COLD-START LOADER FOR ADAM(tm) CP/M(tm)
disassembly and comments by
George A. Havach
[CIS #70337,3062]

addr	hexcode(s)	opcode	comment

This_Way_In:

```
;On system reset, the Adam's Memory Input-Output Controller (MIOC) selects
;the SmartWriter ROMs in lower memory and 32K of intrinsic RAM in upper
;memory, and jumps to the EOS_BOOT routine at the beginning of SmartWriter.
;This code first checks for expansion ROM (in Expansion Connector #2) and
;jumps to it if present; otherwise, it loads EOS (the built-in Elementary
;[or "Extended"] Operating System) from ROM into upper RAM and jumps to
;EOS_START.  This initialization routine then polls for devices on AdamNet
;and checks for media in one of the active drives, in this order:  Disk
;Drive 1 (DSK1), Disk Drive 2 (DSK2), Data-Pack Drive 1 (DDP1), Data-Pack
;Drive 2 (DDP2).  If it finds a validly formatted media, it reads block 0,
;loads it into memory starting at address C800H, and jumps to that location,
;effectively turning over control of the system to any code present there.
;
;That's where this Cold-Start Loader comes in; residing in block 0 of a
;CP/M-formatted and SYSGEN'ed data pack or disk, it will automatically be
;run on system reset and is expected to load the CP/M operating system into
;place and then pass control to it.  As we shall see, the Loader first
;installs the BIOS (Basic Input-Output System) and then jumps to its beginning
;address, the cold-start entry point, which itself contains a jump instruction
;to the BOOT routine that proceeds to load in the rest of CP/M, i.e., the CCP
;(Console-Command Processor) and BDOS (Basic Disk-Operating System).

C800  18 01        JR C803H        ;skip over next address
C802  E5           DEFB E5H        ;not used

;There's space here for an opening instruction that could be a jump ('JP')
;to an absolute instead of a relative address--i.e., three bytes long instead
;of two--probably as an artifact of assembly.  Digital Research's LINK-80
;Linking Loader, for example, will normally reserve the first three bytes of
;an output file for a jump instruction to the entry point of the program
;proper.

_Prevent_Screen_Interrupts:

;In case we're booting from SmartBASIC or any other condition in which the
;nonmaskable interrupt (NMI) from the TMS9918/9928 Video Processor is enabled,
;we provide here for an immediate return from interrupt servicing to preserve
;the Z80's registers.  The periodic hardware interrupt is disabled on VRAM
;initialization during CP/M operation.

C803  3E C9        LD A,C9H        ;install NMI-service routine...
C805  32 66 00     LD (0066H),A    ;of a simple 'RET' (humpfh!)

_Set_Up_User_Stack:
C808  31 FF C7     LD SP,C7FFH     ;locate stack at bottom of Loader code
```

```
;C800H would do just as well, since the stack pointer is always decremented
 first before any data is stored.  For that matter, the first bytes of the
;Loader could even be overwritten by the stack, with no ill effects whatever.

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Bit_Test_For_Boot_Device:

;On entry from EOS, the Z80 CPU's B register contains the device ID (port
;address) of the data source accessed by EOS_START on system startup:
;
;     DSK1 = 04H (00000100B)       DDP1 = 08H (00001000H)
;     DSK2 = 05H (00000101B)       DDP2 = 18H (00011000H)
;
;The codes assigned to the various drives by this routine (used later by the
;BIOS for CP/M drive designation) are:
;
;     0 = DDP1, 1 = DDP2, 2 = DSK1, 3 = DSK2

_Check_Bit_Three_Of_Boot_Device_ID:
C80B  AF            XOR A           ;zero out the A register (harrumpfh!)
C80C  CB 58         BIT 3,B         ;source on DDP1 or DDP2?
C80E  20 09         JR NZ,C819H     ;yup, better check bit 4
                                    ;(only DDP2 has this bit set!)
C810  3E 02         LD A,02H        ;nope, then source must be a disk drive
C812  CB 40         BIT 0,B         ;is it DSK1?
C814  28 07         JR Z,C81DH      ;yup, so A = 2 and exit
C816  3C            INC A           ;nope, then it must be DSK2...
C817  18 04         JR C81DH        ;...so A = 3 and exit
__Check_Bit_Four:
C819  CB 60         BIT 4,B         ;source on DDP2?
C81B  20 F9         JR NZ,C816H     ;yup, so A = 1
                                    ;nope, then it must be DDP1, so A = 0

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Set_Page_Zero_Parameters:

;One of the tasks of the Cold-Start Loader is to initialize the values of
;several one-byte parameters located in page zero (addresses 0000-00FFH) of
;intrinsic RAM, for later use by the BIOS in selecting available drives:

     Is_RAM_Disk_There      EQU 004DH    ;00H = 'TRUE', FFH = 'FALSE'
     Boot_Device_ID         EQU 004EH
     Boot_Device_Designator EQU 004FH

C81D  32 4F 00      LD (004FH),A    ;store drive code at Boot_Drive
C820  3E FF         LD A,FFH        ;store default value of 'FALSE'...
C822  32 4D 00      LD (004DH),A    ;at Is_RAM_Disk_There
C825  78            LD A,B          ;get boot-device ID into A register...
