A,H         ;Is RAM, save address of
      LD    (DE),A            ;  insertion into buffer
      DEC   DE          ;  pointed to byuu DE, DE-1
      LD    A,L
      LD    (DE),A
      DEC   DE
      RET
;
;     Commands A & H - Modify address NNNN to XX
;       <SPACE> increments address
;
CMD_AH      LD    (SAVONE),A  ;Save enttry condition
      LD    HL,(NXTADR) ;Default to current mod addr
      CALL  HEXIN@
$?18  LD    (NXTADR),HL ;Adjust addr for mod
      RET   C           ;Return on <ENTER)
      PUSH  HL
      CALL  WRREGS
      LD    HL,13<8!0   ;Cursor to 13,0
      LD    B,3
      LD    A,15        ;Svc @VDCTL set cursor
      RST   28H
      LD    HL,(NXTADR) ;P/u mod address again
      CALL  WR2HEX@           ;Wtie the address & save
      PUSH  HL          ;  the mod addr again
      LD    HL,14<8!0   ;Cursor to 14,0
      LD    B,3
      LD    A,15        ;Svc @VDCTL set cursor
      RST   28H
      POP   HL          ;Recover mod addr
      CALL  AHDSP
      LD    A,'-'
      CALL  @DSP
      POP   DE          ;Recover mod addr in DE
      CALL  AHGET
      EX    DE,HL       ;Switch mod addr/value
      JR    Z,$?19            ;Bypass change on <SPACE>
      LD    (HL),E            ;Insert new val in memory
$?19  RET   C           ;To CMND on non-digit
      INC   HL          ;  else increment address
      JR    $?18        ;  pointer & loop
AHDSP LD    A,(SAVONE)
      CP    'a'
      JP    NZ,WR1HEX@  ;Write (HL) & bump H
DSPASC@     LD    A,(HL)            ;Else write in ASCII
      CP    20H         ;Convert non-displayable
      JR    C,TYP3            ;  values to '.'
      CP    0C0H
      JR    C,TYP4
TYP3  LD    A,'.'
TYP4  JP    @DSP
AHGET LD    A,(SAVONE)
      CP    'a'
      JP    NZ,HEXIN@
GETASC@     PUSH  HL          ;Provide lower/upper
      LD    HL,INPUC@+1 ;  case entry in type
      LD    (HL),6FH    ;  by modifying sys5 code
      CALL  INPUT@
      LD    (HL),0EFH   ;Restore the UC -> lc
      POP   HL          ;  conversion
      LD    L,A
      RET
;
;     Command R - Load register pair RP with NNNN
;
CMD_R CALL  INPUT@            ;Get 1st symbol char
      RET   Z           ;Return if end
      LD    C,A         ;  else save char in C
      CALL  INPUT@            ;Get 2nd symbol char
      RET   Z           ;Return if end
      LD    D,A         ;  else save char in D
      LD    E,' '       ;Init for space
      CALL  INPUT@            ;Get 3rd symbol char
      RET   C           ;Return on end
      JR    Z,$?20            ;Bypass if not primed
      LD    E,A         ;  else put "'" into E
      CALL  INPUT@            ;Ck for space separator
      RET   NZ          ;Return if none
      RET   C
```

```
$?20  LD    HL,REGTBL   ;Register symbol table
      LD    B,12        ;Init for 12 registers
$?21  LD    A,(HL)            ;Match first symbol?
      CP    C
      JR    Z,$?24            ;If a match, test 2nd
      INC   HL          ;  else pt to next reg
$?22  INC   HL
$?23  INC   HL
      DJNZ  $?21        ;Loop for 12 regs
      RET               ;Return if no match
$?24  INC   HL          ;Pt to 2nd table char
      LD    A,(HL)            ;  & p/u the symbol
      CP    D           ;Ck the 2nd char input
      JR    NZ,$?22           ;-> next if no match
      INC   HL          ;Match, ck 3rd reg symbol
      LD    A,(HL)            ;P/u the 3rd table symbol
      CP    E           ;  & compare with input
      JR    NZ,$?23           ;-> next if no match
      LD    A,18H       ;Convert counter to index
      SUB   B           ;  into reg save area
      SUB   B
      LD    C,A         ;Index into BC
      LD    B,0
      LD    HL,AFREG    ;Start of reg save area
      ADD   HL,BC       ;Add index to get pointer
      PUSH  HL          ;Save the pointer
      LD    A,1EH       ;Erase to end of line
      CALL  @DSP
      POP   DE          ;Recover pointer
      CALL  HEXIN@            ;Read in the new value
      RET   Z           ;No update if none
      EX    DE,HL       ;Exchg value/pointer
      LD    (HL),E            ;Insert new value into
      INC   HL          ;  register save area
      LD    (HL),D
      RET
;
;     Command I - Step one instruction at a time
;
CMD_CI      PUSH  AF          ;Save whether I or C
      LD    DE,(PCREG)  ;Point to inst address
      LD    A,(DE)            ;  & get it
      LD    HL,XY_TAB   ;IX,IY Table
      CP    0DDH        ;Is inst an IX?
      JR    Z,$?25
      CP    0FDH        ;Is inst an IY?
      JR    Z,$?25
      LD    HL,OP_TAB   ;All X IX, IY, & ED
      CP    0EDH        ;Is inst an ED?
      JR    NZ,$?26
      LD    HL,ED_TAB   ;ED Table
$?25  INC   DE          ;Get next byte for
      LD    A,(DE)            ;  IX, IY, and ED inst
      DEC   DE          ;Reset ptr to 1st byte
$?26  LD    C,A         ;Inst byte to reg C
;
;     This next section of code determines the length
;      of all instructions and whether they
;      are CALLs, JumPs, or RETurns.
;
$?27   LD     A,(HL)            ;P/u table value &
      AND    C           ;  strip off certain bits
      INC    HL          ;Pt to table code
      CP     (HL)        ;If a match, the inst is
      INC    HL          ;  fully decoded as to
      JR     Z,$?28            ;  length & type by the
      INC    HL          ;  next byte
      LD     A,(HL)            ;Ck for table end
      CP     5
      JR     NC,$?27
$?28   LD     A,(HL)            ;Get control/length byte
      LD     B,A         ;  into reg B
      AND    0FH         ;Strip off the control
      LD     L,A         ;Put length into reg L
      LD     H,0         ;Zero out reg H
      ADD    HL,DE       ;Next address into HL
      PUSH   DE          ;This addr in DE saved
      LD     DE,NXTBYT   ;Buffer area
      CALL   $?17        ;Insert RST 48 if RAM
      POP    HL          ;Get this inst address
      LD     A,B         ;Get control/length byte
      AND    0F0H        ;Strip off length
      JR     Z,$?29            ;Go if regular inst
      INC    HL
      CP     20H
      JR     C,$?34            ;Branch if 'JP (HL)'
      JR     Z,$?33            ;Go if 'JP (IX/IY)'
      CP     40H
      JR     C,$?32            ;Go if 'JR' or 'DJNZ'
      JR     Z,$?31            ;Branch if 'JP' inst
      CP     60H
      JR     C,$?30            ;Branch if 'RET' inst
      JR     Z,$?28A           ;Branch if CALL inst
      LD     A,C         ;  else calc target of
      AND    38H         ;  the RST inst
      LD     L,A
      LD     H,0
      POP    AF          ;Rcvr entry command
      CP     'c'
      JR     Z,$?29            ;Go in "call" mode
      LD     A,L         ;Must check RST for
      CP     5<3         ;  40, 48, 56 inhibit
      JR     NC,$?29           ;Convert to CALL
      JR     $?35        ;  else single step
$?28A  POP    AF          ;Recover entry command
      CP     'i'         ;Was command an 'I'
      JR     Z,$?31            ;Go for 'CALLs' if 'I'
$?29   JP     $?15        ;Go for 'CALLs' if 'C'
$?30   LD     HL,(SPREG)  ;RET inst, p/u RET addr
$?31   LD     A,(HL)            ;JP inst, p/u jump addr &
      INC    HL          ;  insert into reg HL
      LD     H,(HL)
      LD     L,A
      JR     $?35
```

```
$?32  LD    C,(HL)            ;'JR' or 'DJNZ', get 'E'
      LD    A,C         ;Make A=0 if C is
      RLCA              ;  positive, else make
      SBC   A,A         ;  A=FF for negative
      LD    B,A         ;Put -> B, FF if 'E' neg
      INC   HL          ;  or 0 if 'E' pos.
      ADD   HL,BC       ;Add the displacement
      JR    $?35
$?33  LD    HL,(IXREG)  ;Init for JP (IX)
      BIT   5,C         ;Test inst for DD/FD
      JR    Z,$?35            ;Bit 5 off = DD
      LD    HL,(IYREG)  ;JP (IY), p/u jump addr
      JR    $?35
$?34  LD    HL,(HLREG)  ;JP (HL), p/u jump addr
$?35  CALL  $?17
      JR    $?29
;
;     The next three tables are used to determine
;     length & instruction type for all instructions
;     used in the single-step mode. Table format uses
;     three bytes for each decoding process. The 1st
;     byte is ANDed with the inst byte to strip off
;     selected bits and include others. The result is
;     compared to the next table byte (test byte) for
;     a match. If matched, then the inst byte has been
;     identified as to its class & length. The 3rd byte
;     denotes the class and length as follows:
; High order nybble
;       0 = Regular instruction
;       1 = JP (HL) instruction
;       2 = JP (IX) of JP (IY) instruction
;       3 = JR or DJNZ instructions
;       4 = JP instructions
;       5 = RET instructions
;       6 = CALL instructions
;       7 = RST instructions
; Low order nybble = the length
;     The last byte of each table is the length of
;     all other instructions.
;
;     Table for regular instruction (no IX, IY, ED)
;
OP_TAB      DB    0C7H,0C0H,51H     ;C8, D8, E8, F8
      DB    0FFH,0C9H,51H     ;C9
      DB    0FFH,0E9H,11H     ;E9
      DB    0CFH,01H,03H      ;01, 11, 21, 31
      DB    0E7H,22H,3  ;22, 2A, 32, 3A
      DW    0C2C7H            ;C2, C1, D2, DA, E2, EA,
      DB    43H         ;  F2, FA
      DB    0FFH,0C3H,43H     ;C3
      DW    0C4C7H            ;C4, CC, D4, DC, E4, EC,
      DB    63H         ;  F4, FC
      DB    0FFH,0CDH,63H     ;CD
      DW    06C7H       ;06, 0E, 16, 1E, 26, 2E
      DB    02H         ;  36, 3E
      DB    0F7H,0D3H,02      ;D3, DB
      DW    0C6C7H            ;C6, CE, D6, DE, E6, EE,
      DB    02H         ;  F6, FE
      DB    0FFH,0CBH,2 ;All CB instructions
      DB    0F7H,10H,32H      ;10, 18
      DB    0E7H,20H,32H
      DB    0C7H,0C7H,71H     ;RST instructions
      DB    1           ;All others are 1-byte
;
;     Next table is for ED - extended instructions
;
ED_TAB      DB    0C7H,43H,04H      ;43, 4b, 53, 5B, 73, 7B
      DB    0F7H,45H,52H      ;45, 4D
      DB    2           ;All other ED are 2-byte
;
;     IX, IY Index instructions table
;
XY_TAB      DB    0FEH,34H,03 ;34, 35
      DB    0C0H,40H,03 ;4X, 5X, 6X, 7X (X = 0-F)
      DB    0C0H,80H,03 ;8X, 9X, AX, BX (X = 0-F)
      DB    0FFH,21H,04 ;21
      DB    0FFH,22H,04 ;22
      DB    0FFH,2AH,04 ;2A
      DB    0FFH,36H,04 ;36
      DB    0FFH,0CBH,04      ;CB
      DB    0FFH,0E9H,22H     ;E9
      DB    02H         ;All others are 2-byte
;
;     Routine to display memory on CRT screen
;
WRMEM PUSH  BC          ;Save main counter 4/16
      LD    A,'='
      CALL  @DSP
      INC   A           ;'>'
      CALL  @DSP
      LD    B,16        ;Init for 16 lines
      PUSH  HL          ;Save memory pointer
$?36  CALL  GRPHIC            ;Ck if need graphic bars
      CALL  WR1HEX@           ;Call on HEX display only
      DJNZ  $?36        ;Loop until full line
      POP   HL          ;Rcvr memory pointer
;
;     Now write the line in ASCII
;
      CALL  WRSPA@
      LD    B,16
$?37  CALL  $?41        ;Space after 8th
      LD    A,(HL)            ;P/u the byte -> reg A
      CP    20H         ;Repl controls with '.'
      JR    C,$?38
      CP    0C0H        ;Tabs/specials with '.'
      JR    C,$?39
$?38  LD    A,'.'
$?39  CALL  @DSP
      INC   HL          ;Bump memory address
      DJNZ  $?37
      POP   BC          ;Get line counter
      RET
;
```

