

# Need More System Storage? Embed a Hitachi Microdrive!

**HITACHI**  
Inspire the Next

## Dr. William F. Heybruck

Senior Storage Engineer

bill.heybruck@hitachigst.com

### Introduction

Embedded processor solutions often have need of large amounts of data buffering or program storage which must be accommodated by the host system. DRAMs for caching and some type of non-volatile memory, including EPROM, Flash and RAM, for boot records are viable but costly options. The introduction of the Hitachi Microdrive® in 1999 made possible the usage of hard disk drive technology to minimize the dependence on silicon-based solutions. With a small amount of additional logic and some simple subroutines, hard disk drives can easily be used for caching, boot records, and in some system designs, even customer data.

The Hitachi Microdrive is a CompactFlash® + Type II device available in various capacities up to 6 Gigabytes in a package the size of a common matchbook. It requires a relatively small amount of power to operate, and can be optimized to further enhance its power utilization in host systems. The size and power consumption of the Microdrive make it an ideal solution for many embedded systems. It can be attached to the processor board and replaced on an as needed basis, or used to both serve as a memory extension to support host processor functions and as the data storage receptacle for large amounts of end-user data.

### Theory of operation

The Hitachi Microdrive processes data in 512 byte sectors and can be accessed in one of three different modes: Memory, I/O or IDE. In IDE Mode, data can be accessed using either DMA or UDMA (mode 2) protocol at speeds up to 33MB/sec. Logical Block Addressing (LBA Mode) is supported to free the programmer from drive configuration concerns such as the number of cylinders, heads, and sectors. This is a departure from the old DOS days, where sectors were addressed by Cylinder, Head and Sector (or CHS). Today, Logical Block Addressing maps the drive into a sequential range of sectors. With a 4GB Microdrive, for example, LBA Mode has the sectors mapped in a contiguous orientation using address numbers from 0 through 7A1000h. On a 6GB Microdrive, the address numbers range from 0 through 16E3A5800h.

During a typical data processing operation, the host identifies the desired sector address it wishes to access. It sends this sector address to a set of registers on the drive. A Read or Write command is then issued by the host to the drive command register. If a Read command were issued, the drive obtains the data from the sector address as set by the host and proceeds to read the corresponding 512 byte sector into its buffer. Once this operation is complete, the drive sets the Data Request (DRQ) status bit which informs the host that the data is ready for retrieval.

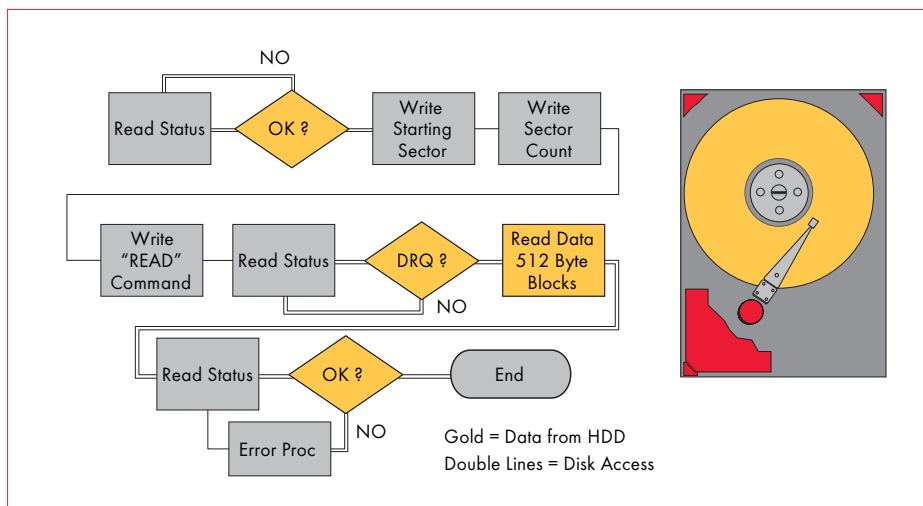
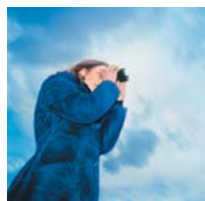


