HP110 RAM Expansion Documentation

Samuel H. Chau
Nov 25, 1986

This article describes the procedure to expand the 272K byte RAM capacity of the HP110 to 512K bytes. Do not attempt the modifications if the proper tools are not available, or if you do not have considerable experience in IC soldering and de-soldering, as well as the knowledge to properly handle static-sensitive CMOS digital ICs. The HP110 Service Manual provides a step-by-step procedure for disassembling and re-assembling the HP110. Never attempt disassembly without following instructions. The Service Manual also provide schematics and component locator diagrams to facilitate the modification process.

Tools required:

- -- Grounded, temperature-controlled soldering iron with 0.15" tip and set at 700-800 degrees F.
- -- Anti-static solder sucker or desoldering station.
- -- Anti-static workstation mat with wrist strap.
- -- Flux core solder, preferably with water-soluble flux.
- -- TORX T-8 screwdriver.
- -- Tweezers, needle-nose pliers, small flat-blade screwdriver, Xacto knife wire stripper, IC extractor.
- -- 28 or 30-gauge insulated wire-wrap wire.

ICs required:

(12) NEC 43256-12L 32K*8 CMOS Static RAMs (or Hitachi HD62256-LP12)
(3) 74HC138 CMOS 3-8 line decoders
~~(1) 27C64-25 CMOS EPROM~~

Procedure:

1. Isolate the SYSTEM PCA from the rest of the HP110. Follow disassembly procedures as described in the HP110 Service Manual, p/n 00090-90021.

2. Using a grounded, temperature-controlled soldering iron and an anti-static solder sucker, unsolder and remove ICs U14, U16, U36-51 and U65-67. U36-51 and U66-67 are 28-pin 8K*8 CMOS SRAMs. U65 is a 74HC139 and U14,16 are 74HC138 decoders.

3. Solder twelve (12) 28-pin NEC 43256-12L 32K*8 CMOS SRAM chips onto the SYSTEM PCA, using (now-empty) sockets for U40-51. Sockets for U36-39 and U66-67 should not hold any chips.

4. Solder a new U14 and U16 (both 74HC138s) onto their original sockets, but each with pins 1,2,3 and 5 lifted.

5. Solder a new U65 (74HC138) onto its original socket, but with the following modifications:

   pins 1,2,3,5 & 15 lifted
   pins 11,13 & 14 clipped
   pins 4,6,7,8,9,10,12 & 16 soldered onto the PCA

6. Complete the wiring on all lifted IC pins as shown on the schematic. This includes U14 pins 1,2,3 & 5, U16 pins 1,2,3 & 5 and U65 pins 1-6 & 15. U14,16 and U65 in conjuction performs new address decoding for the 32K*8 RAM chips.

7. Connect pin 1 of all 12 43256-12L RAM chips together and connect them to U7 pin 19. Do not unsolder U7. This provides address line MA15 to the RAM chips.

8. Cut the trace to pin 26 of U40-51 as shown on the component locator diagram. Connect the isolated net to U7 pin 18. This provide address line MA14 to the RAM chips.

9. Cut the short traces connecting U7 pins 1,2 to ground. Connect U7 pin 1 to U11 pin 2. Connect U7 pin 2 to U11 pin 3. This provides buffered CPU address/data lines to latch U7.

10. On the I/O PCA, remove the configuration EPROM (XU28) and replace with a 27C64 correctly programmed to reflect the new 512K byte RAM capacity.