C826  32 4E 00      LD (004EH),A    ;and store it at Boot_Device_ID

 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Install_BIOS:

;We can't use EOS' own block-read utilities to overwrite it with the CP/M
```

```
;system, so we have to resort to direct DCB access to get the BIOS resident
;in upper memory.  Once installed, the BIOS will load the CCP and BDOS into
 place after the Cold-Start Loader jumps to it.

_Set_Up_To_Read_BIOS:
C829  26 06         LD H,06H          ;initialize block count (6 needed)
C82B  01 01 00      LD BC,0001H       ;initialize block number
                                      ;(1 thru 6 on the boot device)
C82E  11 00 DA      LD DE,DA00H       ;initialize destination address
                                      ;(DA00H = start of the BIOS)
__Loop_To_Read_BIOS_Blocks:
C831  E5            PUSH HL           ;save block count
C832  F5            PUSH AF           ;save boot-device ID
C833  C5            PUSH BC           ;save block number
C834  D5            PUSH DE           ;save destination address
C835  CD 55 C9      CALL C955H        ;go get 'em
C838  30 60         JR NC,C89AH       ;oops! read error--start loop over
C83A  D1            POP DE            ;else get back destination address...
C83B  21 00 04      LD HL,0400H       ;increment it...
C83E  19            ADD HL,DE         ;by 1K...
C83F  EB            EX DE,HL          ;and stick it back into DE
C840  C1            POP BC            ;get back block number...
C841  03            INC BC            ;and increment it by 1
C842  F1            POP AF            ;get back boot-device ID
C843  E1            POP HL            ;get back block count...
C844  25            DEC H             ;and decrement it by 1
C845  20 EA         JR NZ,C831H       ;if not last block, loop for next

;= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

;****   TEST FOR 64K MEMORY EXPANDER:   ****

;If this were CP/M 3.0 (Plus), the 64K Memory Expander might be used as a
;double bank of RAM in its own right, extending the Transient-Program Area
;(TPA).  However, CP/M 2.2 can at best make use of it as a "RAM disk" for file
;storage, so another task of the Cold-Start Loader is to initialize it for
;later access by the BIOS.  This operation is accomplished here by, first,
;checking for the presence of extra memory and then loading the disk-driver
;code into place in lowermost expansion RAM.
;
;System memory configuration is selected by addressing a specific port:

     Bank_Switch_Port EQU 7FH

Switch_In_Lower_Expansion_Memory:
C847  3E 02         LD A,02H          ;select lower 32K of expansion RAM...
                                      ;(assuming it's there!)...
C849  D3 7F         OUT (7FH),A       ;and upper 32K of intrinsic RAM

_Expansion_RAM_Test:
C84B  21 84 00      LD HL,0084H       ;arbitrary starting point in page zero
C84E  11 00 04      LD DE,0400H       ;incremental amount (1K)
C851  06 05         LD B,05H          ;loop counter (just the first 5K...
                                      ;should be enough to tell)
__Loop_To_Test_Expansion_RAM:
C853  7E            LD A,(HL)         ;peek an address
C854  4F            LD C,A            ;store its byte value in C register
C855  EE 55         XOR 01010101B     ;use mask of alternating 0's and 1's
```

```
C857  77           LD (HL),A       ;try poking masked value back again
C858  BE           CP (HL)         ;do we have "live" memory or not?
C859  20 32        JR NZ,C88DH     ;nope, no Memory Expander present
C85B  EE 55        XOR 01010101B   ;else restore previous bit pattern...
                                   ;in A register...
C85D  71           LD (HL),C       ;and replace original byte value in memory
C85E  BE           CP (HL)         ;do the two numbers agree?
C85F  20 2C        JR NZ,C88DH     ;nope, get out
C861  19           ADD HL,DE       ;else increment pointer...
C862  10 EF        DJNZ C853H      ;and loop to test for more


Initialize_RAM_Disk:

;Because the RAM disk might conceivably contain preloaded code on a cold
;boot (see cartridge-downloading routine below), the authors of the Cold-
;Start Loader (and the BIOS) included a scheme whereby this code could be
;protected from being wiped out on system initialization.  So, if the Loader
;happens to find a particular "word" at 0082H in lower expansion RAM, it will
;skip initializing the RAM-disk directory and first 1K block of user storage.
;The presence of this "secret word" at this location thus ensures that any
;data residing in the RAM disk will survive even a cold boot intact.
;
;reserved page-zero locations:

     BIOS_Return_Address EQU 0080H
     Initialize_Protect  EQU 0082H

Check_For_Protection_Code_1:
C864  01 E9 14     LD BC,14E9H     ;code for "occupied" RAM disk
C867  2A 82 00     LD HL,(0082H)   ;get word at reserved location
C86A  ED 42        SBC HL,BC       ;anybody home?
C86C  28 0D        JR Z,C87BH      ;yup, please don't disturb residents

_Blank_Out_First_Two_User_Blocks:

;Since the first 2K must be overhead for system use, the CP/M directory begins
;here ("track 0"), and so the maximum capacity (user space) of the RAM disk
;is actually a full 62K, instead of the 56K allowed in the Coleco BIOS!