```
;      This routine determines if veritical graphic
;      bars should be surrounding the current character
;
GRPHIC       LD    DE,(NXTADR) ;P/u modification address
       INC   DE          ;  & increment it
       PUSH  HL          ;Save current memory
       XOR   A           ;  display address
       SBC   HL,DE       ;Ck if mod addr=disp addr
       IF    @MOD4
       LD    A,95H       ;Graphic left bar
       ENDIF
       IF    @MOD2
       LD    A,15H
       ENDIF
       JR    Z,$?40            ;Insert graphic if equal
       CALL  $?41        ;Not =, insert space if
       INC   HL          ;  between pos 8 & 9
       LD    A,L         ;Result is zero if next
       OR    H           ;  char address is also
                         ;  the display address
       POP   HL          ;Get current mem disp adr
       IF    @MOD4
       LD    A,0AAH            ;Graphic right bar output
       JP    Z,@DSP            ;Go if yes
       JR    $?42        ;  else continue
       ENDIF
       IF    @MOD2
       JR    NZ,$?42           ;Go if not
       XOR   A           ;  lead in
       CALL  @DSP        ;Init video lead in
       LD    A,15H
       JP    @DSP        ;  and display
       ENDIF
$?40   EQU   $
       IF    @MOD2
       PUSH  AF
       XOR   A
       CALL  @DSP        ;Lead in code
       POP   AF          ;Restore
       ENDIF
       CALL  @DSP        ;Display char
       POP   HL          ;Recover current display
$?41   LD    A,B         ;  address & output a
       CP    8           ;  space if between the
       RET   NZ          ;  8th & 9th bytes
$?42   JR    WRSPA@            ;  else just return
;
;      This routine will return with zero flag set
;      on entry of a comma or a SPACE.  Entry of <ENTER>
;      will set carry flag and return
;
INPUT@       PUSH  DE
$?43   CALL  @KEY
       CP    0DH         ;ENTER?
       JR    Z,$?44
       CP    20H         ;Get another char if
       JR    C,$?43            ;  entry was control
```

```
INPUC@      SET   5,A         ;Cvrt UC to lc
      CALL  @DSP        ;Not control, disp it
      POP   DE
      CP    ','         ;Return with zero flag
      RET   Z           ;  set if a comma
      CP    ' '         ;Return with zero flag
      RET               ;  set if <SPACE>
$?44  POP   DE
      SCF               ;<ENTER> will set
      RET               ;  the carry flag
;
;     This routine will read in digits
;     and convert them to binary
;
HEXIN@      CALL  INPUT@            ;Get char and return on
      RET   Z           ;  SPACE, COMMA, or ENTER
      LD    HL,0        ;Init value to zero
$?45  CALL  CVB         ;Convert to binary if ok
      JP    C,CMND            ;  else back on bad digit
      ADD   HL,HL       ;Multiply current value
      ADD   HL,HL       ;  by 16 and insert the
      ADD   HL,HL       ;  new digit into the
      ADD   HL,HL       ;  lo-order nybble of L
      OR    L
      LD    L,A
      CALL  INPUT@            ;Get another character
      JR    NZ,$?45           ;Go if not separator
      RRA               ;Force <ENTER> to set
      ADC   A,81H       ;  the carry flag
      RET
;
;     Routine to convert expected ASCII hex digit to
;     its binary value. Set Carry-flag on bad digit
;
CVB   SUB   '0'         ;Convert digit to binary
      RET   C           ;Error if < '0'
      ADD   A,0C9H            ;Ck for > F (46H-30H=16H)
                        ;  (16H + E9H = FFH)
      RET   C           ;Error if > ASCII 'F'
      ADD   A,6         ;(E9H-EFH) to (EFH-05H)
      JR    C,ATOF            ;Carry denotes was <A-F>
      ADD   A,27H       ;(EFH-FFH) to (F6H-06H)
      RET   C           ;Error if (3AH-3FH/:-?)
ATOF  ADD   A,0AH       ;(00D-06D) to (10D-16D)
                        ;  or (F6H-FFH) to (0-9)
      OR    A           ;Set zero flag on zero
      RET
;
;     Routine to write one byte as two hex digits
;
WR1HEX@     LD    A,(HL)
      INC   HL
      JR    CV2HEX@
;
;     Routine to write 2 bytes (HL) as 4 hex digits
;
WR2HEX@     LD    A,H
```

```
      CALL  CV2HEX@
      LD    A,L
;
;     Routine converts a byte to 2 hex digits
;
CV2HEX@     PUSH  AF          ;Save the byte in A
      RRA               ;Move hi-order
      RRA               ;  into lo-order
      RRA
      RRA
      CALL  $?46        ;Strip off hi-order
                        ;  & convert to ASCII
      POP   AF          ;Recover the byte
$?46  AND   0FH         ;Strip off hi-order
                        ;  & convert to ASCII
      ADD   A,90H
      DAA
      ADC   A,40H
      DAA
$?47  JP    @DSP
;
;     Miscellaneous routines
;
WRHEX CALL  CV2HEX@
WRSPA@      LD    A,20H
      JR    $?47
;
WR3BYT      CALL  $?48
      CALL  $?48
$?48  LD    A,(HL)
      INC   HL
      JR    $?47
;
;     Command B - Block move
;
BLOCK CP    'b'
      JR    NZ,FILL
      LD    HL,(DSPADR) ;'b'lock move s,d,len
      CALL  HEXIN@            ;Default to display addd
      RET   C           ;Back on <ENTER>
      LD    (DSPADR),HL ;Save start addr
      JR    NZ,BLO1           ;Go if start entered
      CALL  WR2HEX@           ;  else show default
      LD    A,','
      CALL  @DSP
BLO1  LD    HL,(NXTADR) ;Default next address
      CALL  HEXIN@
      LD    (NXTADR),HL ;Save dest address
      JR    NZ,BLO2           ;Go if entered
      PUSH  AF
      CALL  WR2HEX@           ;  else show default
      LD    A,','
      CALL  @DSP
      POP   AF
BLO2  LD    HL,256            ;Default length to 256
      JR    C,BLO3            ;Go if <ENTER> used prev.
      CALL  HEXIN@            ;Get new length
      JR    NZ,BLO4           ;Go if entered
BLO3  PUSH  HL
      CALL  WR2HEX@           ;  else dsply default
      POP   HL
BLO4  LD    B,H         ;Length to BC
      LD    C,L
      LD    HL,(DSPADR) ;Set source
      LD    DE,(NXTADR) ;  and dest
      LDIR
      LD    (NXTADR),DE ;Set new mod addr
      RET
;
;     'f'ill aaaa,bbbb,cc
;
FILL  CP    'f'
      JR    NZ,JUMP
      CALL  HEXIN@            ;Get starting address
      RET   Z
      PUSH  HL          ;Save starting address
      CALL  HEXIN@            ;Get ending address
      EX    (SP),HL           ;Place ending into BC
      POP   BC          ;  & starting into HL
      RET   Z
      PUSH  HL          ;Save starting again
      CALL  HEXIN@            ;Get fill character
      LD    E,L         ;Save fill in E
      POP   HL          ;Recover starting addr
      RET   Z
      XOR   A           ;Clear the C-flag
FIL1  PUSH  HL
      SBC   HL,BC
      POP   HL
      RET   NC          ;Return when start = end
      LD    (HL),E            ;Stuff char into memory
      INC   HL
      JR    FIL1
;
;     'j'ump over next instruction
;
JUMP  CP    'j'
      JR    NZ,QUERY
      LD    HL,(PCREG)  ;Get current PC location
      INC   HL          ;  and increment it
      LD    (PCREG),HL
      RET
;
;     'q'uery ii - 'q'uery oo,dd
;     input/output to port
;
QUERY CP    'q'
      JR    NZ,DISKIO
      LD    A,1EH       ;Clear to end of line
      CALL  @DSP
      CALL  HEXIN@            ;Get port number
      RET   Z           ;Back if no value
      LD    C,L
      JR    C,QUE1            ;If <ENTER>, do input
      CALL  HEXIN@            ;Get byte to output
      RET   Z           ;Quit if none
      OUT   (C),L       ;Do the output
      RET
QUE1  LD    A,'='       ;Dsply separator
      CALL  @DSP
      IN    A,(C)       ;Read the port and
      CALL  CV2HEX@           ;  dsply the value
      JP    INPUT@
;
;     If a command is entered and not found in SYS5,
;     SYS9 will be searched if the extended debugger
;     is active.
;
EXTDBG      LD    HL,(EXTDBG$)      ;Try extended debug
      JP    (HL)
;
;     Disk I/O - d,c,s,r/w/*,addr,lngth
;
DISKIO      SUB   30H         ;Cnvrt drive to binary
      CP    8           ;Check on max drive
      JR    NC,EXTDBG   ;Exit if not <0-7>
      LD    C,A         ;Xfer drive # to reg C
      CALL  @GTDCT            ;  & get the DCT
      LD    A,(IY+7)    ;Get sectors/cyl & heads
      AND   0E0H        ;Remove sectors/cyl
      RLCA              ;  & keep # of heads
      RLCA              ;Shift into bits 0-2
      RLCA
      INC   A           ;Adj for 0 offset
      LD    B,A
      LD    A,(IY+7)    ;# of sectors per cyl
      AND   1FH         ;Remove heads
      INC   A           ;Adj for zero offset
      LD    H,A
      XOR   A           ;Accumulate total # of
DIS1  ADD   A,H         ;Sectors per cyl
      DJNZ  DIS1
      BIT   5,(IY+4)    ;Test if 2-sided drive
      JR    Z,DIS2
      ADD   A,A         ;Times 2 if 2-sided
DIS2  LD    (SAVTWO+1),A      ;Save sectors per cyl
      LD    A,1EH       ;Clear to end of line
      CALL  @DSP
      CALL  INPUT@            ;Input CYL #
      RET   C
      CALL  HEXIN@            ;  cyl in hex
      RET   C
      LD    D,L         ;Cylinder entered?
      JR    NZ,DIS3
      LD    D,(IY+9)    ;P/u directory cyl
DIS3  CALL  HEXIN@            ;Sec in hex
      LD    E,L         ;Sector entered?
      LD    A,1         ;Init to 1 sector i/o
      JR    NZ,DIS4
      LD    E,0         ;Default to sector 0
      LD    A,(SAVTWO+1)      ;Default to total sectors
```

```
DIS4  LD    (NXTBYT),A
      RET   C
      CALL  INPUT@            ;Get I/O direction (R,W,*)
      RET   C
      LD    B,A         ;Save I/O char in B
      CALL  INPUT@            ;Get buffer I/O address
      RET   C
      CALL  HEXIN@
      PUSH  HL          ;Save buffer address
      JR    C,DIS6
      PUSH  HL
      CALL  HEXIN@            ;Sector count entered?
      LD    A,L
      POP   HL
      JR    Z,DIS6            ;Go if no sector count
      LD    (NXTBYT),A  ;Else update count
DIS6  LD    A,B         ;P/u I/O direction
      CP    'r'         ;Read?
      JR    Z,DIS9
      CP    'w'         ;Write?
      JR    Z,DIS10
      CP    '*'         ;Write System sector?
      JR    Z,DIS11
DIS7  INC   H           ;Bump up a buffer page
      INC   E           ;Bump sector number
      LD    A,(SAVTWO+1)      ;P/u max # sectors
      DEC   A           ;Compare max to where
      CP    E           ;  we are
      JR    NC,DIS8           ;Jump if more on cyl
      LD    E,0         ;Reset sector # to 0
      INC   D           ;Bump cylinder
DIS8  LD    A,(NXTBYT)  ;Reduce I/O sector count
      DEC   A
      LD    (NXTBYT),A
      JR    NZ,DIS6           ;Loop if not through
DIS8A POP   HL          ;Rcvr buffer start addr
      LD    A,B         ;P/u i/o direction
      CP    'r'         ;Read?
      RET   NZ          ;Ret if not read
      LD    L,0         ;Reset memory buffer ptr
      LD    (DSPADR),HL ;  to display the 1st
      LD    (NXTADR),HL ;  sector read
      LD    A,'s'       ;Set full screen mode
      LD    (SAVTWO),A
      RET
;
DIS9  EQU   $
      PUSH  HL
      PUSH  DE
      PUSH  BC
      LD    D,H         ;Pass buffer to DE
      LD    E,L
      INC   DE          ;Start +1
      LD    (HL),0            ;Clear a byte
      LD    BC,255            ;Length - 1
      LDIR              ;Clear buffer
      POP   BC          ;Unstack
      POP   DE
      POP   HL
;
      CALL  @RDSEC            ;Read the sector
      JR    Z,DIS7            ;Loop on read ok
      CP    6           ;  or directory read
      JR    Z,DIS7
      JR    DIS12       ;  else error
DIS10 CALL  @WRSEC            ;Write sector
      JR    Z,DIS7            ;Loop on write ok
      JR    DIS12
DIS11 CALL  @WRSSC            ;Write system sector
      JR    Z,DIS7            ;Loop on write prot ok
;
;     disk I/O/ error output display routine
;
DIS12 PUSH  DE          ;Save track & sector
      PUSH  AF          ;Save error code
      CALL  WRSPA@            ;Output a space
      LD    A,'*'
      CALL  @DSP        ;  followed by asterisk
      POP   AF
      CALL  CV2HEX@           ;Write error code #
      LD    A,'*'
      CALL  @DSP        ;  followed by space
      CALL  INPUT@            ;Continue?
      POP   DE          ;Rcvr track/sector
      JR    NC,DIS7           ;Loop unless <ENTER>
      JR    DIS8A       ;Exit on <ENTER>
LAST  EQU   $
      IF    LAST.GT.MAXCOR$-2
      ADISP 'ERROR: Module too big'
      ENDIF
      ORG   MAXCOR$-2
      DW    LAST-SYS5   ;Overlay size
      END
```

SYS5/EQU