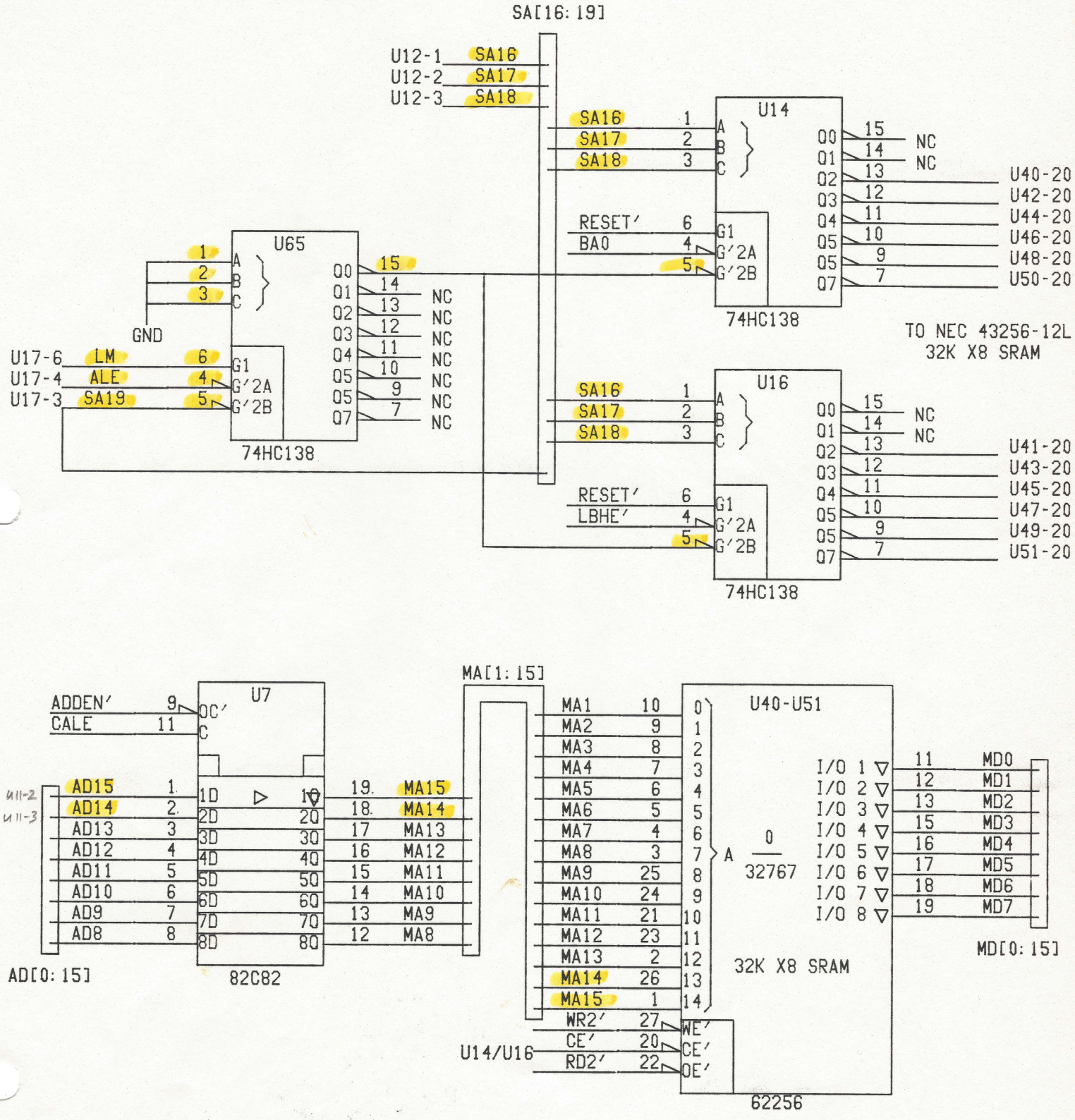
DON'T!!

11. Re-assemble the HP110 according to Service Manual instructions.

WITH WAIT STATE ELIMINATION, NEC V30 CPU: NORTON SI = 2.8+

HP110 512K RAM EXPANSION SCHEMATIC

SAMUEL H. CHAU 11-26-86



Date: Mon, 27 Apr 87 14:49:28 pdt

Hi Jim!

> Yes. Yes. Yes! Yes!! Yes!!! Oh, Please!!!! (Now to get the RAM...)

Well, here it comes, more or less. It's in another e-mail message I'm sending immediately after this one. Assuming you have the HP-110 Programmer's Tool Kit, and therefore MASM.EXE, LINK.EXE and EXE2BIN.EXE, here's a short batch file that automates the assembling of the source:

```
if not exist %1.asm goto :ex
masm %1,,,,
if errorlevel==1 goto :ex
link %1,,,,
if errorlevel==1 goto :ex
exe2bin %1 %1.sys
erase %1.exe
:ex
```

When the batch file finishes up, just make sure RAM110.SYS is sitting somewhere in drive A:. Then add the entry

```
device = ram110.sys
```

to the CONFIG.SYS file, and reboot. Presto, you have an extra 240 Edisk! The drive name will be after the external drives selected in the PAM config menu. And you don't have to run FORMAT since the driver handles it the first time it's installed.

Jim, you will be the 4th person to use the extra RAM in the '110. My manager Mike was the first to get it, and I gave a copy to Lee Woodriff over the weekend. Incidentally, the driver defines only 32 root directory entries for the Edisk. If you need more, just increase the Ndire EQUate in the source file by an appropriate multiple of 16 and re-assemble.

Do have fun with it! Actually, the hardware mods are even more fun, as I'm sure you'll agree. To facilitate that, I'm also sending a RAM test routine to help check out the address decoding.

Enjoy! (When do I get a ride on the Cessna again? 8-)