C86E  21 00 08     LD HL,0800H     ;else point to start of user space
C871  36 E5        LD (HL),E5H     ;"blank" character (used in formatting)
C873  54           LD D,H          ;get beginning address...
C874  5D           LD E,L          ;into DE...
C875  13           INC DE          ;and increment it to next address up
C876  01 FF 07     LD BC,07FFH     ;byte count (one less than 2K)
C879  ED B0        LDIR            ;clever way to fill a block of memory...
                                   ;with the same byte value!

_Transfer_Disk_Drivers:
C87B  21 A1 C8     LD HL,C8A1H     ;now transfer...
C87E  11 00 01     LD DE,0100H     ;the RAM-disk-driver routines...
C881  01 3F 00     LD BC,003FH     ;(all 63 bytes of them!)...
C884  ED B0        LDIR            ;to their proper place in expansion memory
__Set_Up_For_Return:
C886  21 8C C8     LD HL,C88CH     ;pointer to return address in intrinsic RAM
;NOTE:  At this point, the value in the C register will be 00H.
```

```
_Check_For_CP/M_Program_Cartridge:
C889  C3 15 01      JP 0115H          ;check for CP/M-formatted ROM cartridge

;The driver code just transferred to the Memory Expander will take care of
;returning us to the main line of code that follows.

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Exit_To_CP/M:
;on entry:  C = 00H if 64K Memory Expander is present

_Memory_Expander_Present:
C88C  21            DEFB 21H          ;entry point after ROM-cartridge check
                                      ;(i.e., RAM disk already initialized)

;On entry at this point, this and the next two bytes are read as 'LD HL,FF0EH'
;(i.e., harmless nonsense code), so the C register's value is preserved, and
;the page-zero parameter Is_RAM_Disk_There gets loaded with 00H (= 'TRUE').
;Otherwise, on entry at the next address, the C register gets loaded with the
;default value of FFH (= 'FALSE'), which gets transferred to Is_RAM_Disk_There.

_No_Memory_Expander_Present:
C88D  0E FF         LD C,FFH          ;entry point after failed RAM test
                                      ;(i.e., no Memory Expander present)
__Restore_Normal_Memory_Configuration:
C88F  3E 01         LD A,01H          ;select upper and lower 32K...
C891  D3 7F         OUT (7FH),A       ;of intrinsic RAM (normal configuration)
__Update_RAM_Disk_Status:
C893  79            LD A,C            ;store value currently in C register...
C894  32 4D 00      LD (004DH),A      ;at Is_RAM_Disk_There

_Run_CP/M:
C897  C3 00 DA      JP DA00H          ;go to BIOS cold-start entry point (BOOT)
                                      ;* * * * * * * * * * * * * *
                                      ;*  ...and away we go!!  *
                                      ;* * * * * * * * * * * * * *

_Error_On_Reading_BIOS_Block:
C89A  D1            POP DE            ;get back destination address
C89B  C1            POP BC            ;get back block number
C89C  F1            POP AF            ;get back boot-device ID
C89D  E1            POP HL            ;get back block count
C89E  C3 31 C8      JP C831H          ;go back and try again

;!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

;**************************
;*   RAM-DISK DRIVERS:    *
;**************************

;The next 63 (3FH) bytes, at addresses C8A1H through C8DFH, are designed for
;transfer to the lower 32K bank of expansion RAM (if present), starting at
;address 0100H, where they will look like this:

Move_A_Block_To_RAM_Disk:

;on entry (for later use by the BIOS):
;   HL = starting address of data source
```

```
            (RAM-disk I/O buffer at 0400-07FFH in lowermost expansion RAM)
;   DE = starting address of destination (also in expansion RAM)
    BC = return address after transfer (in upper 32K of intrinsic RAM)

_Switch_In_Full_Expansion_Memory:
0100  3E 0A         LD A,0AH          ;select upper and lower 32K...
0102  D3 7F         OUT (7FH),A       ;of expansion RAM

;Since intrinsic RAM is now switched out, the block to be moved must already
;have been transferred to the RAM-disk I/O buffer.  Thus, access of the RAM
;disk by the BIOS is a two-step operation:  first, the data block is stored
;temporarily in the RAM disk's I/O buffer (at 0400-07FFH); next, after the
;BDOS allocates disk space based on the current directory, the following
;routine is called to move the data block accordingly.

_Transfer_Data_Block:
0104  ED 43 80 00   LD (0080H),BC     ;store return address at reserved location
0108  01 00 04      LD BC,0400H       ;1K to transfer
010B  ED B0         LDIR              ;done!

_Switch_In_Upper_Intrinsic_Memory:
010D  3E 02         LD A,02H          ;select lower 32K of expansion RAM...
010F  D3 7F         OUT (7FH),A       ;and upper 32K of intrinsic RAM
__Return_To_Caller:
0111  2A 80 00      LD HL,(0080H)     ;get return address from reserved location
0114  E9            JP (HL)           ;exit back to caller


heck_For_ROM_Cartridge:

;Coleco provided code here for downloading a CP/M-compatible ROM cartridge to
;the RAM disk on cold boot; up to 30K (including a directory) of programs
;could be accommodated in this way on a fast-load basis.