```
;SYS5/EQU - Equates from cross reference of SYS5
      ADISP '<SYS5/EQU>'
;
$?1   EQU   1E32H
$?10  EQU   1F1DH
$?11  EQU   1F2EH
$?12  EQU   1F38H
$?13  EQU   1F8FH
$?14  EQU   1F9BH
$?15  EQU   1F9FH
$?16  EQU   1FA4H
$?17  EQU   1FC5H
$?18  EQU   1FDFH
$?19  EQU   200FH
$?2   EQU   1E37H
$?20  EQU   2057H
$?21  EQU   205CH
$?22  EQU   2061H
$?23  EQU   2062H
$?24  EQU   2066H
$?25  EQU   20A6H
$?26  EQU   20A9H
$?27  EQU   20AAH
$?28  EQU   20B7H
$?28A EQU   20F1H
$?29  EQU   20F6H
$?3   EQU   1E49H
$?30  EQU   20F9H
$?31  EQU   20FCH
$?32  EQU   2102H
$?33  EQU   210BH
$?34  EQU   2117H
$?35  EQU   211AH
$?36  EQU   2180H
$?37  EQU   218EH
$?38  EQU   219AH
$?39  EQU   219CH
$?4   EQU   1EB4H
$?40  EQU   21BFH
$?41  EQU   21C3H
$?42  EQU   21C7H
$?43  EQU   21CAH
$?44  EQU   21E1H
$?45  EQU   21EBH
$?46  EQU   2223H
$?47  EQU   222BH
$?48  EQU   223BH
$?5   EQU   1EC4H
$?6   EQU   1EC5H
$?8   EQU   1EEEH
$?9   EQU   1F16H
$A1   EQU   03B7H
$A2   EQU   03B8H
$A3   EQU   03B9H
```

```
$CKEOF      EQU   1470H
@$SYS EQU   08F0H
@@1   EQU   0000H
@@2   EQU   0000H
@@3   EQU   0000H
@@4   EQU   0000H
@ABORT      EQU   1B08H
@ADTSK      EQU   1CDAH
@BANK EQU   0877H
@BKSP EQU   1486H
@BREAK      EQU   196FH
@BYTEIO     EQU   1300H
@CHNIO      EQU   0689H
@CKBRKC     EQU   0553H
@CKDRV      EQU   1993H
@CKEOF      EQU   158FH
@CKTSK      EQU   1CF5H
@CLOSE      EQU   1999H
@CLS  EQU   0545H
@CMNDI      EQU   197EH
@CMNDR      EQU   197BH
@CTL  EQU   0623H
@DATE EQU   07A8H
@DBGHK      EQU   199FH
@DCINIT     EQU   19C0H
@DCRES      EQU   19C4H
@DCSTAT     EQU   19B5H
@DCTBYT     EQU   1A2BH
@DEBUG      EQU   19A0H
@DECHEX     EQU   03E1H
@DIRCYL     EQU   18F7H
@DIRRD      EQU   18BBH
@DIRWR      EQU   1803H
@DIV16      EQU   06E3H
@DIV8 EQU   1927H
@DODIR      EQU   19AFH
@DOKEY      EQU   19A9H
@DSP  EQU   0642H
@DSPLY      EQU   052DH
@ERROR      EQU   1B0FH
@EXIT EQU   1B0BH
@FEXT EQU   1984H
@FLAGS      EQU   196AH
@FNAME      EQU   199CH
@FRENCH     EQU   0000H
@FSPEC      EQU   1981H
@GATRD      EQU   1874H
@GATWR      EQU   1875H
@GERMAN     EQU   0000H
@GET  EQU   0638H
@GTDCB      EQU   1990H
@GTDCT      EQU   1A1EH
@GTMOD      EQU   19B2H
@HDFMT      EQU   19E4H
@HEX16      EQU   07BDH
@HEX8 EQU   07C2H
@HEXDEC     EQU   06F6H
```

```
@HIGH$      EQU   1948H
@HITRD      EQU   1897H
@HITWR      EQU   1898H
@HZ50 EQU   0000H
@ICNFG      EQU   0086H
@INIT EQU   198DH
@INTL EQU   0000H
@IPL  EQU   1BF2H
@JCL  EQU   0630H
@KBD  EQU   0635H
@KEY  EQU   0628H
@KEYIN      EQU   0585H
@KITSK      EQU   0089H
@KLTSK      EQU   1CD0H
@LOAD EQU   1B38H
@LOC  EQU   14B3H
@LOF  EQU   14DEH
@LOGER      EQU   0503H
@LOGOT      EQU   0500H
@MOD2 EQU   0000H
@MOD4 EQU   0FFFFH
@MSG  EQU   0530H
@MUL16      EQU   06C9H
@MUL8 EQU   190AH
@NMI  EQU   0066H
@OPEN EQU   198AH
@OPREG      EQU   0084H
@PARAM      EQU   1987H
@PAUSE      EQU   0382H
@PEOF EQU   14A2H
@POSN EQU   1434H
@PRINT      EQU   0528H
@PRT  EQU   063DH
@PUT  EQU   0645H
@RAMDIR     EQU   19ACH
@RDHDR      EQU   19D8H
@RDSEC      EQU   19F4H
@RDSSC      EQU   18D8H
@RDTRK      EQU   19E0H
@READ EQU   1513H
@REMOVE     EQU   19A6H
@RENAME     EQU   1996H
@REW  EQU   149BH
@RMTSK      EQU   1CD7H
@RPTSK      EQU   1CEBH
@RREAD      EQU   1473H
@RSLCT      EQU   19D4H
@RST00      EQU   0000H
@RST08      EQU   0008H
@RST10      EQU   0010H
@RST18      EQU   0018H
@RST20      EQU   0020H
@RST28      EQU   0028H
@RST30      EQU   0030H
@RST38      EQU   0038H
@RSTNMI     EQU   0FE9H
@RSTOR      EQU   19C8H
```

```
@RSTREG      EQU   0680H
@RUN  EQU   1B1DH
@RWRIT       EQU   13ADH
@SEEK EQU   19D0H
@SEEKSC      EQU   1421H
@SKIP EQU   1430H
@SLCT EQU   19BCH
@SOUND       EQU   0392H
@STEPI       EQU   19CCH
@TIME EQU   078DH
@USA  EQU   0FFFFH
@VDCTL       EQU   0B99H
@VDCTL3      EQU   0D38H
@VER  EQU   1560H
@VRSEC       EQU   19DCH
@WEOF EQU   14ECH
@WHERE       EQU   1979H
@WRITE       EQU   1531H
@WRSEC       EQU   19E8H
@WRSSC       EQU   19ECH
@WRTRK       EQU   19F0H
@_VDCTL      EQU   0D42H
ADDR_2_ROWCOL      EQU   0DF1H
AFLAG$       EQU   006AH
AUTO? EQU   1FF1H
BAR$  EQU   0201H
BOOTST$      EQU   439DH
BREAK?       EQU   1C60H
BRKVEC$      EQU   1C88H
BUR$  EQU   0200H
CASHK$       EQU   0A7BH
CFCB$ EQU   00E0H
CFGFCB$      EQU   00E0H
CFLAG$       EQU   006CH
CKMOD@       EQU   1A7FH
CKOPEN@      EQU   1568H
CMD_AH       EQU   1FD6H
CMD_C EQU   1E81H
CMD_CI       EQU   208BH
CMD_D EQU   1EABH
CMD_DEC      EQU   1EC9H
CMD_G EQU   1F82H
CMD_INC      EQU   1EB1H
CMD_O EQU   1ECEH
CMD_R EQU   203FH
CMD_S EQU   1E9DH
CMD_U EQU   1EA1H
CMD_X EQU   1E9CH
CMND  EQU   1E4CH
CONFIG$      EQU   203FH
CORE$ EQU   0300H
CRTBGN$      EQU   0F800H
CV2HEX@      EQU   221AH
CVB   EQU   2200H
CYL_GRN      EQU   16AEH
D@FBYT8      EQU   1A26H
DATE$ EQU   0033H
```

```
DAYTBL$      EQU   04C7H
DBGSV$       EQU   00A0H
DCBKL$       EQU   0031H
DCT$  EQU   0470H
DCTBYT8@     EQU   1A29H
DCTFLD@      EQU   1A34H
DFLAG$       EQU   006DH
DIRBUF$      EQU   2300H
DIS_DO_RAM  EQU   0846H
DODATA$      EQU   0B94H
DODCB$       EQU   0210H
DO_CONTROL  EQU   0C44H
DO_DSPCHAR  EQU   0CB8H
DO_INVERT_DIS      EQU   0C8CH
DO_INVERT_ENA      EQU   0C89H
DO_INVERT_OFF      EQU   0C9BH
DO_MASK      EQU   0000H
DO_RET       EQU   0BCBH
DO_RETI      EQU   0BCCH
DO_SCROLL    EQU   0CCEH
DO_TABS      EQU   0BEAH
DSKTYP$      EQU   04C0H
DSPASC@      EQU   201BH
DTPMT$       EQU   04C2H
DVREND$      EQU   0FF4H
DVRHI$       EQU   0206H
ED_TAB       EQU   2150H
EFLAG$       EQU   006EH
ENADIS_DO_RAM      EQU   0817H
EXTDBG$      EQU   19A4H
FDDINT$      EQU   000EH
FEMSK$       EQU   006FH
FLGTAB$      EQU   006AH
GETASC@      EQU   2031H
GET_@_ROWCOL       EQU   0DAEH
HERTZ$       EQU   0750H
HEXIN@       EQU   21E4H
HIGH$ EQU   040EH
HKRES$       EQU   1A6CH
IFLAG$       EQU   0072H
INBUF$       EQU   0420H
INPUC@       EQU   21D5H
INPUT@       EQU   21C9H
INTIM$       EQU   003CH
INTMSK$      EQU   003DH
INTVC$       EQU   003EH
JCLCB$       EQU   0203H
JDCB$ EQU   0024H
JFCB$ EQU   00C0H
JLDCB$       EQU   0230H
JRET$ EQU   0026H
KCK@  EQU   07D6H
KFLAG$       EQU   0074H
KIDATA$      EQU   08FCH
KIDCB$       EQU   0208H
LBANK$       EQU   0202H
LDRV$ EQU   0023H
```

```
LFLAG$      EQU   0075H
LNKFCB@     EQU   1566H
LOW$  EQU   001EH
LSVC$ EQU   000DH
MAXCOR$     EQU   2400H
MAXDAY$     EQU   0401H
MINCOR$     EQU   3000H
MODOUT$     EQU   0076H
MONTBL$     EQU   04DCH
NFLAG$      EQU   0077H
OPREG$      EQU   0078H
OPREG_SV_AREA     EQU   086EH
OPREG_SV_PTR      EQU   0835H
OP_TAB      EQU   211FH
ORARET@     EQU   14DCH
OSRLS$      EQU   003BH
OSVER$      EQU   0085H
OVRLY$      EQU   0069H
PAKNAM$     EQU   0410H
PAUSE@      EQU   0382H
PCSAVE$     EQU   07AFH
PDRV$ EQU   001BH
PHIGH$      EQU   001CH
PRDCB$      EQU   0218H
PUTA@DE     EQU   0DCDH
PUT_@ EQU   0DCAH
PUT_@_ROWCOL      EQU   0DC6H
RFLAG$      EQU   007BH
ROWCOL_2_ADDR     EQU   0DD0H
RST38@      EQU   1BFFH
RSTOR$      EQU   04C4H
RWRIT@      EQU   13A2H
S1DCB$      EQU   0238H
SBUFF$      EQU   1D00H
SET@EXEC    EQU   1A79H
SET_SCROLL  EQU   0CF3H
SFCB$ EQU   008CH
SFLAG$      EQU   007CH
SIDCB$      EQU   0220H
SODCB$      EQU   0228H
SPACE4$     EQU   2142H
STACK$      EQU   0380H
START$      EQU   0000H
SVCRET$     EQU   000BH
SVCTAB$     EQU   0100H
SYSERR$     EQU   1B13H
TCB$  EQU   004EH
TFLAG$      EQU   007DH
TIME$ EQU   002DH
TIMER$      EQU   002CH
TIMSL$      EQU   002BH
TIMTSK$     EQU   0713H
TMPMT$      EQU   04C3H
TRACE_INT   EQU   07B1H
TYP3  EQU   2024H
TYP4  EQU   2026H
TYPHK$      EQU   0A8FH
```

```
TYPTSK$      EQU   0B26H
USTOR$       EQU   0013H
VFLAG$       EQU   007FH
WR1HEX@      EQU   2211H
WR2HEX@      EQU   2215H
WRINT$       EQU   0080H
WRSPA@       EQU   2231H
XY_TAB       EQU   2157H
ZERO$ EQU   0401H
ZEROA@       EQU   13A0H
      END
```

SYS9/ASM

```
;SYS9/ASM - LS-DOS 6.2
      ADISP '<SYS9 - LS-DOS 6.2>'
;
*LIST OFF               ;Get SYS5/EQU
*REF  'SYS5/EQU:1'
*LIST ON
*GET  'COPYCOM:1'       ;Copyright message
      ORG   0A0H
;
SAVONE      DS    1
SAVTWO      DS    1
      DS    1           ;Space for saved byte (1)
NXTADR      DS    2
NXTBYT      DS    1
DSPADR      DS    2
AFREG DS    6           ;AF, BC, DE
HLREG DS    2           ;HL
      DS    8           ;AF', BC', DE', HL'
IXREG DS    2           ;IX
IYREG DS    2           ;IY
SPREG DS    1           ;SP
REGSAV      DS    1
PCREG DS    2           ;PC
;
      ORG   1E00H
;
SYS9  AND   70H
      RET   Z           ;Back on zero entry
      LD    HL,(EXTDBG$)      ;P/u hook address
      XOR   A;          ;See if already resident
      LD    DE,-ORARET@
      ADC   HL,DE       ;ADD does not affect Z
      RET   NZ          ;Ret if resident already
      LD    HL,(HIGH$)  ;Change high$ to provide
      LD    (DEBUGE+2),HL     ;Stuff last byte used
      LD    BC,LAST-DEBUGE    ;Room for relocating
      XOR   A           ;  this module to high
      SBC   HL,BC
      LD    (HIGH$),HL
      INC   HL          ;Pt to new entry point
      PUSH  HL          ;Save it for later
      EX    DE,HL       ;Move extended debug
      LD    HL,DEBUGE   ;  up to top of core
      LDIR
      POP   HL          ;Rcvr pointer to ent pt
      LD    (EXTDBG$),HL      ;  & reset sysres vector
      RET
;
;     Start of extended debug utility
;
DEBUGE      JR    NEXT
      DW    $-$
      DB    6,'EXTDBG'
      DW    0,0
```