CDs <> More CDs
Sam

Date: Mon, 27 Apr 87 14:49:54 pdt
From: Samuel Chau <hpsrla!hpccc!samc>
To: hpsrla!hpsadla!jimh

```
                name    EDISK
                page    60,132
                title   NOMAD RAM Disk Driver

;--------------------------------------------------------------------------
; EDISK - RAM disk Emulator for the HP-110
;
; Sam Chau  04/20/87    10:00 pm  a
;
; This is an installable Edisk driver for a NOMAD (HP-110) expanded to
; 512K bytes of internal RAM.  The hardware expansion fills a void in the
; CPU address space (44000-7FFFF hex) with RAM, which is used by the driver
; to define a 240K byte Edisk.  Installation will fail on an unmodified
; HP-110.
;
; Note that the Edisk controlled by this driver is *NOT* initially formatted;
; the FORMAT utility resident in drive B: must be used to initialize the
; Edisk.  However, once FORMAT is run, the drive contents will be valid
; and preserved until the machine loses battery power or FORMAT is run
; again.  Re-booting the machine will not affect the drive contents.
; Since this driver is re-installed during a reboot, a copy of the driver
; must be kept in drive A: at all times.  This ensures that the Edisk will
; still be accessible after rebooting is completed.
;
; To install the Edisk driver, add the line
; DEVICE = RAM110.SYS
; to the CONFIG.SYS file in drive A:.  If CONFIG.SYS doesn't already exist,
; one must be created and edited to contain the above line.  Then, reboot
; HP110 with the <SHIFT><CTRL><BREAK> key sequence.  A message will be
; displayed if installation is successful.  Note that the
; <SHIFT><CTRL><EXTEND CHAR><BREAK> reboot key sequence will prevent the
; Edisk driver from being installed, since the default drive is B: (ROM)
; for this reboot sequence.
;
; The Edisk drive controlled by this driver will come after the two built-in
; drives (A: and B:) and any external drives specified in the P.A.M.
; System Configuration menu.  Since the default PATH environment variable
; does not contain the Edisk drive name, it will be necessary to modify
; the PATH variable if executable files stored in Edisk are to run while the
; default drive is set to A:.
;--------------------------------------------------------------------------

                page

;--------------------------------------------------------------------------
; EQUates, macros, etc.
;--------------------------------------------------------------------------
CODE            SEGMENT BYTE PUBLIC
                ASSUME  CS:CODE,DS:NOTHING,ES:NOTHING
                ORG     0

Max_Cmd         equu    12      ;MS-DOS command code maximum
                                ;this is 12 for MS-DOS 2.X
CR              equu    0dh     ;ASCII carriage return
LF              equu    0ah     ;ASCII line feed
Space           equu    20h     ;ASCII space
EOM             equu    '$'     ;end of message signal (brain-dead MS-DOS)

CmdLen  =       0               ;Length of this command
Unit    =       1               ;Sub unit specifier
CmdCode =       2               ;Command code
Status  =       3               ;Status
Media   =       13              ;Media descriptor
Trans   =       14              ;Transfer address
Count   =       18              ;Block transfer count
BPBaddr =       18              ;BPB address
Start   =       20              ;Block start address

                page

;--------------------------------------------------------------------------
; Device Driver Header
;--------------------------------------------------------------------------
Header          dw      -1,-1           ;link to next device, -1 = end of list
                dw      6000h           ;Device Attribute word
                                        ;Block device
                                        ;IOCTL not supported
                                        ;Non-IBM format
                dw      STRATEGY        ;device "Strategy" entry point
                dw      DRIVER          ;device "Interrupt" entry point
NumDrives       db      0               ;# of Units

;--------------------------------------------------------------------------
; MS-DOS Command Codes dispatch table.
; The "Interrupt" routine uses this table and the
; Command Code supplied in the Request Header to
; Transfer to the appropriate driver subroutine.
;--------------------------------------------------------------------------
CmdTbl:
                dw      $INIT           ;0 = initialize driver
                dw      $MEDIACHECK     ;1 = Media check on block device
                dw      $BUILDBPB       ;2 = build BIOS parameter block
                dw      $EXIT           ;3 = I/O control read
                dw      $READ           ;4 = read from device
                dw      $BUSYEXIT       ;5 = ND_Read    (not implemented)
                dw      $EXIT           ;6 = Inp_Stat   (not implemented)
                dw      $EXIT           ;7 = Inp_Flush  (not implemented)
                dw      $WRITE          ;8 = write to device
                dw      $WRITE          ;9 = write with verify
                dw      $EXIT           ;10 = Outp_Stat  (not implemented)
                dw      $EXIT           ;11 = Outp_Flush (not implemented)
                dw      $EXIT           ;12 = I/O control write

                page

;--------------------------------------------------------------------------
; MS-DOS Request Header structure definition
;
; The first 13 bytes of all Request Headers are the same
; and are referred to as the "Static" part of the Header.
; The number and emaning of the following bytes varies.
; In this "Struc" definition we show the Request Header
; contents for Read and Write calls.
;--------------------------------------------------------------------------
;Request                 struc           ;request header template structure
;                                ;beginning of "Static" portion
;Rlength                 db      ?       ;length of request header
;Unit                    db      0       ;Unit number for this request
;Command                 db      ?       ;request header's command code
;Status                  dw      0       ;driver's return Status word
;                                ;bit 15 = Error
;                                ;bit 10-14      = Reserved
;                                ;bit 9          = Busy
;                                ;bit 8          = Done
```

```
;                               ;bit 0-7        = Error code if bit 15 = 1
;Reserve                db      8 dup (0)       ;reserved area
;                               ;end of "Static" portion, the remainder in
;                               ;this example is for read and Write functions.
;
;Media                  db      ?       ;Media Descriptor byte
;Address                dd      ?       ;memory address for Transfer
;SecCount               dw      ?       ;sector Count value
;Sector                 dw      ?       ;starting sector value
;