;
;on entry:  HL = return address (in upper 32K of intrinsic RAM)

_Check_For_Protection_Code 2:
0115  22 80 00      LD (0080H),HL     ;store return address at reserved location
0118  2A 82 00      LD HL,(0082H)     ;check for protection code...
011B  11 E9 14      LD DE,14E9H       ;at reserved location in page zero
011E  B7            OR A              ;clear carry flag
011F  ED 52         SBC HL,DE         ;anybody home?
0121  28 EE         JR Z,0111H        ;yup, so leave data undisturbed and exit

_Select_Cartridge_ROM:
0123  3E 0E         LD A,0EH          ;else select lower 32K of expansion RAM...
0125  D3 7F         OUT (7FH),A       ;and upper 32K of cartridge ROM

_Check_For_Cartridge_Identifier:

;The CP/M program cartridge is designed to contain an identifier code in the
;first two bytes of ROM, just like a ColecoVision game cartridge (which has
;55AAH).

0127  2A 00 80      LD HL,(8000H)     ;test for CP/M-cartridge identifier...
012A  11 53 CA      LD DE,CA53H       ;at the beginning of cartridge ROM
012D  B7            OR A              ;clear carry flag
012E  ED 52         SBC HL,DE         ;correct identifier present?
```

```
0130  20 DB       JR NZ,010DH      ;nope, exit and restore...
                                   ;previous memory configuration

_Download_Cartridge_To_RAM_Disk:
0132  21 02 80    LD HL,8002H      ;else transfer first 30K of cartridge...
0135  11 00 08    LD DE,0800H      ;(i.e., 1K directory plus 29K of programs)
0138  01 00 78    LD BC,7800H      ;to blocks 2 through 31 (first block...
013B  ED B0       LDIR             ;is 0!) of the RAM disk
__Return_To_System:
013D  18 CE       JR 010DH         ;back to caller

;||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

;+++++++++++++++++++++++++++++
;+   EL COPYRIGHT NOTICE:    +
;+++++++++++++++++++++++++++++

C8E0  20 20 20 20  DEFB            '                          '
      20 20 20 20                  ;[lots of wasted space here!]
      20 20 20 20
      20 20 20 20
      20 20 20 20
      20 20
C8F6  49 6E 74 65  DEFB            'Internal Rev. 1.50  July 5, 1984'
      72 6E 61 6C
      20 52 65 76
      2E 20 31 2E
      35 30 20 20
      4A 75 6C 79
      20 35 2C 20
      31 39 38 34
C916  20 20 20 20  DEFB            '    (c) 1983,1984, '
      28 63 29 20
      31 39 38 33
      2C 31 39 38
      34 2C 20
C929  43 6F 6C 65  DEFB            'Coleco Industries Inc. '
      63 6F 20 49
      6E 64 75 73
      74 72 69 65
      73 20 49 6E
      63 2E 20

;------------------------------------------------------------------------------

;*********************
;*   BLOCK READER:   *
;*********************

;This long and fairly involved routine basically sets up to make two reads
;of each requested BIOS block into a temporary buffer (if the first read is
;successful, the second will simply rewrite the same block to the same area
;of memory).  If no errors are detected, we then request a read of the next
;sequential block before returning to the main loop.  So, on the next time
,through, we check to see if we didn't already start to read the requested
;block before; if so, then we close out the first read, do a second read,
;and again try for the next block if no error.
```

```
Workspace_For_Read_Routines:

 Current_Device:
C940  00            DEFS 1      ;loaded with boot-device ID on first pass

Current_Block:
C941  00 00         DEFS 2      ;holds currently requested BIOS block number

Next_Device:
C943  00            DEFS 1      ;loaded with boot-device ID on setting up to read
                                ;next block; zeroed out on each read of a block

Next_Block:
C944  00 00         DEFS 2      ;loaded with next BIOS block number after
                                ;requested one has been read twice

Check_If_Block_Was_Read_As_Next_Block:

;This subroutine, evidently added later and somewhat out of place here, is
;called by the routine below to check whether we haven't already started to
;read the requested BIOS block on the previous pass.
;
;on entry:
;     C = boot-device ID
;    DE = block number
;    HL = pointer to Next_Device
;on exit:   zero flag set if Next_Block same as block number, clear if not

C946  E5            PUSH HL         ;save pointer
C947  7E            LD A,(HL)       ;get Next_Device into A register
C948  B9            CP C            ;same as boot-device ID?
C949  20 08         JR NZ,C953H     ;nope, exit
C94B  23            INC HL          ;else point to lo byte of Next_Block
C94C  7E            LD A,(HL)       ;get value into A register
C94D  BB            CP E            ;same as for block number?
C94E  20 03         JR NZ,C953H     ;nope, exit
C950  23            INC HL          ;now, point to hi byte of Next_Block
;The read routine used here is perfectly general, so the block number could be
;any 16-bit value from 0000H to FFFFH.
C951  7E            LD A,(HL)       ;get value into A register
C952  BA            CP D            ;same as for block number? (set flags)
C953  E1            POP HL          ;get back pointer
C954  C9            RET             ;back to caller


Set_Up_To_Read_BIOS_Block_From_Boot_Device:

C955  D5            PUSH DE         ;save destination address (again!)