```
;
;      'n'ext aaaa - position to next relative block
;      used in stepping through a program file
;      dumped to core in load module format
;
NEXT  CP    'n'-'0'
      JR    NZ,ENTER
      LD    HL,(NXTADR) ;Init if no further input
      CALL  HEXIN@            ;Argmt aaaa entered?
      INC   HL          ;Bump from type to length
      LD    D,0
      LD    E,(HL)            ;P/u block length
      LD    A,E
      CP    3           ;Len= 0,1,2?
      JR    NC,NEX1           ;If len= 0,1,2 (256-8)
      INC   D           ;  next block is +257-259
NEX1  INC   DE          ;Bump by one for len byte
      ADD   HL,DE       ;Add length to index
      LD    (NXTADR),HL ;Next block
      LD    A,L         ;Now set up the display
      AND   0C0H        ;Address
      LD    L,A
      LD    (DSPADR),HL
      RET
;
;      Enter hex data into memory
;
ENTER CP    'e'-'0'           ;'e'nter <addr>
      JR    NZ,LOCATE
      LD    HL,(NXTADR) ;Pt to current address
      CALL  HEXIN@            ;Get new address to enter
      LD    (NXTADR),HL
      RET   C           ;Back on <ENTER>
      JR    NZ,ENT1           ;Go if new addr
      CALL  WR2HEX@           ;  else dsply default
      CALL  WRSPA@
ENT1  LD    A,1EH       ;Clear the line
      CALL  @DSP
ENT2  CALL  WR1HEX@           ;Set up the display
      DEC   HL
      LD    A,'-'
      CALL  @DSP
      EX    DE,HL
      CALL  HEXIN@            ;Get the modify info
      EX    DE,HL
      JR    Z,ENT3            ;No change if no new data
      LD    (HL),E            ;  else update byte
ENT3  RET   C           ;Back if <ENTER> pressed
      INC   HL
      LD    (NXTADR),HL ;Index to next address
      JR    ENT2
;
;      'l'ocate aaaa,dd
;
LOCATE      CP    'l'-'0'
      JR    NZ,TYPE
      LD    HL,(NXTADR) ;Default current address
      INC   HL
      CALL  HEXIN@            ;Prompt new address
      LD    (NXTADR),HL
      JR    NZ,LOC1           ;Go if new addr
      PUSH  AF          ;Save flags
      CALL  WR2HEX@           ;Display default
      LD    A,','
      CALL  @DSP
      POP   AF
      LD    A,(NXTBYT)  ;P/u default byte
      LD    L,A
LOC1  JR    C,LOC2            ;Go if <ENTER> used
      CALL  HEXIN@            ;  else get new byte
      JR    Z,LOC2            ;Go if none entered
      LD    A,L
      LD    (NXTBYT),A  ;  else set byte to find
      JR    LOC3
LOC2  LD    A,L         ;Display byte info
      CALL  CV2HEX@
LOC3  LD    HL,(NXTADR) ;Set up for search
      LD    A,(NXTBYT)
      LD    BC,0        ;Set loop to 64K
      CPIR              ;Find a match
      RET   NZ          ;Back if none
      DEC   HL
      LD    (NXTADR),HL ;Store new mod addr
      LD    A,L
      AND   0C0H
      LD    L,A
      LD    (DSPADR),HL
      RET
;
;     't'ype aaaa - type ascii into memory
;
TYPE  CP    't'-'0'
      JR    NZ,VERIFY
      LD    HL,(NXTADR) ;Default current address
      CALL  HEXIN@            ;Prompt for new address
      LD    (NXTADR),HL
      RET   C           ;Back on <ENTER>
      JR    NZ,TYP1           ;Go if new addr
      CALL  WR2HEX@           ;  else dsply default
TYP1  LD    A,1EH       ;Clear to end of line
      CALL  @DSP
TYP2  CALL  WRSPA@
      CALL  DSPASC@           ;Display current contents
      LD    A,'-'
      CALL  @DSP
      PUSH  HL          ;Provide lower/upper
      CALL  GETASC@           ;  case entry
      POP   HL          ;  conversion
      RET   C
      CP    20H         ;Advance on space
      JR    Z,TYP5
      LD    (HL),A            ;Store new info
TYP5  INC   HL
      LD    (NXTADR),HL ;Advance the location
      JR    TYP2
;
;     'v'erify aaaa,bbbb,lngth - verify block
;
VERIFY      CP    'v'-'0'
      JR    NZ,WORD
      LD    HL,(DSPADR) ;1st default start of dsp
      CALL  HEXIN@            ;Prompt new start
      LD    (DSPADR),HL
      JR    NZ,VER1           ;Go if address entered
      PUSH  AF
      CALL  WR2HEX@           ;  else dsply default
      LD    A,','
      CALL  @DSP
      POP   AF
VER1  JR    C,VER2            ;Jump if <ENTER> used prev.
      LD    HL,(NXTADR) ;2nd default current mod addr
      CALL  HEXIN@            ;Prompt new 2nd start
      LD    (NXTADR),HL
      JR    NZ,VER2           ;Go if entered
      PUSH  AF
      CALL  WR2HEX@           ;  else dsply default
      LD    A,','
      CALL  @DSP
      POP   AF
VER2  LD    HL,0        ;Default length to verify
      JR    C,VER3            ;Go if <ENTER> used prev
      CALL  HEXIN@            ;Get new length
      JR    NZ,VER3           ;Go if new len entered
      PUSH  HL
      CALL  WR2HEX@           ;Dsply default len
      POP   HL
VER3  LD    B,H         ;Xfer length to BC
      LD    C,L
      LD    HL,(DSPADR) ;Set up for compare
      LD    DE,(NXTADR)
VER4  LD    A,(DE)
      CP    (HL)        ;Compare the two locations
      JR    NZ,VER5           ;Go on non-match
      INC   DE          ;  else inc pointers
      INC   HL          ;  and loop for length
      DEC   BC
      LD    A,B
      OR    C
      JR    NZ,VER4
VER5  LD    (NXTADR),DE ;Store non-match or end of
      LD    (DSPADR),HL ;  block
      RET
;
;     'w'ord aaaa,dddd - search for word dddd
;
WORD  CP    'w'-'0'
      JR    NZ,PRINT
      LD    HL,(NXTADR) ;Default current address
      INC   HL          ;  but bypass next word
      INC   HL
      CALL  HEXIN@            ;Get new start
      LD    (NXTADR),HL
      JR    NZ,WOR1           ;Go if value entered
      PUSH  AF          ;  else display default
      CALL  WR2HEX@
      LD    A,','
      CALL  @DSP
      POP   AF
      LD    A,(NXTBYT)  ;Get next default
      LD    L,A
      LD    A,(SAVTWO+1)
      LD    H,A
WOR1  JR    C,WOR2            ;Go if <ENTER>
      CALL  HEXIN@            ;Get next value
      JR    Z,WOR2            ;Go if default
      LD    A,L         ;Store new value
      LD    (NXTBYT),A
      LD    A,H
      LD    (SAVTWO+1),A
      JR    WOR3
WOR2  CALL  WR2HEX@           ;Display value
WOR3  LD    HL,(NXTADR) ;Start looking here
      LD    BC,0        ;Init count to 64K
WOR4  LD    A,(NXTBYT)
      CPIR              ;Find first match
      RET   NZ          ;Return if none
      LD    A,(SAVTWO+1)      ;Get 2nd half of word
      CP    (HL)        ;Is a match?
      JR    NZ,WOR4           ;Continue if not
      DEC   HL
      DEC   HL          ;Pt 1 byte before
      LD    (NXTADR),HL ;  and save that address
      LD    A,L
      AND   0C0H
      LD    L,A
      LD    (DSPADR),HL ;New display start
      RET
;
;     'p'rint aaaa,bbbb - print memory
;
PRINT CP    'p'-'0'           ;If command is not 'P',
PRI1  RET   NZ          ;  back to SYS5
      CALL  HEXIN@            ;Get start
      RET   Z           ;Back if no start addr
      PUSH  HL
      CALL  HEXIN@            ;Get end
      EX    (SP),HL
      POP   BC          ;Start in HL, end in BC
      RET   Z           ;Back if no end addr
      LD    A,L         ;Round to multiple of 16
      AND   0F0H
      LD    L,A
      LD    A,0DH       ;Send 2 blank lines to
      CALL  @PRT        ;  the printer
      CALL  @PRT
PRI2  PUSH  HL          ;Routine to write HL
      LD    A,H         ;  as 4 hex digits
      RRA
      RRA
      RRA
      RRA
      AND   0FH
      ADD   A,90H
      DAA
      ADC   A,40H
      DAA
      CALL  @PRT         ;1st one done
      LD    A,H
      AND   0FH
      ADD   A,90H
      DAA
      ADC   A,40H
      DAA
      CALL  @PRT         ;2nd one done
      LD    A,L
      RRA
      RRA
      RRA
      RRA
      AND   0FH
      ADD   A,90H
      DAA
      ADC   A,40H
      DAA
      CALL  @PRT         ;3rd one done
      LD    A,L
      AND   0FH
      ADD   A,90H
      DAA
      ADC   A,40H
      DAA
      CALL  @PRT         ;4th one done
      LD    A,20H        ;  & 2 spaces
      CALL  @PRT
      CALL  @PRT
      JR    PRI4
PRI3  JR    PRI2
;
;     Write a byte in hex
;
PRI4  LD    A,(HL)
      RRA
      RRA
      RRA
      RRA
      AND   0FH
      ADD   A,90H
      DAA
      ADC   A,40H
      DAA
      CALL  @PRT         ;Output it
      LD    A,(HL)
      AND   0FH
      ADD   A,90H
      DAA
      ADC   A,40H
      DAA
      CALL  @PRT        ;Output it
      LD    A,20H       ;  & a space
      CALL  @PRT
      INC   HL          ;Pt to next byte
      LD    A,L         ;Test multiple of 16
      AND   0FH
      JR    Z,PRI5
      AND   3           ;Space on multiple of 4
      LD    A,20H
      CALL  Z,@PRT
      JR    PRI4
PRI5  LD    A,20H       ;Space at end of 16
      CALL  @PRT
      POP   HL
PRI6  LD    A,(HL)            ;Print in ASCII if
      CP    20H         ;  printable; else
      JR    C,PRI7            ;  convert to '.'
      CP    80H
      JR    C,PRI8
PRI7  LD    A,'.'
PRI8  CALL  @PRT
      INC   HL          ;Loop until 16 chars
      LD    A,L
      AND   0FH
      JR    NZ,PRI6
      LD    A,0DH       ;  then a new line
      CALL  @PRT
      PUSH  HL
      LD    A,L         ;Check if HL is 0000
      OR    H
      JR    NZ,PRI9           ;  is OK > continue
      POP   HL
      JR    PRI10       ;Get OUT now
PRI9  XOR   A           ;Ck on finished
      SBC   HL,BC
      POP   HL
      JR    C,PRI3
PRI10 LD    A,0DH       ;3 new lines if done
      CALL  @PRT
      CALL  @PRT
      JP    @PRT
LAST  EQU   $
      IF    $.GT.DIRBUF$
      ADISP 'ERROR: Module too big'
      ENDIF
      ORG   MAXCOR$-2
      DW    LAST-SYS9   ;Overlay size
;
      END   SYS9
```

SYS10/ASM