;Request                ends            ;end of request header template
;---------------------------------------------------------------------------

                page

;---------------------------------------------------------------------------
; Device Driver "Strategy Routine"
; Each time a request is made for this device, MS-DOS
; first calls the "Strategy routine", then immediately
; calls the "Interrupt routine".
;
; The Strategy routine is passed the address of the
; Request Header in ES:BX, which it saves in a local
; variable and then returns to MS-DOS.
;---------------------------------------------------------------------------
STRATEGY        proc    far
                        ;save address of Request Header
                mov     word ptr cs:PtrSav,bx           ;offset
                mov     word ptr cs:PtrSav+2,es         ;segment
                ret             ;back to MS-DOS
STRATEGY        endp

                page

;---------------------------------------------------------------------------
; Driver RAM Data
;---------------------------------------------------------------------------
%WriteProtect   = 0                     ;error codes
%UnknownUnit    = 1
%DriveNotReady  = 2
%UnknownCommand = 3
%CRCerror       = 4
%BadStructure   = 5
%SeekError      = 6
%UnknownMedia   = 7
%UnknownSector  = 8
%OutOfPaper     = 9
%WriteFault     = 10
%ReadFault      = 11
%OtherFailure   = 12


PtrSav          dd      ?               ;double-word pointer to Request Header
                                        ;  passed to Strat routine by MS-DOS
EDISKsegment    dw      4400h           ;(44000-7FFFFh)
TestLocation    dw      6000h           ;memory test location = 6000:0h
FailureMsg      db      "Installation failed: No RAM at 44000-7FFFF (hex)"
                db      CR,LF,EOM
SuccessMsg      db      "240K RAM Disk defined at 44000-7FFFF (hex)"
                db      CR,LF,EOM
InstallMsg      db      CR,LF
                db      "HP110 RAM Disk Driver  "
                db      "Rev 1.0"               ;revision number
                db      CR,LF,EOM
                page
;---------------------------------------------------------------------------
; Device Parameters -- EQUates
;---------------------------------------------------------------------------
Secsiz          equ     200h            ;512 bytes per sector
Secpau          equ     1               ;1 sector per allocation Unit
Resvsec         equ     1               ;1 reserved sector (boot)
Nfats           equ     1               ;1 file allocation table
Ndire           equ     20h             ;32 root directory entries max
Nsec            equ     1E0h            ;480 sectors total (44000-7FFFF hex)
Mediads         equ     0FAh            ;media descriptor byte
Fatsiz          equ     2               ;2 sectors used by a FAT

;---------------------------------------------------------------------------
; BOOT Record
; This includes the BIOS Parameter Block (BPB) below
;---------------------------------------------------------------------------
BootR           db      0EBh,01Ch,090h  ;JMP instruction
                db      "EDISK110"
;---------------------------------------------------------------------------
; BIOS Parameter Block data structure
;---------------------------------------------------------------------------
BPB             dw      Secsiz          ;# bytes per sector
                db      Secpau          ;# sectors per allocation Unit
                dw      Resvsec         ;# reserved sectors
                db      Nfats           ;# file allocation tables
                dw      Ndire           ;max # root directory entries
                dw      Nsec            ;total # sectors
MediaDescriptor db      Mediads         ;media desctriptor byte
                dw      Fatsiz          ;# sectors used by a FAT

                dw      0h              ;# sectors/track
                dw      0h              ;# heads
                dw      0h              ;# hidden sectors
BootrLen        EQU     $-BootR         ;length of boot record
;---------------------------------------------------------------------------
; BPB pointer array
;---------------------------------------------------------------------------
BPBtable        dw      BPB             ;drive Unit 0

;---------------------------------------------------------------------------
; Device Driver "Interupt Routine"
;
; This entry point is called by MS-DOS immediately after
; the call to the "Strategy Routine". which saved the long
; address of the Request Header in the local variable "PtrSav".
;
; The "Interrupt Routine" uses the Command Code passed in
; the Request Header to Transfer to the appropriate device
; handling routine.  Each command code routine must place
; any necessary return information into the Request Header,
; then perform a Near Return with AX = Status.
;---------------------------------------------------------------------------
DRIVER          proc    far

                push    ax                      ;save CPU registers
                push    bx
                push    cx
                push    dx
                push    ds
                push    es
                push    di
                push    si
```

```
                push    bp

                lds     bx,cs:[PtrSav]          ;get pointer to I/O packet
                mov     al,byte ptr ds:[bx].CmdCode     ;get command code
                cmp     al,Max_Cmd              ;make sure it's legal
                ja      BadCommand              ;bad command

                cbw
                shl     ax,1                    ;command*2 = word table index
                mov     si,offset CmdTbl        ;point SI
                add     si,ax                   ;  at command vector
                mov     al,byte ptr ds:[bx].Unit        ;AL = unit code
                mov     ah,byte ptr ds:[bx].Media       ;AH = media descriptor
                mov     cx,word ptr ds:[bx].Count       ;CX = sector count
                mov     dx,word ptr ds:[bx].Start       ;DX = start sector
                les     di,dword ptr ds:[bx].Trans      ;ES:DI = transfer addr
                push    cs                              ;DS -> this
                pop     ds                              ;  code segment
                jmp     word ptr ds:[si]                ;go do command

;---------------------------------------------------------------------------
;       EXIT - All routines return through this path
;---------------------------------------------------------------------------
BadCommand:
                mov     al,%UnknownCommand              ;unknown command error
                jmp     short $ERROREXIT
$BUSYEXIT:
                mov     ah,03h
                jmp     short Exit1
$FAILEXIT:
                lds     bx,cs:[PtrSav]
                sub     word ptr ds:[bx].Count,cx       ;# of success I/Os
$ERROREXIT:
                mov     ah,081h         ;mark error return