C956  C5            PUSH BC         ;save block number
C957  F5            PUSH AF         ;get boot-device ID...
C958  C1            POP BC          ;into B register...
C959  D1            POP DE          ;and block number into DE

 Check_If_First_Time:

;This test will turn up true only on the first CALL to this routine, in which
;case we skip checking for a block retry and set up to read the first block.
```

```
C95A  3A 40 C9      LD A,(C940H)     ;get Current_Device into A register
C95D  B8            CP B             ;is it same as boot-device ID?
C95E  20 08         JR NZ,C968H      ;nope, go set up for first block read

_Check_If_Retry:

;If perchance there is a read error on a requested block, we will be returned
;to the main loop and then wind up back here again.  This test will detect a
;retry on the same block number, skip setting up for a read request, and jump
;ahead to transferring the contents of the temporary buffer to the block's
;destination.  At best, we will move the requested block into place and then
;proceed to read the next block; at worst, we will find ourselves looping
;endlessly on a bad block.  In the absence of any error handling, there is
;no exit from the Loader but a jump to the BIOS on completion of the block
;reads.

C960  2A 41 C9      LD HL,(C941H)    ;else get Current_Block into HL
C963  B7            OR A             ;clear carry flag
C964  ED 52         SBC HL,DE        ;is Current_Block same as block number?
C966  28 26         JR Z,C98EH       ;yup, so go move transfer buffer...
                                     ;into place reserved for block number

_Set_Up_For_Current_Block:
C968  48            LD C,B           ;else get boot-device ID...
C969  78            LD A,B           ;into A and C registers...
C96A  32 40 C9      LD (C940H),A     ;and store it at Current_Device
C96D  ED 53 41 C9   LD (C941H),DE    ;store block number at Current_Block
__Check_If_Next_Block:
C971  21 43 C9      LD HL,C943H      ;point HL to Next_Device
C974  CD 46 C9      CALL C946H       ;see if we started to read block before
C977  06 02         LD B,02H         ;load loop counter into B register
                                     ;(two reads for each block!)
C979  C5            PUSH BC          ;save loop counter
C97A  28 07         JR Z,C983H       ;yup, so end first read
C97C  C1            POP BC           ;nope, get back loop counter

_Loop_To_Start_Read_Block:
C97D  C5            PUSH BC          ;save loop counter
C97E  CD C8 C9      CALL C9C8H       ;start a block read from boot device
C981  30 42         JR NC,C9C5H      ;error detected, so exit
__End_Read_Block:
C983  3A 40 C9      LD A,(C940H)     ;else get Current_Device into A register
C986  CD EC C9      CALL C9ECH       ;end read (tranfer block to RAM)
C989  30 3A         JR NC,C9C5H      ;error detected, so exit
C98B  C1            POP BC           ;looks good, so get back loop counter
C98C  10 EF         DJNZ C97DH       ;loop to do a second read

_Transfer_BIOS_Block_Into_Place:

;Instead of loading each BIOS block directly into place, we first read the
;requested block into a temporary buffer, then transfer it to its final
;destination in RAM.

C98E  D1            POP DE           ;get back destination address
C98F  21 3B EE      LD HL,EE3BH      ;transfer-buffer address
C992  01 00 04      LD BC,0400H      ;1K block to move
C995  ED B0         LDIR             ;that's one down!
```

```
_Set_Up_For_Next_Block:
C997  2A 41 C9      LD HL,(C941H)     ;get Current_Block into HL
 99A  23            INC HL            ;increment block number by 1...
 99B  22 44 C9      LD (C944H),HL     ;and store it at Next_Block
C99E  3A 40 C9      LD A,(C940H)      ;get Current_Device into A register...
C9A1  32 43 C9      LD (C943H),A      ;and store it at Next_Device
C9A4  CD 4F CA      CALL CA4FH        ;now, locate Command/Status byte...
                                      ;of boot device's DCB
C9A7  30 10         JR NC,C9B9H       ;couldn't find it, so exit...
                                      ;without trying to read next block
__Check_If_Command_Completed:
C9A9  7E            LD A,(HL)         ;found it, now get Command/Status byte...
                                      ;into A register
C9AA  B7            OR A              ;set flags accordingly
C9AB  F2 B9 C9      JP P,C9B9H        ;exit if high bit is clear
                                      ;(i.e., command not yet completed)

_Check_Boot_Device_Status:
C9AE  36 01         LD (HL),01H       ;else poke Command/Status byte...
                                      ;with DCB code for 'Request Status'
__Loop_Until_Command_Completed_1:
;Now, we'll "sit" on the Command/Status byte until the boot device responds.

C9B0  7E            LD A,(HL)         ;get Command/Status byte into A register
C9B1  17            RLA               ;rotate hi bit into carry flag
C9B2  30 FC         JR NC,C9B0H       ;loop until it gets set
__Check_If_No_Errors:
C9B4  B7            OR A              ;test value in A register
 9B5  28 07         JR Z,C9BEH        ;was hi bit the only one set?