```
;SYS10/ASM - LS-DOS 6.2
      ADISP '<SYS10 - LS-DOS 6.2>'
;
CR    EQU   13
;
*LIST OFF               ;Get SYS0/EQU
*REF  'SYS0/EQU:1'
*LIST ON
*GET  'COPYCOM:1'       ;Copyright message
;
      ORG   1E00H
;
SYS10 AND   70H         ;Strip bit 7
      RET   Z           ;Back on zero entry
      CP    10H         ;Remove all for now
      RET   NZ          ;Ret if any other entry
      LD    A,(DE)            ;Test device/file
      BIT   7,A         ;File open or device?
      JR    Z,CLOSDCB   ;Jump if device
      CALL  CKOPEN@           ;Test for remove access
      LD    A,(IX+1)    ;  & link the FCB to IX
      AND   7           ;Test for remove access
      CP    2
      JR    C,REMOV1    ;Jump if access granted
      LD    A,25H       ;"Illegal access ...
      OR    A           ;Set NZ error
      RET
REMOV1      LD    C,(IX+6)    ;P/u drive #
      LD    B,(IX+7)    ;P/u DEC
      CALL  @GATRD            ;Read GAT => DIRBUF$
REMOV2      CALL  Z,@DIRRD    ;Read dir for this DEC
      RET   NZ          ;Return if read errors
      LD    A,22        ;Point to 1st extent
      ADD   A,L
      LD    L,A
REMOV3      LD    E,(HL)            ;P/u relative cylinder
      INC   L
      LD    D,(HL)            ;P/u granule allocation
      LD    (EXTINFO+1),DE    ;Modify later instruction
      LD    A,E         ;Ck if extent in use
      CP    0FEH
      JR    NC,FIXDIR   ;Jump if not used
      INC   L
      CALL  RMVEXT            ;Deallocate ext from GAT
      JR    REMOV3            ;Loop to next extent
;
;     Deallocated last extent; clean up directory
;
FIXDIR      LD    A,L         ;Point to 1st byte
      AND   0E0H        ;  of DIR entry
      LD    L,A
      RES   4,(HL)            ;Show dir entry spare
      CALL  @DIRWR            ;Write the dir record
      CALL  Z,@HITRD    ;Grab HIT => SBUFF$
      LD    H,SBUFF$>8  ;Point to HIT entry
      LD    L,B         ;  & zero out DEC posn
      LD    (HL),0
      CALL  Z,@HITWR    ;Write HIT back to disk
      RET   NZ          ;Ret if read/write errors
EXTINFO     LD    DE,0        ;P/u last extent info
;
;     If extended directory record in use,
;     D -> DEC of FXDE record
;     E -> FE if FXDE, FF if extent unused
;
      LD    B,D         ;Ck for FXDE in use
      LD    A,E
      CP    0FEH        ;X'FE' => FXDE in use
      JR    Z,REMOV2    ;Jump if FXDE in use
      CALL  @GATWR            ;  else write the GAT
      RET   NZ          ;Ret if write error
      PUSH  IX          ;Transfer FCB address
      POP   HL          ;  to HL & zero out FCB
      LD    B,32        ;Init for 32-byte field
      XOR   A           ;Zero accum
ZERLP1      LD    (HL),A            ;Zero out the entire FCB
      INC   HL
      DJNZ  ZERLP1
      RET
;
;     REMOVE will only close a logical device
;
CLOSDCB     CP    10H         ;Is this an open DCB
      LD    A,38        ;Init "file not open
      RET   NZ
      CALL  LNKFCB@           ;Link to DCB (DE->IX)
      LD    C,(IX+6)    ;Get device name
      LD    B,(IX+7)
      LD    (IX+0),'*'  ;Stuff device indicator
      LD    (IX+1),C    ;Stuff 1st char of name
      LD    (IX+2),B    ;Stuff 2nd char of name
      LD    (IX+3),03H  ;Terminate with ETX
      XOR   A
      RET
;
;     Deallocate an extent
;
RMVEXT      PUSH  HL
      PUSH  BC
      LD    A,8         ;P/u the # of grans per
      CALL  @DCTBYT           ;  cylinder into reg A
      RLCA              ;Shift into bits 0-2
      RLCA
      RLCA
      AND   7           ;Remove all else
      INC   A           ;Adjust for zero offset
;
;     Ck for 2-sided operation
;
      LD    L,A         ;Save current grans/cyl
      LD    A,4
      CALL  @DCTBYT            ;Get 2-sided flag
      BIT   5,A         ;Test 2-sided
      LD    A,L         ;Xfer value back
      JR    Z,$+3       ;Bypass if 1-sided
      ADD   A,A         ;  else multiply by 2
      LD    (GRNSCYL+1),A     ;Modify later instruction
      LD    L,E         ;Relative cylinder -> L
      LD    H,DIRBUF$>8 ;Point to GAT byte
      LD    A,D         ;Rel gran & # of grans
      AND   1FH         ;Get # of grans
      LD    C,A         ;  into reg C & adjust
      INC   C           ;  for zero offset
      XOR   D           ;Get rel gran & shift
      RLCA              ;  into bits 0-2
      RLCA
      RLCA
RMVEX1      PUSH  AF          ;Save rel starting gran
      LD    B,(HL)            ;P/u allocation byte
      CALL  RMVGRN            ;Turn off bit for a gran
      LD    (HL),B            ;Update GAT byte
      POP   AF          ;Recover starting gran
      INC   A           ;Bump up
GRNSCYL     CP    0           ;Ck with grans per cyl
      JR    NZ,DECGRNS  ;Go if still on this cyl
      XOR   A           ;  else zero gran counter
      INC   L           ;Bump to next cyl in GAT
DECGRNS     DEC   C           ;Decrement # of grans
      JR    NZ,RMVEX1   ;Go if more to deallocate
      POP   BC          ;  else recover regs
      POP   HL          ;  and go home
      RET
;
;     Remove a bit to deallocate & free up a gran
;
RMVGRN      AND   7           ;Max 8-grans per cyl
      RLCA              ;Shift to create
      RLCA              ; RES opcode
      RLCA
      OR    80H         ;Merge rest of RES code
      LD    (RMVGRN1+1),A     ;Stuff into the instr
RMVGRN1     RES   0,B         ;Reset the proper bit
      RET
;
LAST  EQU   $
      IF    $.GT.DIRBUF$
      ADISP 'ERROR: Module too big'
      ENDIF
      ORG   MAXCOR$-2
      DEFW  LAST-SYS10  ;Overlay size
;
      END   SYS10
```

SYS11/ASM

```
;SYS11/ASM - LS-DOS 6.2
;     ?
      ADISP '<SYS11 - LS-DOS 6.2>'
;
LF    EQU   10
CR    EQU   13
*LIST OFF               ;Get SYS0/EQU
*REF  'SYS0/EQU:1'
*LIST ON
*GET  'COPYCOM:1'       ;Copyright message
;
      ORG   1E00H
;
SYS11 AND   70H
      RET   Z           ;Back on zero entry
      PUSH  HL
      LD    HL,KFLAG$   ;Reset the <ENTER>
      RES   2,(HL)            ;  bit every time
      POP   HL
      CP    20H         ;New @EXIT?
      JR    Z,NEWEXIT
      CP    40H         ;New keyboard request
      JP    Z,KEYREQ    ;  after input of a line?
      CP    50H         ;//INPUT followup
      JP    Z,GETKEY
      CP    10H         ;Initial entry to DO?
      RET   NZ
;
;     <DO> initialization of Sysres hooks
;
      DI                ;Clock off for now
      LD    HL,KFLAG$   ;Reset break bit only on
      RES   0,(HL)            ;  initial entry
      LD    HL,SFLAG$
      BIT   5,(HL)            ;If DO already in effect
      SET   5,(HL)            ;  don't rehook
      JR    NZ,IPLDO1
      LD    A,0ADH            ;Change @EXIT, @ABORT to use
      LD    (@EXIT+1),A ;  SYS11 rather than SYS1
IPLDO1      LD    SP,STACK$
      EI                ;Clock back on
      LD    DE,JFCB$    ;At end of SYSTEM/JCL?
      CALL  @CKEOF
      JP    NZ,@ERROR
      LD    DE,IPLDO2   ;Init JCLCB$
      LD    (JCLCB$+1),DE
      CALL  GETLINE           ;Get a line from the file
      LD    DE,@DOKEY   ;Change vector to SYS11,
      LD    (JCLCB$+1),DE     ;  entry 4
      JR    $?1         ;Go interpret it
IPLDO2      LD    DE,JFCB$    ;JCLCB$ input routine
      JP    @GET
;
;     New @EXIT processing
```

```
;
NEWEXIT     LD    SP,STACK$   ;Reset the stack
      EI
      LD    A,H         ;Ck for error return
      OR    L
      JR    NZ,ABORT
      LD    HL,SFLAG$
      BIT   4,(HL)            ;BREAK key disabled?
      JR    NZ,NEWEX1
      CALL  @CKBRKC           ;Check on <BREAK>
      JR    NZ,ABORT
NEWEX1      LD    DE,JFCB$    ;Exit if end of JCL
      CALL  @CKEOF
      JR    NZ,EXIT
      CALL  GETLINE           ;Grab a JCL line
$?1   JP    @CMNDI
GETLINE     LD    HL,INBUF$   ;Pt to line buffer
      LD    BC,79<8           ;Max 79 chars
      JP    @KEYIN
;
;     New ABORT processor
;
ABORT LD    HL,ABORT$   ;"Job aborted
      LD    DE,@ABORT
      JR    EXIT1
;
;     Scan for ENTER or BREAK
;
KSCN  LD    A,(SFLAG$)  ;Only test BREAK if
      BIT   4,A         ;  BREAK key enabled
      LD    A,(KFLAG$)
      JR    NZ,KSCN1
      BIT   0,A         ;BREAK detected?
      JR    NZ,ABORT
KSCN1 BIT   2,A         ;Test <ENTER>
      RET   Z           ;Back if not
KSCN2 CALL  @KBD        ;Clear the type ahead
      JR    Z,KSCN2
      LD    HL,KFLAG$   ;Reset the ENTER bit
      RES   2,(HL)
      PUSH  BC
      LD    B,3000>8
      CALL  @PAUSE
      POP   BC
      LD    A,(HL)            ;Don't return until clear
      AND   4
      XOR   4
      JR    Z,KSCN2
      RET
;
;     Continuation of EXIT processing
;
EXIT  LD    HL,JOBDUN$  ;"Job done.
      LD    DE,@EXIT
EXIT1 PUSH  DE
      CALL  @LOGOT            ;Log & fall through
;
```

```
;      Turn off the DO processor
;
DOOFF EQU   $
      DI
      LD    HL,SFLAG$   ;Reset <DO> flag
      RES   5,(HL)
      XOR   A
      LD    (JFCB$),A   ;Show FCB is closed
      LD    H,A         ;Set = 0 for @EXIT
      LD    L,A
      LD    DE,KIDCB$   ;Clear any type-ahead
      LD    A,3
      CALL  @CTL        ;  buffer (no streaming)
      LD    A,93H       ;Restore @EXIT SVC
      LD    (@EXIT+1),A ;  back to SYS1
      RET
;
;      Keyboard request processor
;
KEYREQ      LD    HL,10       ;Back stack up 5 words
      ADD   HL,SP       ;SYS0,RET,DE,HL,IX,BC
      LD    C,(HL)            ;Get contents of BC
      INC   HL          ;  prior to keyboard
      LD    B,(HL)            ;  request & DRIVER save
;
;      @KEYIN is requesting an entire line
;
KEYLINE     LD    DE,JFCB$    ;Ck on end of JCL file
      PUSH  BC
      CALL  @CKEOF
      POP   BC
      JR    NZ,EXIT
      LD    A,B         ;Do we need to re-read
      CP    C           ;  the JCL sector?
      JP    NZ,@GET
      CALL  @RREAD            ;Get the sector back
      JP    NZ,@ERROR
      CALL  @GET        ;Get a byte from the
      OR    A           ;  JCL file
      JR    Z,EXIT            ;Exit on Zero byte
      CP    '/'         ;Is this line execution
      JR    Z,GOTSLSH   ;  JCL code to parse?
      CP    A           ;Set Z-flg
      RET
;
;      Found an execution code line
;
GOTSLSH     PUSH  BC          ;Save reg pr BC
      PUSH  DE          ;Save DCB addr
      LD    B,79        ;Only 79-char max line
      LD    HL,INBUF$   ;Get rest of line
      PUSH  HL          ;  into JCL buffer
GOTSL1      LD    (HL),A            ;compare for CR as end
      INC   HL          ;  of line
      CP    CR
      JR    Z,GOTSL2
      CALL  @GET        ;Get a character
      DJNZ  GOTSL1            ;  up to 79 max
      JR    BADJCL            ;Line too long
GOTSL2      POP   HL          ;Rcvr pointer to bufr
      PUSH  HL          ;  and save again
      INC   HL          ;Pt to 2nd char
      LD    A,(HL)
      CP    '/'         ;Found a //?
      JR    NZ,REKEY2
      INC   HL          ;Ck on ///
      SUB   (HL)
      JP    Z,KEYIN6    ;Jump if ///
      SUB   0F6H
      JP    NC,KEYIN5   ;Jump if 3rd char is 0-9
      EX    (SP),HL           ;P/u start of command
      CALL  @LOGER            ;  line & log it
      EX    (SP),HL
GOTSL3      LD    A,(HL)            ;Was char ENTER?
      CP    CR
      JR    Z,REKEY2
      CP    ' '         ;Ignore leading spaces
      INC   HL
      JR    Z,GOTSL3
      DEC   HL
      LD    DE,LILBUF   ;Put possible parm -> buf
      LD    B,5         ;Max length of parm
      CALL  PARSER            ;Parse parm
      JR    NZ,REKEY2
      LD    DE,LILBUF
      LD    BC,PARMTBL  ;Is the parm a macro?
      CALL  FNDPARM
      JR    NZ,REKEY2   ;Bypass if not in tbl
      PUSH  DE          ;Stack routine's entry
      RET               ;  & go to it
REKEY1      POP   BC
REKEY2      POP   HL
      POP   DE
      POP   BC
      JR    KEYLINE
BADJCL      LD    HL,BADJCL$  ;"invalid JCL...
      JP    ABORT+3
;
;     Process //STOP
;
STOP  CALL  DOOFF       ;Turn off DO proc
      POP   HL
      POP   DE
      POP   BC
      EI
      JP    @KEY        ;Go to keyboard
;
;     Process //DELAY
;
DELAY EX    (SP),HL           ;Pt to //delay line
      CALL  @DSPLY            ;  and display it
      EX    (SP),HL
      CALL  @DECHEX           ;Cvrt entry to binary
      LD    B,C         ;Set count
```