                jmp     short Exit1
$EXIT:
                mov     ah,01h          ;mark good return
Exit1:
                lds     bx,cs:[PtrSav]
                mov     word ptr ds:[bx].Status,ax      ;mark oper complete
                pop     bp              ;restore general registers
                pop     si
                pop     di
                pop     es
                pop     ds
                pop     dx
                pop     cx
                pop     bx
                pop     ax
                ret                     ;back to MS-DOS
DRIVER          endp
                page

;---------------------------------------------------------------------------
; Command Code subroutines called by Interrupt Routine
;
; These routines are called with DS:BX pointing to the
; Request Header.
;
; They should return AX = 0 if function was completed
; successfully, or AX = 8000H + Error code if function failed.
;---------------------------------------------------------------------------

                ASSUME  DS:CODE

;---------------------------------------------------------------------------
;       MEDIA CHECK
;---------------------------------------------------------------------------
$MEDIACHECK:    xor     di,di                           ;0 = "don't know"
                lds     bx,[PtrSav]
                mov     word ptr ds:[bx].Trans,di       ;set Media change code
                jmp     $EXIT

;---------------------------------------------------------------------------
;       BUILD BPB
;---------------------------------------------------------------------------
$BUILDBPB:      mov     bl,al                           ;convert unit #
                xor     bh,bh                           ;  into BPB array index
                shl     bx,1                            ;  in BX
                mov     ax,word ptr BPBtable[bx]        ;AX points as BPB
                lds     bx,[PtrSav]
                mov     word ptr ds:[bx].BPBaddr,ax     ;offset
                mov     word ptr ds:[bx].BPBaddr+2,cs   ;segment
                jmp     $EXIT

;---------------------------------------------------------------------------
;       READ
;---------------------------------------------------------------------------
$READ:          jcxz    ReadReturn                      ;br if count=0

                call    RdWrSetUp                       ;set up for transfer
                jc      FAILEXIT                        ;br if setup failed

                call    PerformRead                     ;do the read
                jc      FAILEXIT                        ;br if read failed

ReadReturn:     jmp     $EXIT                           ;else return OK

;---------------------------------------------------------------------------
;       WRITE / WRITE with VERIFY
;---------------------------------------------------------------------------
$WRITE:         jcxz    WriteReturn                     ;br if count=0

                call    RdWrSetUp                       ;set up for transfer
                jb      FAILEXIT                        ;br if setup failed

                call    PerformWrite                    ;do the write
                jb      FAILEXIT                        ;br if write failed

WriteReturn:    jmp     $EXIT                           ;else return OK

FAILEXIT:       jmp     $FAILEXIT

;---------------------------------------------------------------------------
; RdWrSetup - Check read/write parameters before doing access
;
; Entry:        AL = Unit number
;               AH = Media descriptor
;               CX = Sector count
;               DX = First Sector
;            ES:DI = Transfer address
; Returns:      CY=1 Something's wrong, error code is in AL
;---------------------------------------------------------------------------
RdWrSetUp:      cld                             ;ensure SI/DI incrementing
                test    al,al                   ;is unit number 0?
                jz      UnitOK

                mov     al,%UnknownUnit         ;1=unknown Unit error
```

```
                jmp     short RdWrError

UnitOK:         cmp     ah,MediaDescriptor      ;check media descriptor
                je      MediaOK

                mov     al,%UnknownMedia        ;7=unknown medium error
                jmp     short RdWrError

MediaOK:        xor     bh,bh                   ;BX = drive #
                mov     bl,al
                shl     bx,1
                mov     bx,word ptr BPBtable[bx]        ;point BX at BPB
                mov     bx,word ptr [bx+08H]            ;BX=sectors/drive
                cmp     dx,bx                   ;DX within limit?
                jb      SectorOK                ;br if yes

                mov     al,%UnknownSector       ;8=sector not found error
                jmp     short RdWrError

SectorOK:       clc
                ret

RdWrError:      stc
                ret
                page

;---------------------------------------------------------------------------
; PerformRead - Read from RAM disk
;
; Entry:        CX = Number of sectors to read
;               DX = Starting sector number
;            ES:DI = Transfer area
;---------------------------------------------------------------------------
PerformRead:
ReadLoop:       call    RdEnableDX              ;point DS:SI at sector
                jc      ReadExit                ;br if not available

                push    cx                      ;save sector count
                mov     cx,Secsiz               ;transfer one entire sector
                rep     movsb                   ;fullbank read
                pop     cx                      ;restore sector count
                inc     dx                      ;bump sector number
                loop    ReadLoop                ;go read next sector
                clc                             ;CY=0 means no error

ReadExit:       ret

;---------------------------------------------------------------------------
; PerformWrite - Write to RAM disk
;
; Entry:        CX = Number of sectors to write
;               DX = Starting sector number
;            ES:DI = Transfer area
;---------------------------------------------------------------------------
PerformWrite:   push    es                      ;point DS:SI
                pop     ds                      ;  at the transfer area
                mov     si,di                   ;  (instead of ES:DI)

WriteLoop:      call    WrEnableDX              ;ES:DI -> sector DX for write
                jc      WriteExit               ;br if not available

                push    cx                      ;save sector count
                mov     cx,Secsiz               ;transfer one entire sector
                rep     movsb                   ;fullbank write
                pop     cx                      ;restore sector count
                inc     dx                      ;bump sector number
                loop    WriteLoop               ;go write next sector