;A value of 80H here indicates 'Command Complete with No Errors.'  One
;successful block read encourages us to make a quick try for the next block
;before returning to the main loop.

C9B7  36 01         LD (HL),01H       ;nope, hit it once more on the way out

_Exit_With_Next_Block_Set_As_Not_Read:
C9B9  AF            XOR A             ;zero out A register...
C9BA  32 43 C9      LD (C943H),A      ;and Next_Device
C9BD  C9            RET               ;back to main loop

_Exit_To_Read_Next_Block:
C9BE  21 43 C9      LD HL,C943H       ;else point HL to Next_Device...
C9C1  C3 CF C9      JP C9CFH          ;and do a read on next block

__Dead_Byte:
C9C4  C1            DEFB C1H

;This looks like another 'POP BC', but it turns out to be useless code,
;since we never arrive here!

_Exit_On_Read_Error:
C9C5  C1            POP BC            ;dump out loop counter
 9C6  C1            POP BC            ;dump out destination address
 9C7  C9            RET               ;back to main loop

;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
Start_Read_Block_From_Boot_Device:

;This routine sets up the registers for DCB access, then requests a block read
;from the boot device, but returns without ending the read (i.e., doing a
;status request) and without transferring the BIOS block into place.
;
;on exit:  carry flag set if block-read request completed, clear if error

_Set_Next_Block_As_Not_Read:
C9C8  21 40 C9      LD HL,C940H       ;point to Current_Device
C9CB  AF            XOR A             ;zero out A register...
C9CC  32 43 C9      LD (C943H),A      ;and Next_Device

_Load_DCB_Code:
C9CF  3E 04         LD A,04H          ;load DCB code for 'Read Data'...
C9D1  32 E6 C9      LD (C9E6H),A      ;into place for use during DCB access
C9D4  1F            RRA               ;shift lo bit rightward to clear carry flag
                                      ;(so it will be significant on exit)

_Set_Up_For_Block_Read:
C9D5  7E            LD A,(HL)         ;get boot-device ID into A register
C9D6  23            INC HL            ;get block number...
C9D7  5E            LD E,(HL)         ;from workspace...
C9D8  23            INC HL            ;into DE
C9D9  56            LD D,(HL)
C9DA  21 3B EE      LD HL,EE3BH       ;transfer-buffer address

;This temporary storage location for the block read is just slightly above
;the destination for the last (sixth) BIOS block, and the BIOS code itself
;ends at address EF24H, so if the seventh block is ever read into the transfer
;buffer, it will overwrite the end of the BIOS.  Actually, however, block 7
;will remain in the DMA buffer without being transferred into RAM until we
;do a status request to end the read.  Furthermore, block 7 will still be
;waiting in the DMA buffer when the BIOS picks up to read the rest of the
;CP/M system (blocks 7-12) into memory, and thus the BOOT routine will be
;that much ahead when it takes over from the Cold-Start Loader.  The net
;effect is no apparent pause in drive activity while booting CP/M!

C9DD  01 00 00      LD BC,0000H       ;zero out BC (becomes...
                                      ;hi double-byte of Sector Number)
C9E0  F5            PUSH AF           ;save device number
C9E1  D5            PUSH DE           ;save block number
C9E2  E5            PUSH HL           ;save buffer address
C9E3  CD 22 CA      CALL CA22H        ;read that block!

_DCB_Code:
C9E6  00            DEFS 1            ;loaded with 04H (= 'Read Data')...
                                      ;on entering this routine

_Return_After_Read_Request:
C9E7  E1            POP HL            ;get back buffer address
C9E8  D1            POP DE            ;get back block number
C9E9  C1            POP BC            ;get back device number...
 9EA  78            LD A,B            ;into A register (while preserving flags!)
 9EB  C9            RET               ;back to BIOS-block-read loop

;-------------------------------------------------------------------------------
```

```
CA0F  2F            CPL             ;nope, use complement (00001111B) instead
                                    ;else preserve mask

_Check_For_Device_Error_2:
CA10  01 14 00      LD BC,0014H     ;offset into boot device's DCB...
                                    ;for Device-Dependent Status Flags byte
CA13  E5            PUSH HL         ;save pointer to Command/Status byte
CA14  09            ADD HL,BC       ;add in offset and point HL to Flags byte
CA15  A6            AND (HL)        ;look for any bit set in lo nybble...
                                    ;(or in hi nybble if DDP2)
CA16  E1            POP HL          ;get back pointer to Command/Status byte
CA17  C0            RET NZ          ;exit if flag is waving in Flags byte
__Exit_If_No_Device_Error:
CA18  37            SCF             ;else set carry flag (A-OK!)