```
DELAY1      CALL  SILEN1            ;Delay a bit
      DJNZ  DELAY1
      JR    REKEY2
;
;     Process //PAUSE
;
PAUSE POP   HL          ;Display "pause..
      PUSH  HL
      CALL  @DSPLY
PAUSE1      CALL  KSCN        ;Loop for BREAK or ENTER
      JR    Z,PAUSE1
      JR    REKEY2
;
;     Process //KEYIN
;
KEYIN POP   HL          ;Rcvr pointer to "KEYIN
      PUSH  HL
KEYIN1      LD    A,(HL)            ;Display JCL command line
      INC   HL
      CP    CR
      JR    Z,KEYIN2
      CALL  @DSP
      JR    KEYIN1
KEYIN2      CALL  @KEY        ;Get & display the char
      CALL  @DSP
      LD    (KEYIN5+1),A      ;Stuff for compare
      LD    A,CR
      CALL  @DSP        ;Write new line
KEYIN3      POP   HL
      PUSH  HL
      LD    DE,JFCB$    ;Ck for end of JCL
      CALL  @CKEOF
      JP    NZ,EXIT
KEYIN4      CALL  @GET        ;Xfer a line of JCL
      LD    (HL),A            ;  to buffer
      INC   HL
      CP    CR
      JR    NZ,KEYIN4
      POP   HL
      PUSH  HL
      LD    A,(HL)            ;Look for // to find
      CP    '/'         ;Start of procedure block
      JR    NZ,KEYIN3
      INC   HL
      CP    (HL)        ;//?
      JR    NZ,KEYIN3
      INC   HL          ;Point to proc label
      SUB   (HL)        ;Is label a '/' noting
      JR    Z,KEYIN6    ;  exec phase cond's end?
      LD    A,(HL)            ;Nope, get proc label
KEYIN5      CP    0           ;Same as key entry?
      JR    NZ,KEYIN3   ;No match? check next one
KEYIN6      LD    (KEYIN5+1),A      ;Stuff 0 if ///
      POP   HL
      PUSH  HL
      CALL  @LOGER            ;Log the command
      JR    REKEY2
```

```
;
;     Process //ALERT
;
ALERT XOR   A
      LD    (ALERT4+1),A      ;Start with clean flag
ALERT1      LD    A,(HL)            ;Ignore spaces
      INC   HL
      CP    ' '
      JR    Z,ALERT1
      CP    ','         ;Comma separator?
      JR    Z,ALERT1
      CP    CR          ;End of line?
      JP    Z,REKEY2
      CP    ')'         ;Closing paren?
      JR    Z,ALERT2
      CP    '('         ;Start of parms?
      JR    NZ,ALERT3   ;If none of the above...
      LD    (ALERT2+1),HL     ;Save ptr to parm start
      JR    ALERT1
;
;     Check here when closing parm received
;
ALERT2      LD    HL,0        ;P/u ptr to '(' if there
      LD    A,H         ;If the //ALERT1 started
      OR    L           ;  with a '(', then
      JR    NZ,ALERT1   ;  repeat the parm
      JP    BADJCL            ;  parsing, else exit
;
;     Assumed integer parm found
;
ALERT3      DEC   HL          ;Backup pointer
      CALL  @DECHEX           ;Cvrt value to binary
      LD    B,C         ;Keep value as counter
ALERT4      LD    A,0         ;Flip flag: entries 1, 3,
      XOR   0FFH        ;  5, ... are noise, 2,
      LD    (ALERT4+1),A      ;  4,6, ... are silence
      LD    C,A
      BIT   0,C         ;Test noise or silence
      CALL  NZ,@SOUND   ;Call for sound out
      BIT   0,C         ;  then test again
      CALL  Z,SILENCE   ;Silence ...
      CALL  KSCN        ;Ck BREAK or ENTER
      JP    NZ,REKEY2   ;Go on enter
      JR    ALERT1            ;Loop if not
;
;     Silence routine
;
SILENCE     OR    B           ;A was zero
      RET   Z
      CALL  SILEN1            ;Delay a bit
      DJNZ  SILENCE           ;  for duration
      RET
SILEN1      PUSH  BC          ;Delay for 0.1 sec
      LD    BC,6555
      CALL  @PAUSE
      POP   BC
      RET
```

```
;
;     Process //FLASH
;
FLASH CALL  @DECHEX
      LD    B,C         ;P/u the flash count
      POP   HL
      PUSH  HL
FLASH1      PUSH  BC
      CALL  @DSPLY            ;Display the prompt
      LD    BC,4000H    ;Countdown to flash msg
FLASH2      CALL  KSCN        ;Keep testing <ENTER>
      JP    NZ,REKEY1   ;  key during countdown
      DEC   BC          ;BREAK would abort
      LD    A,B
      OR    C
      JR    NZ,FLASH2   ;Loop until count=0
      LD    A,27        ;Erase the message line
      CALL  @DSP        ;Cursor up to prev line
      LD    A,30
      CALL  @DSP        ;Erase to end of line
      CALL  SILEN1            ;Delay while blanked
      POP   BC
      DJNZ  FLASH1
FLASH3      JP    REKEY2
;
;     Process //SLEEP and //WAIT
;
SLEEP DB    3EH         ;Make it LD A,0AFH
WAIT  XOR   A
      LD    (SLPWT+1),A ;Save entry state
      EX    (SP),HL           ;Display the JCL line
      CALL  @DSPLY
      EX    (SP),HL
      LD    DE,TIMFLD   ;Pt to time field
      LD    B,3         ;Set up loop counter
      JR    PAKTIM1
PAKTIM      CP    ':'-'0'           ;Test valid separator
      JP    NZ,BADJCL
PAKTIM1     PUSH  BC
      CALL  @DECHEX           ;Cvrt the hours
      LD    (HL),C            ;Store time parm
      LDI               ;Shift & bump HL & DE
      POP   BC          ;Rcvr the loop counter
      DJNZ  PAKTIM            ;Loop for 3 values
SLPWT LD    A,0         ;P/u sleep/wait flag
      OR    A
      JR    Z,TSTIME    ;Go if //WAIT
      LD    HL,TIMFLD+2 ;Point to seconds
      LD    DE,TIME$
      LD    B,2
SLP1  LD    A,(DE)            ;Add secs/mins
      ADD   A,(HL)
      LD    (HL),A            ;Store
      SUB   60          ;Ck overflow to mins/hrs
      JR    C,SLP2            ;Go if none
      LD    (HL),A            ;Update value mod 60
      DEC   HL          ;  & bump next field
      INC   (HL)
      INC   HL          ;Adj for dec
SLP2  INC   DE          ;Bump time$
      DEC   HL          ;Bump user field
      DJNZ  SLP1
      LD    A,(DE)            ;Add hours
      ADD   A,(HL)
      LD    (HL),A
      SUB   24          ;Wrap past midnight?
      JR    C,TSTIME    ;Go if not else
      LD    (HL),A            ;  adjust mod 24
;
;     Wait until the system clock advances to request
;
TSTIME      CALL  KSCN        ;Scan for BREAK
      LD    HL,TIMFLD
      LD    DE,TIME$+2
      LD    B,3         ;Set loop counter
CKTIME      LD    A,(DE)            ;P/u a time value
      CP    (HL)        ;Match user input?
      JR    NZ,TSTIME   ;Go if no match
      INC   HL          ;Inc the user req ptr
      DEC   DE          ;Dec the time string ptr
      DJNZ  CKTIME            ;Loop for 3 values
      JR    FLASH3            ;All match, exit!
;
;     Process //INPUT request
;
INPUT POP   HL          ;Recover JCL line &
      CALL  @DSPLY            ;  display it
      LD    A,0DDH            ;Change sysres hook
      LD    (@DOKEY+1),A
      POP   DE          ;Maintain Stck integrity
      POP   BC          ;Get @KEYIN values
;
;     This next routine will satisfy the request
;
GETKEY      CALL  @KEY        ;Fetch from keyboard
      PUSH  AF          ;Don't disturb flag
      DEC   A
      JR    Z,UNHOOK    ;Change back on BREAK
      CP    CR-1        ;  or ENTER
      JR    Z,UNHOOK
      POP   AF          ;Recover flag
      RET
UNHOOK      LD    A,0CDH            ;Restore Sysres hook
      LD    (@DOKEY+1),A
      POP   AF          ;Get saved character
      RET
;
;     Parameter list & scanners
;
;     Parse a field
;     (HL) => command line
;     (DE) => FCB area
;     Z    <= found valid field
;     NZ   <= found invalid field
```

```
;
PARSER      LD    B,8         ;Set length
PAR1  LD    A,B
      LD    (PAR6+1),A
      INC   B
PAR2  LD    A,(HL)
      CP    03H         ;ETX?
      JR    Z,PAR5
      CP    CR          ;<ENTER>?
      JR    Z,PAR5
      CP    '('         ;Begin of parm?
      JR    Z,PAR5
      INC   HL          ;Bump pointer to next
      CALL  TST09AZ           ;Test if 0-9,A-Z
      JR    NC,PAR3           ;Go if one of the above
      CP    'a'         ;Check on lower case
      JR    C,PAR5            ;Jump on non-alpha
      CP    'z'+1       ;Is it a-z?
      JR    NC,PAR5           ;Jump on non-alpha
      RES   5,A         ;Convert lower to upper
PAR3  DEC   B           ;Count down
      JR    Z,PAR4
      LD    (DE),A            ;Xfer the char
      XOR   A           ;Show at least 1 valid
      LD    (PAR6+1),A  ;  char was detected
      INC   DE          ;Bump FCB pointer
      JR    PAR2
;
PAR4  INC   B           ;Here on max chars ck'd
      JR    PAR2
PAR5  LD    C,A         ;Save separator
      LD    A,03H       ;Stuff an ETX
      LD    (DE),A
PAR6  LD    A,0         ;Set Z-flag if at least
      OR    A           ;  1 valid char detected
      LD    A,C         ;Recover separator char
      RET
TST09AZ     CP    '0'         ;Special character?
      RET   C           ;Go if not in range
      CP    '9'+1       ;Jump on bad digit
      JR    C,EXITC           ;Go if 0-8 & make NC
      CP    'A'         ;Jump on spec char
      RET   C           ;Go with C-flag if 3B-40
      CP    'Z'+1       ;Jump on A-Z
EXITC CCF               ;Switch flag of result
      RET
;
;     Find parameter in table
;     (HL) => pointer to line
;     (DE) => pointer to buffer area
;     (BC) => pointer to parameter table
;       C  <= entry # of parm in table
;     (DE) <= parm vector address
;       Z  <= set if found
;      NZ  <= if not found in table
;     Routine similar as FIND.PARM in SYS1 - dif width
;
```

```
FNDPARM     PUSH  HL
      LD    H,B         ;Xfer the table address
      LD    L,C
FND1  LD    A,(DE)            ;P/u input byte
      CP    (HL)        ;Match 1st char of table?
      JR    Z,FND3            ;Jump if 1st matches
FND2  PUSH  BC          ;  else bypass that entry
      LD    BC,7        ;Width of table
      ADD   HL,BC
      POP   BC
      LD    A,(HL)            ;Test for table end
      OR    A
      JR    NZ,FND1           ;Loop if not at end
      POP   HL
      INC   A           ;  else set NZ return
      RET
;
;     1st matches, do the rest?
;
FND3  LD    B,4         ;# chars remaining
      PUSH  HL
      PUSH  DE
FND4  INC   DE
      INC   HL
      LD    A,(DE)            ;P/u input char
      CP    03H         ;ETX?
      JR    Z,FND7
      CP    CR          ;End of line?
      JR    Z,FND7
      CP    (HL)        ;Match with table?
      JR    NZ,FND6           ;Exit & test the char
      DJNZ  FND4        ;Loop for limit
FND5  POP   DE          ;Must be a match
      POP   BC
      LD    HL,5        ;Point to vector
      ADD   HL,BC
      LD    E,(HL)            ;Xfer vector to DE
      INC   HL
      LD    D,(HL)
      POP   HL
      XOR   A           ;  & show it found
      RET
;
;     No match if alphanumeric unless a space
;
FND6  CALL  TST09AZ           ;Ck for 0-9, A-Z
      JR    NC,FND8           ;Go if one of the above
FND7  LD    A,(HL)            ;Loop if table has
      CP    ' '         ;  trailing spaces
      JR    Z,FND5
FND8  POP   DE
      POP   HL
      JR    FND2
;
LILBUF      DS    6
TIMFLD      EQU   LILBUF
BADJCL$     DB    'Bad JCL, '
```

```
ABORT$      DB    'Job aborted',CR
JOBDUN$     DB    'Job done',CR
PARMTBL     DB    'ABORT'
      DW    ABORT
      DB    'ALERT'
      DW    ALERT
      DB    'DELAY'
      DW    DELAY
      DB    'EXIT '
      DW    EXIT
      DB    'FLASH'
      DW    FLASH
      DB    'KEYIN'
      DW    KEYIN
      DB    'PAUSE'
      DW    PAUSE
      DB    'SLEEP'
      DW    SLEEP
      DB    'STOP '
      DW    STOP
      DB    'WAIT '
      DW    WAIT
      DB    'INPUT'
      DW    INPUT
      DB    0           ;End of table
LAST  EQU   $
      IF    $.GT.DIRBUF$
      ADISP 'ERROR: Module too big'
      ENDIF
      ORG   MAXCOR$-2
      DW    LAST-SYS11  ;Overlay size
;
      END   SYS11
```

SYS12/ASM