                clc                             ;CY=0 means no error

WriteExit:      ret


;---------------------------------------------------------------------------
; WrEnableDX - Make available sector DX, and point ES:DI at it
;
; Entry:        DX = Sector number
; Returns:   ES:DI = Pointer to start of sector
;               CY=1 if error occurs (AL contains error code)
;---------------------------------------------------------------------------
WrEnableDX:     push    bx
                push    si
                push    ds
                call    RdEnableDX              ;make sector available
                jc      WrEnableExit            ;br if failed

                call    Writeable?              ;is it write protected?
                jc      WrEnableExit            ;br if yes

                push    ds                      ;point ES:DI
                pop     es                      ;  at sector
                mov     di,si                   ;  (instead of DS:SI)
                clc

WrEnableExit:   pop     ds
                pop     si
                pop     bx
                ret

;---------------------------------------------------------------------------
; RdEnableDX - Get pointer to start of RAMdisk sector in DS:SI
;              (Convert FB sector umber to physical address)
; Entry:        DX = Sector number
; Returns    DS:SI = Start of sector
; Destrotys:    Nothing
;---------------------------------------------------------------------------
RdEnableDX:     push    cx
                push    dx
                and     dx,1FFh                 ;max sector # =511
                mov     cl,5                    ;multiply sector # by 32
                shl     dx,cl                   ;  to get segment value
                add     dx,cs:EDISKsegment      ;add segment base (4400h)
                mov     ds,dx                   ;load DS
                xor     si,si                   ;SI=0   (offset)
                pop     dx
                pop     cx
                clc
                ret

                page

;---------------------------------------------------------------------------
; Writeable? - check if drive/sector is write protected
;
; Entry:     DS:SI = Ptr to physical start of sector
; Returns:      CY = 1 if sector is write protected (AL=0)
; Destroys:     Nothing
;---------------------------------------------------------------------------
Writeable?:     push    ax                      ;save AX
```

```
                mov     al,byte ptr ds:[si]     ;AL=data read from sector
                not     byte ptr ds:[si]        ;invert data
                mov     ah,byte ptr ds:[si]     ;AH=data read from sector
                not     byte ptr ds:[si]        ;invert data again
                xor     al,ah                   ;AL=0FFh if location writable
                inc     al                      ;Z=1 if writable
                pop     ax                      ;restore AX
                clc                             ;set CY=0
                jz      Writable                ;br if writable

                mov     al,%WriteProtect        ;else return AL=error code
                stc                             ;  and CY=1

Writable:       ret


;---------------------------------------------------------------------------
; This Initialization code for the driver is called only once, when the
; driver is loaded.  It returns the address of the BIOS Parameter Block
; pointer array.  Only MS-DOS services 01-0CH and 30H can be called by the
; Initialization function.
;
; Init returns its own address to the DOS as the start of free memory after
; the driver, so that the memory occupied by INIT will be reclaimed after
; it is finished with its work.
;---------------------------------------------------------------------------
$INIT:          mov     dx,offset InstallMsg    ;display driver
                mov     ah,9                    ;  installation message
                int     21h

                push    ds
                push    ax
                mov     ax,cs:TestLocation
                mov     ds,ax                   ;test memory location
                xor     si,si                   ;  (6000:0h)
                call    Writeable?              ;is it RAM?
                                                ;  (i.e. a modified NOMAD?)
                pop     ax
                pop     ds
                jnc     RAMfound

                mov     dx,offset FailureMsg    ;unmodified NOMAD
                jmp     MesgOut                 ;  cannot define EDISK

RAMfound:       inc     NumDrives               ;bump number defined drives
                push    ax                      ;save registers
                push    cx
                push    es
                push    ds
                push    si
                push    di

                mov     ax,cs:EDISKsegment
                mov     ds,ax                   ;test memory location
                mov     es,ax                   ;set DS to ES for stosb
                xor     si,si                   ;  4400:0h
                mov     ax,word ptr ds:[si]     ;fetch location
                cmp     ax,word ptr cs:BootR    ;is it a JMP opcode?
                je      EdiskOK                 ;yes, assume Edisk intact

;---------------------------------------------------------------------------
; The following code formats the Edisk.  This is done by zeroing out all the
; sectors, creating a valid boot sector (excluding disk boot code) and
; initializing the FAT and root directory sectors.  This code is called only
; once if the first byte of the boot sector does not contain a JMP instruction
; opcode (EB 1Ch), as is the case during initial system cold power-up (battery
; first connected) and possibly when the Edisk contents are corrupted.
;---------------------------------------------------------------------------
                cld                             ;ensure SI/DI incrementing
                mov     cx,Nsec                 ;get # sectors

ZeroLoop:       push    cx                      ;save sector count (for loop)
                xor     di,di                   ;set DI to start of sector
                mov     cx,Secsiz               ;set sector size (for rep)
                xor     al,al                   ;will be writing 0's
                rep     stosb                   ;write out 1 sector (ES:DI)
                pop     cx                      ;restore sector count
                mov     ax,es                   ;point ES to next sector
                add     ax,Secsiz/16            ;  (increment by 20h)
                mov     es,ax
                loop    Zeroloop                ;go write next sector

                push    ds
                pop     es                      ;set ES=DS (4400h)
                xor     di,di                   ;ES:DI is destination
                lea     si,cs:BootR             ;DS:SI is source