CA19  C9            RET             ;back to caller
```

;==========

```
;********************
;*   DCB ACCESS:    *
;********************
```

;Now comes the hard part: bypassing the Elementary Operating System (EOS) ;and getting the Adam to do a direct block read, using the boot device's DCB ;(Device-Communication Block). For the uninitiated, be it known that instead ;of accessing each peripheral device directly through a hardware port address, ;as in most microcomputers, the Adam's Z80 Central Processing Unit (CPU) ;communicates indirectly by way of a memory map located in uppermost RAM at addresses FEC0H through FFFEH (FFFFH is "reserved"), called the PCB/DCB ,area--the heartland of AdamNet, the internal 62.5-kilobit-per-second serial ;network. The PCB (Processor-Communication Block) comprises just 4 bytes ;starting at FEC0H, by which the Z80 communicates with the Master 6801 (at ;port address 0) that manages two-way message transmission over AdamNet. ;Each device (there's room for 15 on the Net, and they're assigned port ;addresses 1 through 15) that answers "Here!" to rollcall at initialization ;(active devices are polled each there's a system reset) gets its very own ;space of 21 (15H) bytes set aside for all I/O operations ("communication"). ;[Let's see...15 times 21 equals 315; that's 13BH, plus FEC3H gives FFFEH-- ;yup!] The DCB's are "stacked" in ascending order by primary device ID:

```
;     01   Keyboard
;     02   SmartWriter Printer
;     04   Floppy-Disk Drive 1
;     05   Floppy-Disk Drive 2
;     08   Data-Pack Drive 1 (same DCB is shared by Data-Pack Drive 2,
;            which has a secondary device ID of 01H)
```

;If a device isn't attached and active, it simply doesn't get a DCB assigned ;to it; that's how we keep track of all the devices currently on AdamNet.

;The Master 6801 (which is a microprocessor in its own right, with 2K of ROM ;and 128 bytes of RAM) is interfaced to the Z80 by the Memory Input-Output ;Controller (MIOC), which periodically strobes the PCB/DCB area by a DMA (Direct Memory Access) process, reading the bit patterns of the Command/ ,Status bytes (the first byte of the PCB and of each DCB). Whenever it sees ;a pattern signaling a command (possible DCB commands are: 01H = Request ;Status, 02H = Reset, 03H = Write Data, 04H = Read Data), it immediately ;signals the Master 6801 to carry out the action requested, without inter-

```
;rupting the CPU.  As soon as the Master (really a slave!) gets a response
;from the selected device, it reports back to the MIOC, which then writes a
 characteristic value back to that same byte (it READS commands and WRITES
,status).  A status byte of 80H (only the high bit set) signifies 'Command
;Complete'; any other bit(s) set signals an error condition (e.g., 9BH =
;'Timed Out').  A value of 00H indicates an 'Idle State' (i.e., device is not
;busy and thus available).
;
;This Command/Status byte is byte 0 of each DCB.  The other 20 bytes describe
;key parameters of each device and its I/O operations.  What's of interest to
;us here are the following:
;
;    1-2    Address of Data Buffer (lo, hi)
;    3-4    Length of Data Buffer (lo, hi)
;    5-8    Block-Device Sector Number (32-bit number: lo, hi, lo, hi)
;             [the drives are block devices; the keyboard and printer are
;             character devices]
;     9     Secondary Device ID (the "hi nybble" of devices like DDP2
;             that have a port address above 0FH)
;    16     Device Address (ID)
;    20     Device-Dependent Status Flags (also written on command completion)
;
;In the block-read routine that follows, the stack is used to store the values
;needed for bytes 1 thru 8; these values are subsequently written directly to
;the boot device's DCB.  Then, the Command/Status byte is "poked" with a value
;of 04H (DCB_RD) to request a read of one block of the BIOS from the boot
;device into a temporary buffer in RAM (at EE3BH), whence it will be later
;transferred to its final location lower down:  block 1, DA00H; block 2,
 DE00H; block 3, E200H; block 4, E600H; block 5, EA00H; and block 6, EE00H.

_Exit_If_Device_Not_Ready_2:
CA1A  C1            POP BC          ;unload the stack
CA1B  C1            POP BC
CA1C  C1            POP BC
CA1D  C1            POP BC
CA1E  C1            POP BC
CA1F  E1            POP HL          ;get back return address (C9E6H)
CA20  23            INC HL          ;increment to point past the 04H there
CA21  E9            JP (HL)         ;back to exit from read-block routine


Request_Block_Read_Using_DCB:

;on entry:
;     A = boot device ID
;    BC = 0000H (high double-byte of Sector Number)
;    DE = block number (low double-byte of Sector Number)
;    HL = buffer address
;on exit:  carry flag set if block-read request completed, clear if error

_Store_Block_Transfer_Data_On_Stack:

;All the pertinent data to be written directly to the boot device's DCB are
 first conveniently stored on the stack for subsequent transfer.

CA22  F5            PUSH AF         ;save boot-device ID
CA23  C5            PUSH BC         ;save high double-byte of Sector Number
CA24  D5            PUSH DE         ;save low double-byte of Sector Number
```

```
CA25  11 00 04      LD DE,0400H       ;1K
CA28  D5            PUSH DE           ;save Length of Data Buffer
CA29  E5            PUSH HL           ;save Address of Data Buffer

;At this point, the stack pointer points to the beginning of all the data
;(stacked in reverse order) that will be transferred (by an 'LDIR') to
;the boot device's DCB in preparation for a block read.  All that remains
;is to determine whether the device is available for a command.

__Check_If_Boot_Device_Ready:
CA2A  CD EC C9      CALL C9ECH        ;is that boot device ready, already?