```
;SYS12/ASM - LS-DOS 6.2
      ADISP '<SYS12 - LS-DOS 6.2>'
;
CR    EQU   13
*LIST OFF               ;Get SYS0/EQU
*REF  'SYS0/EQU:1'
*LIST ON
*GET  'COPYCOM:1'       ;Copyright message
;
      ORG   1E00H
;
SYS12 AND   70H         ;Strip bit 7
      RET   Z           ;Back on zero entry
      CP    30H         ;Locate module address?
      JP    Z,GTMOD
      CP    20H         ;Mini dir?
      JP    Z,MDIR
      CP    10H         ;RAMDIR?
      RET   NZ          ;Ret if any other entry
;
;     RAMDIR interfacing
;     HL = user buffer area
;      B = drive #
;      C = 0 for entire directory
;      C = 1-254 for selected DEC-1 (02-FF)
;      C = 255 for disk space; in use/free
;
RAMDIR      LD    A,7         ;Ck on valid drive #
      CP    B
      LD    A,32        ;Init "Illegal drive
      RET   C
      CALL  LNKFCB@           ;Save regs
      LD    A,B         ;Get drive where needed
      LD    B,C         ;Tnsfer DEC to B
      LD    C,A         ;  & drive to C
      OR    '0'         ;Make it ASCII
      LD    (DSTDRV+1),A      ;Stuff for STUFBUF
      CALL  CKDRV       ;Be sure disk is there
      RET   NZ
      INC   B           ;Test 0, 1-254, 255
      JR    NZ,DIRINFO  ;Go if directory req
;
;     Get FREE SPACE info
;
      PUSH  HL          ;Save buffer pointer
      CALL  SPACE       ;Get our info
      LD    B,(HL)            ;P/u free space in K
      DEC   HL          ;  into BC
      LD    C,(HL)
      DEC   HL
      LD    A,(HL)            ;Get total space in K
      DEC   HL          ;  into HL
      LD    L,(HL)
      LD    H,A
      SBC   HL,DE       ;Calc "in use" (C flg is 0)
      EX    DE,HL       ;Tnsfer to DE
      POP   HL          ;Rcvr user bufr ptr
      LD    (HL),E            ;Stuff "in use"
      INC   HL
      LD    (HL),D
      INC   HL
      LD    (HL),C            ;Stuff "free to use"
      INC   HL
      LD    (HL),B
      XOR   A           ;Show no error
      RET
;
;     Do RAMDIR directory info
;
DIRINFO     DEC   B           ;If DEC=0, do it all
      JR    Z,DOALL           ;Go if all of it
      INC   B           ;1=>2, 2=>3, ..., FE=>FF
;
;     Calculate the number of directory sectors
;     = (#sectors x #heads) - 2 for GAT & HIT
;
      LD    A,7         ;Get highest # sector
      CALL  @DCTBYT
      LD    D,A         ;Store heads & sectors
      AND   1FH         ;Mask off # sectors
      LD    E,A         ;  & stuff into E
      INC   E           ;Bump for 0 offset
      XOR   D           ;Recover # heads
      RLCA              ;  into bits 0-2
      RLCA
      RLCA
      INC   A           ;Bump for 0 offset
      CALL  @MUL8       ;Multiply sectors x heads
      LD    E,A         ;Now check double bit
      LD    A,4
      CALL  @DCTBYT
      BIT   5,A         ;Set if 2-sided
      LD    A,E
      JR    Z,ONESID    ;Go if not set else
      ADD   A,A         ;  double value
ONESID      SUB   2           ;Reduce for GAT & HIT
      LD    D,A         ;D => # dir sectors
      LD    A,B         ;Get requested DEC
      AND   1FH
      CP    D           ;See if in range
      JR    C,DIRINF1   ;Go if so
      LD    A,16        ;"Illegal logical file #
      OR    A           ;Return out of range error
      RET
;
DIRINF1     PUSH  HL          ;Save buffer ptr
      CALL  @DIRRD            ;Get its directory record
      POP   DE          ;Rcvr buf ptr
      RET   NZ          ;Back on an error
      LD    A,(HL)            ;Get attributes
      AND   0D8H        ;Only if in use & VIS
      XOR   10H         ;Flip state so NZ=no
      LD    A,25        ;Init file access denied
      RET   NZ          ;Back on no file, SYS, INV
GETSTUF     PUSH  HL          ;Save DIR ptr
      CALL  STUFBUF           ;Stuff the filespec
      POP   HL
      LD    A,(HL)
      AND   7           ;Keep the access level
      LD    (DE),A
      INC   DE
      INC   L           ;Go up to EOF offset
      INC   L
      INC   L
      LDI               ;Move in the offset & LRL
      LDI
      LD    A,L         ;Bump to ERN
      ADD   A,15
      LD    L,A
      LD    A,(HL)            ;P/u ERN
      LD    (DE),A            ;  and tnsfer it
      INC   L
      INC   DE
      LD    H,(HL)
      LD    L,A         ;# sectors to HL
      EX    DE,HL       ;  hence to DE
      LD    (HL),D            ;Stuff ERN Hi-order
      INC   HL          ;Bump bufr ptr
      INC   DE          ;Adjust for rounding
      INC   DE
      INC   DE
      SRL   D           ;Divide by 4 to calc K
      RR    E
      SRL   D
      RR    E
      LD    (HL),E            ;Xfer result into bufr
      INC   HL
      LD    (HL),D
      INC   HL
      LD    (HL),'+'    ;Stuff buffer terminator
      EX    DE,HL       ;Buffer ptr to DE again
      XOR   A           ;Set Z=no error
      RET
;
;     RAMDIR - Do all of the directory
;
DOALL EX    DE,HL       ;Buffer pointer to DE
      CALL  HITRD1            ;Read in the HIT
      RET   NZ          ;Exit if read error
      JR    DOALL3
;
DOALL1      POP   BC          ;Recover HIT pointer lo
      LD    H,DIRBUF$>8
      LD    L,B         ;Advance to next dir
DOALL2      LD    A,L         ;  record ot this sector
      ADD   A,32
      LD    L,A
      JR    NC,DOALL3   ;Bypass if still same
      INC   L           ;  else point to next one
      BIT   5,L         ;Finished with
      JR    Z,DOALL3    ;  this drive?
      XOR   A
      RET
;
DOALL3      LD    A,(HL)            ;P/u HIT entry
      OR    A
      JR    Z,DOALL2    ;Jump if spare
      LD    B,L         ;Save DEC in reg B
      PUSH  BC          ;  & to stack
      LD    A,L         ;Pt to dir record for
      AND   0E0H        ;  this DEC
      LD    L,A         ;Get the dir sector for
      XOR   B           ;  this DEC
DOALL4      CP    0FFH        ;Same as on in core?
      JR    Z,DOALL5    ;Jump if so else
      LD    (DOALL4+1),A      ;  update one we have and
      CALL  @DIRRD            ;  read it into buffer
      JP    NZ,MDIR12   ;Jump on read error
DOALL5      LD    H,SBUFF$>8  ;Sysbuf hi-order
      LD    A,(HL)            ;P/u attributes
      AND   0D8H        ;Test FXDE & in-use
      XOR   10H         ;If not used or FXDE
      JR    NZ,DOALL1   ;  then back to DOALL1
      PUSH  HL
      CALL  GETSTUF           ;Get the dir info
      POP   HL
      JR    DOALL1
;
;     Routine to display a mini directory
;      C => drive number in binary
;      B => option, 0 = display, 1 = buffer stuff
;        2 = display /EXT, 3 = buffer /EXT
;        4 = space into buffer
;     HL => address of buffer to dtuff dir info & EXT
;      Z <= set on valid conclusion
;     NZ <= set on any error
;
MDIR  LD    A,7         ;Test for bad drive #
      CP    C
      LD    A,32        ;Init "illegal drive...
      RET   C
      CALL  CKDRV       ;Be sure disk is there
      RET   NZ
      CALL  LNKFCB@           ;Save the regs
      LD    A,B         ;Stuff the option
      LD    (TSTOPT+1),A
      CP    4           ;If option 4, go get
      JP    Z,SPACE0    ;  space info
      LD    A,43        ;Init "SVC parm error
      RET   NC          ;Back if option > 4
      PUSH  HL          ;Save possible buffer
      PUSH  BC
      LD    DE,LILBUF   ;Save possible EXT
      LD    BC,3
      LDIR
      POP   BC
      LD    A,C         ;Get drive # and
      OR    '0'         ;  make it ASCII
      LD    (DSTDRV+1),A
      LD    A,5         ;Init to 5 files/line
      LD    (MDIR11+1),A
      LD    A,23        ;  & 23 lines/page
      LD    (CKPAGE+1),A
      CALL  HITRD1            ;Read in the HIT
      POP   DE          ;Rcvr possible buffer
      RET   NZ          ;Exit if read error
      JR    MDIR3
MDIR1 POP   BC          ;Recover HIT pointer Lo
      LD    H,DIRBUF$>8
      LD    L,B         ;Advance to next dir
MDIR2 LD    A,L         ;  record of this sector
      ADD   A,32
      LD    L,A
      JR    NC,MDIR3    ;Bypass if still same
      INC   L           ;  else point to next one
      BIT   5,L         ;Finished with
      JR    Z,MDIR3           ;  this drive?
      LD    A,(TSTOPT+1)      ;If option1 or 3,
      AND   1           ;  must stuff buffer end
      JR    NZ,CLSBUF
      LD    A,CR        ;  else do a blank line
      CALL  @DSP
      XOR   A
      RET
;
CLSBUF      LD    A,0FFH            ;Put in buffer terminator
      LD    (DE),A
      XOR   A
      RET
;
MDIR3 LD    A,(HL)            ;P/u HIT entry
      OR    A
      JR    NZ,MDIR2    ;Jump if spare
      LD    B,L         ;Save DEC in reg B
      PUSH  BC          ;  & to stack
      LD    A,L         ;Pt to dir record for
      AND   0E0H        ;  this DEC
      LD    L,A         ;Get the dir sector for
      XOR   B           ;  this DEC
MDIR4 CP    0FFH        ;Same as one in core?
      JR    Z,MDIR5           ;Jump if so
      LD    (MDIR4+1),A ;Else update one we have
      CALL  @DIRRD            ;  and read it into buf
      JR    NZ,MDIR12   ;Jump on read error
MDIR5 LD    H,SBUFF$>8  ;Sysbuf hi-order
      LD    BC,MDIR1    ;Set up the return addr
      PUSH  BC
TSTOPT      LD    A,0         ;P/u option #
      PUSH  HL
      PUSH  DE
      CALL  TSTSAM            ;Check for extension match
      POP   DE
      POP   HL
      RET   NZ          ;Back to MDIR1
      LD    A,(TSTOPT+1)
      RRCA              ;Test option 1 or 3
      LD    A,(HL)
      JR    NC,DSPLYIT  ;Go if 0 or 2
      AND   90H         ;Test FXDE & in-use bits
      XOR   10H         ;If not used, FXDE
      RET   NZ          ;Back to MDIR1
      LD    BC,16
      LDIR              ;User's buffer
      INC   L           ;Bypass stored passwords
      INC   L
      INC   L
      INC   L
      LD    C,2         ;  and tnsfer ERN
      LDIR
      RET               ;Back to MDIR1
;
DSPLYIT     AND   0D8H        ;Test if we want this
      XOR   10H         ;Only if in-use & VIS
      RET   NZ          ;Back to MDIR1
      LD    DE,LILBUF+3
      PUSH  DE
      CALL  STUFBUF           ;Move filespec to buffer
      POP   HL          ;Rcvr LILBUF ptr
      CALL  @DSPLY            ;Display the file
MDIR11      LD    A,0         ;Count down 5-across
      DEC   A
      LD    (MDIR11+1),A      ;Update count
      RET   NZ          ;Loop if more to go
      LD    A,5         ;  else re-init
      LD    (MDIR11+1),A
      LD    A,CR
      CALL  @DSP        ;New line
CKPAGE      LD    A,0         ;P/u display count
      DEC   A
      LD    (CKPAGE+1),A
      RET   NZ
      LD    A,23
      LD    (CKPAGE+1),A      ;Reset for max
      CALL  @KEY        ;Wait for keyboard input
      JP    @CLS        ;Clear screen and ret
;
MDIR12      POP   BC
      RET
;
TSTSAM      BIT   1,A         ;Ck if /EXT option
      RET   Z           ;Ret with Z if
      LD    BC,13       ;  option <> /EXT
      ADD   HL,BC       ;Else point to /EXT
      LD    B,3         ;  field of dir record
      LD    DE,LILBUF   ;  & check for match
TSTS1 LD    A,(DE)
      CP    '$'         ;'$' matches with all
      JR    Z,TSTS2
      CP    'A'         ;If numeric, don't conv
      JR    C,$+4        ;  to upper case
      RES   5,A          ;Cvrt to UC if lc
      CP    (HL)
      RET   NZ           ;Ret on no match
TSTS2 INC   HL
      INC   DE
      DJNZ  TSTS1        ;Loop for 3 chars
      RET
;
;     Routine to construct the filespec field
;
STUFBUF     LD    A,L
      ADD   A,5          ;Pt to start of filename
      LD    L,A
      LD    C,13         ;Init for 15 (-2) chars
      LD    B,8          ;Filename
STUFB1      LD    A,(HL)
      INC   HL
      CP    ' '          ;Exit on 1st space
      JR    Z,STUFB2
      LD    (DE),A             ;Stuff the char
      INC   DE
      DEC   C            ;String count down
      DJNZ  STUFB1             ;Field loop
      JR    STUFB3             ;Bypass ext calculation
STUFB2      LD    A,L          ;Calculate start of
      ADD   A,B          ;EXT field in dir record
      DEC   A
      LD    L,A
STUFB3      LD    A,(HL)             ;Display EXT if present
      CP    ' '
      JR    Z,STUFB5     ;Exit if no extension
      LD    A,'/'        ;Display slash
      LD    (DE),A             ;Stuff the char
      INC   DE
      DEC   C            ;Dsply char countdown
      LD    B,3          ;3 chars max for EXT
STUFB4      LD    A,(HL)
      INC   HL
      CP    ' '
      JR    Z,STUFB5     ;Exit on 1st blank
      LD    (DE),A             ;Else stuff the char
      INC   DE
      DEC   C
      DJNZ  STUFB4             ;Loop 3 chars
STUFB5      LD    A,':'        ;Stuff drive separator
      LD    (DE),A             ;Reg C already counted
      INC   DE           ;  for in the init
DSTDRV      LD    A,0          ;P/u the drive #
      LD    (DE),A
      INC   DE
STUFB6      LD    A,' '        ;Stuff a space
      LD    (DE),A
      INC   DE
      DEC   C            ;Count down
      JR    NZ,STUFB6    ;Display trailing spaces
      LD    A,3          ;Stuff the ETX
      LD    (DE),A
      RET
;
;     Routine to get the free space info
;
SPACE0      PUSH  HL          ;Save buf start
      LD    DE,16       ;Index for space
      PUSH  DE
      ADD   HL,DE
      CALL  SPACE       ;Get the space data
      POP   BC          ;  name & date
      POP   DE          ;Nos whift in the
      LD    HL,DIRBUF$+0D0H   ;  disk name and date
      LDIR
      XOR   A
      RET
;
SPACE CALL  @GATRD            ;Read GAT
      RET   NZ          ;Ret on GAT read error
      PUSH  IY
      CALL  @GTDCT            ;Get DCT vector
      EX    DE,HL       ;User bufr ptr to DE
      LD    H,0         ;P/u highest # cylinder
      LD    L,(IY+6)    ;  & adjust for 0 offset
      INC   HL
      LD    A,(IY+8)    ;P/u # of sectors/granule
      AND   1FH         ;Mask out bits 5-7
      INC   A           ;Adjust for 0 offset
      PUSH  AF          ;Save # of sectors/gran
      PUSH  DE          ;Save user bufr ptr
      LD    E,A
      LD    A,(IY+8)    ;Now use grans/cyl
      AND   0E0H        ;Mask out bits 0-4
      RLCA              ;  & shift to bits 0-2
      RLCA
      RLCA
      INC   A           ;Adj for 0 offset
      CALL  @MUL8       ;Calc # of sectors/cyl
      BIT   5,(IY+4)    ;Double-sided?
      JR    Z,$+3       ;Bypass if one-sided
      ADD   A,A         ;  else double the count
      POP   BC          ;Rcvr user buf ptr
      CALL  DOMUL16           ;Calculate total sectors
      INC   HL          ;Bump to next buf posn
      PUSH  HL          ;  & save pointer
      LD    HL,DIRBUF$  ;Pt to start of GAT
      LD    DE,0        ;Init gran counter
      LD    A,(DIRBUF$+0CCH)  ;P/u excess cyls
      ADD   A,35        ;Add base # cyls
      LD    B,A         ;Set a loop counter
PUGAT LD    A,(HL)            ;P/u GAT byte
KEEP7 SCF               ;Keep bit 7 set
      RRA               ;Slide gran bit to carry
      JR    C,BYTEND?   ;Ignore if in use
      INC   DE          ;Free, bump gran counter
BYTEND?     CP    0FFH        ;End of byte?
      JR    NZ,KEEP7    ;Loop if not
      INC   L           ;Bump GAT byte pointer
      DJNZ  PUGAT       ;Loop for # cyls
      EX    DE,HL       ;# free grans -> HL
      POP   BC          ;Pop user bufr ptr
      POP   AF          ;Rcvr # of sectors/gran
      POP   IY
DOMUL16     CALL  @MUL16            ;Calc # of free sectors
      LD    H,B         ;Cvrt # of free sectors
      LD    D,L
      LD    L,C         ;  to free spc in K by
      LD    E,A
      INC   DE          ;  dividing the # by 4
      INC   DE          ;Round up adjustment
      SRL   D           ;Divide 16 bit reg by 2
      RR    E
      SRL   E           ;  & divide again
      RR    E
      LD    (HL),E            ;Stuff the value
      INC   HL
      LD    (HL),D
      RET
;
;     Read the hash index table
;
HITRD1      LD    HL,DIRBUF$  ;Pt to System dir bufr
      PUSH  BC
      PUSH  DE
      CALL  @DIRCYL           ;Dir cyl to reg D
      LD    E,1         ;Sector one
      CALL  @RDSSC            ;Read System sector
      POP   DE
      POP   BC
      LD    A,22        ;"HIT read error"
      RET
;
;     Routine to locate the address of a module
;     DE => pointer to module name
;     HL <= address of module start if found
;     DE <= address of end of module name +1 if found
;      Z <= set if found, else NZ & A=error code #8
;
GTMOD PUSH  BC          ;Save this reg pair
      LD    C,0FFH            ;Init length counter
      PUSH  DE          ;Save name start
GTM1  INC   C           ;Bump counter
      LD    A,(DE)            ;Search for end-of-name
      INC   DE
      CP    ' '+1
      JR    NC,GTM1
      POP   DE          ;C = length of name
;
;     Start search at system core
;
      LD    HL,@$SYS    ;Pt to low driver Zone
;
;     Loop through core searching names
;
```