                push    cs
                pop     ds                      ;set DS=CS
                mov     cx,BootrLen             ;set length of boot record
                rep     movsb                   ;write boot record to Edisk

                mov     ax,es                   ;(ES still contains 4400h)
                add     ax,Secsiz/16
                mov     es,ax                   ;point ES at start of FAT
                xor     di,di                   ;  (4420:0h)
                mov     al,Mediads              ;get Media Descriptor byte
                stosb                           ;write out to FAT
                xor     ax,ax
                dec     ax                      ;AX=FFFFh
                stosw                           ;write to FAT (2 reserved
                                                ;  12-bit entries)

EdiskOK:        pop     di                      ;restore registers
                pop     si
                pop     ds
                pop     es
                pop     cx
                pop     ax

                mov     dx,offset SuccessMsg    ;installation succseeful

MesgOut:        mov     ah,9h                   ;output message to screen
                int     21h

InstallDone:    lds     bx,cs:[PtrSav]          ;return pointer to INIT start
                mov     word ptr [bx].Trans,offset $INIT
                mov     word ptr [bx].Trans+2,cs

$INITEXIT:      lds     bx,cs:[PtrSav]
                mov     al,cs:NumDrives
                mov     byte ptr [bx].Media,al
                mov     word ptr [bx].BPBaddr,offset BPBtable
                mov     word ptr [bx].BPBaddr+2,cs
                jmp     $EXIT

CODE            ENDS
                end
```

```
Received: by hpsrla; Mon, 27 Apr 87 20:03:37 pdt
Date: Mon, 27 Apr 87 14:50:10 pdt
From: Samuel Chau <hpsrla!hpccc!samc>

                name    NOMADtst
                page    55,132
                title   'HP-110 Expanded RAM Memory Test'

CR              equ     0Dh                     ;carriage return
LF              equ     0Ah                     ;line feed
EOM             equ     '$'                     ;end of message signal
NumSectors      equ     1E0h                    ;480 sectors total
SecSiz          equ     200h                    ;512 bytes/sector
EDISKsegment    equ     7FE0h                   ;start at 7FE00h

CSEG            SEGMENT PARA PUBLIC 'CODE'
                ORG     100h
                ASSUME  CS:CSEG,DS:CSEG,ES:CSEG,SS:CSEG

RAMtest         proc    near

                mov     dx,offset RunMsg
                mov     ah,9
                int     21h
                mov     cx,NumSectors           ;CX:=# sectors to test
                mov     ax,EDISKsegment         ;DS:=segment of first sector

NextSect:       mov     ds,ax                   ;  into DS
                push    cx                      ;save sector count
                mov     cx,SecSiz
                call    TestRAMsect
                pop     cx                      ;restore sector count
                jc      BadSect
                sub     ax,SecSiz/16            ;bump DS to next sector
                loop    NextSect                ;go test it

                mov     dx,offset TestPassMsg   ;RAM test passed
                jmp     short MesgOut

BadSect:        mov     dx,offset TestFailMsg   ;RAM test failed
MesgOut:        push    cs
                pop     ds                      ;retore DS for int 21h
                mov     ah,9
                int     21h
                mov     ax,4C00h                ;return to MS-DOS
                int     21h

RAMtest         endp

RunMsg          db      CR,LF
                db      "HP-110 Extra RAM Memory Test (44000-7FFFF hex)"
                db      CR,LF
                db      "Wait (10 sec) ..."
                db      EOM
TestPassMsg     db      CR
                db      "Test passed: RAM OK"
                db      CR,LF,EOM
TestFailMsg     db      CR
                db      "Test failed: RAM BAD (or missing)"
                db      CR,LF,EOM

;---------------------------------------------------------------------------
; TestRAMsect - Non-destructive RAM sector test
;
; Entry:        CX = sector size
;              DS:0 = first RAM location to test
; Returns:      CY=0 if all locations in bank tested OK
;               CY=1 if bad RAM location found within bank
;                   DS:SI is the failed location
; Destroys:     Nothing
;---------------------------------------------------------------------------
TestRAMsect:    push    ax                      ;save registers
                push    bx
                push    cx
                push    dx
                push    si

                mov     dh,055h                 ;checkerboard pattern
                mov     dl,0aah                 ;alternate pattern
                xor     si,si                   ;initialize SI

TestLoop:                                       ;checkerboard test
                mov     bl,byte ptr ds:[si]     ;save original data
                mov     byte ptr ds:[si],dl     ;write test pattern 1
                mov     al,byte ptr ds:[si]     ;read it back into AL
                mov     byte ptr ds:[si],dh     ;write test pattern 2
                mov     ah,byte ptr ds:[si]     ;read it back into AH
                mov     byte ptr ds:[si],bl     ;restore original data

                cmp     al,dl                   ;pattern 1 OK?
                jnz     BadLocation             ;no, must be bad
                cmp     ah,dh                   ;pattern 2 OK?
                jnz     BadLocation             ;no, must be bad
                                                ;otherwise, continue
                                                ;complement test
                mov     al,byte ptr ds:[si]     ;original data into AL
                not     byte ptr ds:[si]        ;invert RAM data
                mov     ah,byte ptr ds:[si]     ;flipped data into AH
                not     byte ptr ds:[si]        ;restore RAM data
                xor     al,ah                   ;complement check
                inc     al                      ;pass? (is it 0?)
                jnz     BadLocation             ;no, must be bad

                inc     si                      ;bump SI
                loop    TestLoop                ;  to test next byte

                clc                             ;all locations tested OK
                jmp     short Exit

BadLocation:    mov     byte ptr ds:[si],bl     ;restore location again,
                                                ;  just to be safe
                stc                             ;CY=1 means bad RAM

Exit:           pop     si                      ;restore registers
                pop     dx
                pop     cx
                pop     bx
                pop     ax
                ret

CSEG            ENDs
                end     RAMtest
```