CA2D  30 EB         JR NC,CA1AH       ;nope, get outta here

_Transfer_Data_To_Boot_Device_DCB:
CA2F  E5            PUSH HL           ;else save pointer to start of DCB...
CA30  EB            EX DE,HL          ;and get destination for transfer into DE
CA31  13            INC DE            ;now, point to byte 1 of DCB...
CA32  21 02 00      LD HL,0002H       ;and point HL...
CA35  39            ADD HL,SP         ;to beginning of data on stack
CA36  01 08 00      LD BC,0008H       ;eight bytes to transfer
CA39  ED B0         LDIR              ;done!

;HL now points to the item on the "top" of the stack, which contains the
;boot-device ID in its high (second) byte.  DE now points to the next byte
;in the DCB, i.e., the Secondary Device ID.

_Check_For_Secondary_Device_ID:
CA3B  23            INC HL            ;point to boot-device ID on the stack...
CA3C  7E            LD A,(HL)         ;and get it into A register
CA3D  23            INC HL            ;now, point to next byte on the stack
                                      ;(i.e., return address from this routine!)
CA3E  0F            RRCA              ;exchange nybbles in A register
CA3F  0F            RRCA
CA40  0F            RRCA
CA41  0F            RRCA
CA42  E6 0F         AND 00001111B     ;mask off hi (old lo) nybble
;The A register now holds the secondary device ID, if any (i.e., for DDP2).
CA44  12            LD (DE),A         ;write value to Secondary Device ID byte...
                                      ;(byte 9) of boot device's DCB

_Set_Up_Stack_For_Return:
CA45  D1            POP DE            ;get back pointer to start of DCB...
CA46  F9            LD SP,HL          ;reset stack pointer to return address...
CA47  EB            EX DE,HL          ;and get start of DCB back into HL
CA48  D1            POP DE            ;get back return address
                                      ;(C9E6H, remember?!)
CA49  1A            LD A,(DE)         ;get code for 'Read Data' stored there...
                                      ;into A register
CA4A  13            INC DE            ;now, point to the NEXT address...
CA4B  D5            PUSH DE           ;and push THAT address back on the stack!
__Read_That_Block:
CA4C  77            LD (HL),A         ;poke Command/Status byte...
                                      ;with DCB code for 'Read Data'

;With everything finally in place, we give the signal for action on the
;boot device.  If all goes well, whir goes the drive, and the block is read
;into the DMA buffer.  The next status request from the device will end the
;read by forcing a DMA transfer of the data block into the designated location
```

```
;(temporary-buffer address) in RAM.

 Return_On_Completion:
CA4D  37              SCF               ;signal for read request completed
CA4E  C9              RET               ;POPs correct return address off stack

;=============================================================================

;****   DCB SERVICE 2:   ****

Locate_Boot_Device_DCB:

;This supporting subroutine searches the DCBs for the one belonging to the
;boot device and then, if it finds it, locates the crucial Command/Status
;byte (byte 0).
;
;on entry:   A = boot-device ID
;on exit:
;   HL = address of Command/Status byte of boot-device's DCB
;   carry flag set if boot device's DCB is found,
;      clear if not found or no devices on AdamNet

_Locate_Command/Status_Byte_Of_Boot_Device_DCB:
CA4F  21 C3 FE        LD HL,FEC3H       ;get number of DCB's defined...
                                        ;(fourth byte of PCB)...
CA52  46              LD B,(HL)         ;into B register
CA53  04              INC B             ;increment it...
CA54  05              DEC B             ;and then restore original value
                                        ;(i.e., check for zero condition)
CA55  28 14           JR Z,CA6BH        ;if no active devices on Net, then exit
CA57  21 D4 FE        LD HL,FED4H       ;else point to Device ID byte (byte 16)...
                                        ;of first device on the Net
CA5A  11 15 00        LD DE,0015H       ;number of bytes in a DCB (offset to next)
CA5D  E6 0F           AND 00001111B     ;mask off hi nybble of boot device ID
                                        ;(i.e., use primary device ID only)
__Loop_To_Locate_Boot_Device_DCB:
CA5F  BE              CP (HL)           ;is this the boot device's DCB?
CA60  28 03           JR Z,CA65H        ;yup, found it
CA62  19              ADD HL,DE         ;nope, look at next DCB
CA63  10 FA           DJNZ CA5FH        ;loop until we find the one we want

_Locate_Command/Status_Byte:
CA65  11 F0 FF        LD DE,FFF0H       ;two's complement of 0010H
CA68  19              ADD HL,DE         ;same as HL - 10H <also sets carry flag!>
;HL now points to the Command/Status byte (byte 0) of the boot device's DCB.
CA69  37              SCF               ;set carry flag to signal success
                                        ;<redundant because of 'ADD' instruction!>
CA6A  C8              RET Z             ;normal exit
__Exit_If_No_Devices_On_Net:
CA6B  3E 9B           LD A,9BH          ;BIOS error code for 'Timeout Error'
                                        ;(but not used by Cold-Start Loader!)
CA6D  B7              OR A              ;clear carry flag
CA6E  C9              RET

;#############################################################################

;++++++++++++++++++++
;+   LEFTOVERS:   +
```