```
GTM2  LD    A,H          ;Are we currently
      CP    @BYTEIO>8    ;  the driver zone?
      JR    NC,GTM2A     ;No - check High memory
;
;     In the Driver zone - is it allocated?
;
      PUSH  BC           ;Save BC
      LD    BC,(DVRHI$) ;P/u next available
      OR    A            ;  addr in driver zone
      PUSH  HL           ;Is this module
      SBC   HL,BC        ;  accounted for in
      POP   HL           ;  the driver zone?
      POP   BC
      JR    NC,GTM8            ;No - get out of d/z
;
;     Check the module for legal header
;
GTM2A LD    A,(HL)             ;Ck for "JR xx"
      CP    18H
      JR    NZ,GTM7            ;Exit if no JR opcode
      PUSH  HL           ;Save pointer to start
      INC   HL           ;Advance 4 bytes to
      INC   HL           ;  length of name
      INC   HL
      INC   HL
      LD    A,(HL)             ;P/u length field
      AND   0FH          ;Strip flags
      CP    C            ;Lengths match?
      JR    NZ,GTM5
      INC   HL           ;Point to start of name
      LD    B,A          ;Set loop counter
      PUSH  DE           ;Save user's name ptr
GTM3  LD    A,(DE)             ;Compare the name
      CP    (HL)         ;  strings
      JR    NZ,GTM4            ;Go on a mismatch
      INC   HL
      INC   DE
      DJNZ  GTM3         ;Loop for B=length
      EX    DE,HL        ;Name +1 to DE
;
;     Found a match - exit with info
;
      POP   HL           ;Keep DE to name end +1
      POP   HL           ;Module start address
      POP   BC           ;Reg restoral
      XOR   A            ;Set Z-flg to show
      RET                ;  found
;
;     No match - loop to next module
;
GTM4  POP   DE
GTM5  POP   HL
      INC   HL           ;Point to last byte
      INC   HL           ;  used
      LD    A,(HL)             ;P/u lo-order of addr
      INC   HL
      LD    H,(HL)             ;P/u hi-order of addr
      LD    L,A
GTM5A INC   HL          ;Bump to next address
      LD    A,H         ;Ck for wrap to zero
      OR    L
      JR    NZ,GTM2           ;Loop if not through
GTM6  POP   BC          ;Restore reg BC
      LD    A,8         ;"Device not avail...
      OR    A           ;Set NZ to show error
      RET
;
;     Found non-JR opcode - Advance to high memory?
;
GTM7  LD    A,H         ;Past driver core?
      CP    @BYTEIO>8
      JR    NC,GTM6           ;Exit with "not found"
GTM8  LD    HL,(HIGH$)  ;  else p/u himem pointer
      JR    GTM5A       ;  & hup to it if in use
;
;     Check a drive for availability
;
CKDRV PUSH  IY          ;-We use IY in disk I/O
      CALL  @GTDCT            ;Get driver routine addr
      LD    A,(IY+0)    ;P/u drive vector
      CP    0C3H        ;JP opcode = drv enabled
      JP    NZ,CKDR5    ;Bypass if disabled
      PUSH  HL
      PUSH  DE
      LD    A,(IY+6)    ;Make sure the current
      CP    (IY+5)            ;  cyl count is in range
      JP    NC,CKDRV1   ;Go if in range
      CALL  @RSTOR            ;Issue FDC RESTORE cmd
      JP    NZ,CKDR7A   ;Go if error
;
CKDRV1      LD    D,(IY+5)    ;P/u current track
      LD    E,0         ;Set for sector 0
      CALL  @SEEK       ;Set track info to FDC
      JR    NZ,CKDR7A   ;Go if error
      CALL  @RSLCT            ;Wait until not busy
      JR    NZ,CKDR7A   ;Not there - ret NZ
      BIT   3,(IY+3)    ;If hard drive, bypass
      JR    NZ,CKDR3A   ;  GAT data update
      BIT   4,(IY+4)    ;If ALIEN ctrlr, bypass
      JR    NZ,CKDR2B   ;  test of index pulses
      IF    @MOD4
      LD    A,(FDDINT$) ;Check 'SMOOTH' Option
      OR    A
      LD    A,09        ;Set MSB of countdown
      JR    Z,INTRON    ;INTs on if not 'Smooth'
      SRL   A           ;Divide the count by two
      DI
      ENDIF
      IF    @MOD2
      LD    A,20
      ENDIF
INTRON      LD    (CDCNT+1),A ;Store in 'LD H,nn' opcode
      LD    HL,32       ;Set up count (short)
;
```

```
;     Test for diskette in drive & rotating
;
CKDR1 CALL  INDEX       ;Test index pulse
      JR    NZ,CKDR1    ;Loop until pulse
      BIT   7,(IY+4)    ;Check CKDRV inhibit bit
      JR    NZ,CKDR2B   ;  -if on skip index test
CDCNT LD    H,00H       ;CKDRV counter (long)
      ;Count set from above
CKDR2 CALL  INDEX       ;Test index pulse
      JR    Z,CKDR2           ;Jump on no index
      IF    @MOD4
      EI                ;OK for INTs now
      ENDIF
      LD    HL,0020H    ;Index off wait (short)
CKDR2A      CALL  INDEX
      JR    NZ,CKDR2A   ;Jump on index
;
;     Diskette is rotating!!
;
CKDR2B      PUSH  AF          ;Save FDC status
      CALL  @DIRCYL           ;Get directory track in D
      LD    HL,SBUFF$   ;Pt to Sys HIT bufr
      LD    E,L         ;Sector 0 for GAT
      CALL  @RDSSC            ;Read the GAT
      JR    NZ,CKDR7    ;Jump on error
      LD    HL,(SBUFF$+0CCH)  ;P/u excess tracks
      LD    A,22H       ;Add offset of 34
      ADD   A,L
      LD    (IY+6),A    ;Max track # to DCT
      RES   5,(IY+4)    ;Set to side 0
      BIT   5,H         ;Test double-sided
      JR    Z,CKDR3           ;Jump if only single
      SET   5,(IY+4)    ;Set for side 2
CKDR3 POP   AF          ;Recover FDC status
CKDR3A      RLCA              ;Shift write prot to 7
      OR    (IY+3)            ;Merge Soft WP bit
      AND   80H         ;Mask unwanted
      ADD   A,A         ;Write prot to C-flg
;
CKDR4 EQU   $
      EI
      POP   DE
      POP   HL
CKDR5 POP   IY
      RET
;
INDEX LD    A,H         ;Check countdown timer
      OR    L
      JR    Z,CKDR7           ;Err exit if 0
      DEC   HL
      CALL  @RSLCT            ;Reselect drive
      BIT   1,A         ;Test index pulse
      RET
;
CKDR7 POP   AF
CKDR7A      LD    A,8         ;Set device no avail
      OR    A           ;Set NZ
      JR    CKDR4       ;Exit
;
LILBUF      DS    18
LAST  EQU   $
      IF    $.GT.DIRBUF$
      ADISP 'ERROR: Module too big'
      ENDIF
      ORG   MAXCOR$-2
      DEFW  LAST-SYS12  ;Overlay size
;
      END   SYS12
```

SYS13/ASM

```
;SYS13/ASM - LS-DOS 6.2
      ADISP '<SYS13 - LS-DOS 6.2>'
;
CR    EQU   13
LF    EQU   10
*GET  'COPYCOM:1'       ;Copyright message
;
      ORG   1E00H
;
SYS13 JR    START
      DS    32%0        ;Slack
;
START AND   70H         ;Strip bit 7
      CP    70H         ;Go if 0111 0000
      JP    Z,NOCMD           ;  to no <*> command
NOSYS13     LD    A,101       ;Get flags
      RST   40
      LD    (IY+'E'-'A'),0    ;Reset ECI flag
      LD    HL,NXCI$    ;"No ECI present...
      LD    A,12        ;Display and log it
      RST   40
      XOR   A           ;Z=no error
      RET
;
NOCMD LD    HL,NOCMD$   ;"No sys13...
      LD    A,12        ;Display and log it
      RST   40
      XOR   A
      RET
;
NXCI$ DB    'No Extended Command Interpreter Present, as SYS13 '
      DB    LF,CR
NOCMD$      DB    'No command <*> present, as SYS13 '
      DB    LF,CR
;
*LIST OFF
      DEFS  -$&0FFH%0
      DEFS  256%0
*LIST ON
LAST  EQU   $-1
;
      END   SYS13
□
```