LOCATION OBJECT CODE LINE SOURCE LINE

in EPROM (≠ IO Address) for I/O Address multiply by 2, add 8000

default setting = 96K

```
                    26                  ORG 40H
0040 0018           27 RAM_Limit        DW      1800H           ; paragraph size of system ram
0042 20             28                  DB      32              ; High Water Mark  } serial buffer
0043 08             29                  DB       8              ; Lo Water Mark    }
0044 0080           30 LCD_Screen       DW      8000H           ; seg start of LCD Mem
0046 FF17           31 LCD_ScreenSize   DW      17FFH           ; size of lcd memory (not inc font)
0048 8081           32 LCD_FontSpace    DW      8180H           ; seg start of LCD font space
004A 0008           33 LCD_FontSize     DW      0800H           ; num of bytes in font
004C 2F00           34 LCD_SMemSize     DW      02FH            ; number of lines in screen memory
004E FFFF           35                  DW      0FFFFH          ; DUMMY
0050 E043           36 RAM_CKS          DW      43E0H           ; seg end of ramdisk
                    37
                    38 ; configuration options
                    39 HAS_MODEM        EQU     02H
                    40 HAS_RS232        EQU     04H
0052 0600           41 ConfigInfo       DW      HAS_MODEM+HAS_RS232
                    42
                    43 ; Sytem Power
0054 1316           44 RUN_PWR          DB 013H,016H            ; Operating Current 80 ma
0056 1316           45 WAIT_PWR         DB 013H,016H            ; Wait Current      80 ma
0058 0015           46 SLEEP_PWR        DB 000H,015H            ; Sleep Current    350 ua
005A 0262           47 MODEM_PWR        DB 002H,062H            ; Modem Current     10 ma
005C 406C           48 CHARGE_PWR       DB 040H,06CH            ; Charger Current  270 ma
005E 0728           49 RS_232_PWR       DB 007H,028H            ; RS-232 Current    30 ma
                    50
```

→ 7FE0 for 512K (never tested)

for estimate of battery %

↑ ~ 16.4 mA/COUNT

will case close with the extra chips?

has never been tested with different RAM size