Figure 1. Processing Read Data Command Sequence (non-DMA) on a HDD

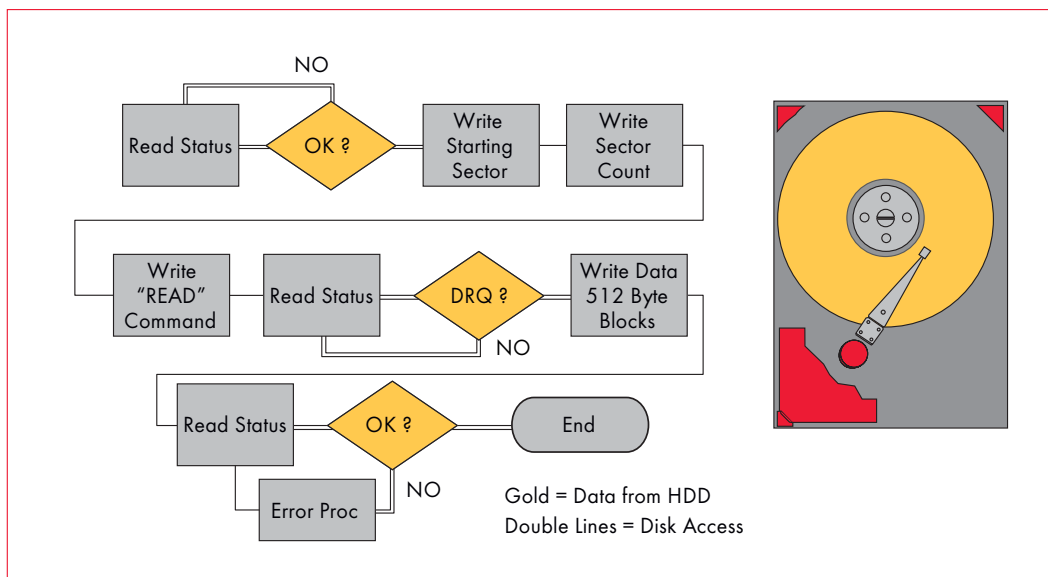


Figure 2. Processing Write Data Command Sequence (non-DMA) on a HDD

If the host were to issue a Write command, the drive would set the DRQ bit indicating it is ready to receive data. The host then writes 512 bytes of data to the drive which places it in the buffer. When a full sector is received, DRQ will be cleared and the drive will take the data from its buffer and write it to the sector as identified by the host. When done, the drive will clear the BSY status bit. Details regarding status bits settings are provided in the Compact Flash + (CF+) specification.

The operating mode must be selected for the Microdrive to properly interface with an embedded processor. For the simplest hardware and software interaction, either Memory Mode or IDE Mode are suggested. Both are default settings. In Memory Mode, access to the CIS data is available. In either Memory or IDE Mode, a block of addresses are reserved for the hard disk and an address decoder is used to generate the Card Enable (CE) lines of the interface. The registers are accessed using standard memory Reads and Writes in Memory Mode, or I/O Reads and Writes in IDE Mode. In either mode, the actual address space can be determined by the host designer. In the event that a processor is used that supports a separate I/O space, the customary IDE address space is 1F0-1F7h (CS0) and 3F6-3F7h (CS1), but this is not a requirement for many ad hoc embedded designs. It is important to note that the active state of the RESET line changes depending on Memory-I/O or IDE Mode. In IDE Mode, RESET is low active.

### Operating mode selection and device setting

To select Memory Mode, the Output Enable (OE) and Reset signals must be inactive—set high and low, respec-

tively—at Power On. In order to select IDE Mode, OE must be active (or low) and RESET should be high. IDE Mode can also be selected by holding OE low and cycling power. In either case, IDE Mode can only be selected using Power On Reset. To activate the DMA protocol in IDE mode, connect the processor DMARQ to -INPACK (pin 43) and -DMACK to -REG (pin 44).

In IDE Mode, the level of CSEL determines if the drive will act as a Master or Slave device. If CSEL is low, the drive responds to Master designated commands. Commands are directed to the Master or Slave via the setting of the DRIVE bit located in the head/drive register. UDMA mode functionality is discussed in a separate White Paper and is available through the Hitachi Global Storage Technologies web site.

Once the desired operating mode has been selected, status of the designated data processing operation can be read from the status registers (locations) on the drive. In the Memory Mode example described herein, CS0 is activated with an address of CXXXh, CS1 is activated with DXXXh, and REG is activated with an access to X8XXh. To access the CIS, read accesses to addresses C8XXh will return CIS values. A Read from C007h, reads the Status Register from the drive slot. A write to C007h writes the data into the Command Register. In order to guard against data errors, all parameters must be pre-written into the appropriate registers before the command is written.

To Read a sector of data, the host sends the desired sector address to registers C003h through C006h on the drive. The number of sectors requested is placed in C002h. After this is complete, the Read Sector command

is finally sent from the host and written to the C007h register. The BSY status bit is then set and the DRQ or ERR bits are monitored while the Read operation is executed. If an error occurs and the Read cannot complete, the ERR bit will be activated on the drive. If the Read is good, the DRQ bit is activated and the host can begin reading the data in 512 byte increments (or 256 word reads) from address C000h for each of the sectors requested. Write operations are handled in the same manner, with the exceptions of issuing a Write command and the direction of the data flow after DRQ status has been received by the host.

The file format of the data on the 6GB and 4GB drives is DOS FAT32. Microdrives with 1GB and 2GB capacities are formatted and shipped supporting the FAT16 format. [Details of the DOS FAT16 and FAT32 format are discussed elsewhere (e.g. the Introduction to FAT 16/32 Files Systems White Paper and the DOS technical reference are provided the end of this document).] The subroutines devoted to the handling of the FAT and directory are substantial. For many embedded applications, any code and data can be written to and read from the drive with any format determined by the programmer. However, the use of anything other than a standard format eliminates portability and requires reformatting of the drive before using that drive in a system using standard operating system file routines.

### A CF+ interface for the 8051

A CF+ interface prototype to support all three operating modes (Memory, I/O, and IDE) was designed and built upon the introduction of the first Hitachi Microdrive (Figure 3). The CF+ interface is geared to test and verify the required hardware and software necessary for external systems to successfully communicate with the Compact-Flash Microdrive. A Hi Tech Equipment Single Board 8031 Computer was used to evaluate the connection. Appendix A is the software listing for the routines written to verify the logical interface of the Microdrive.

### Memory Access Mode

The CF+ interface for the 8031 consists of two devices: the 20L8 PAL as shown in Figure 1 and the 74LS373 address latch. The 20L8 PAL is programmed as an address decoder and performs other rudimentary logic tasks such as combining two Card Detect signals into one signal fed to the main processor, buffering the Read and Write processor signals into the OE and WR signals, and converting the A11 address bit to the REG line used in Memory Mode. The address latch simply holds the low order addresses while the AD bus switches to data.

A data bus width of 8 or 16 bits is possible in all modes. Memory Mode defaults to an 8 bit width. Selection of the 16 bit width can be accomplished using a combination of the CE1, CE2 and A0 address lines. Details regarding this can be found in the CF II + Specification.

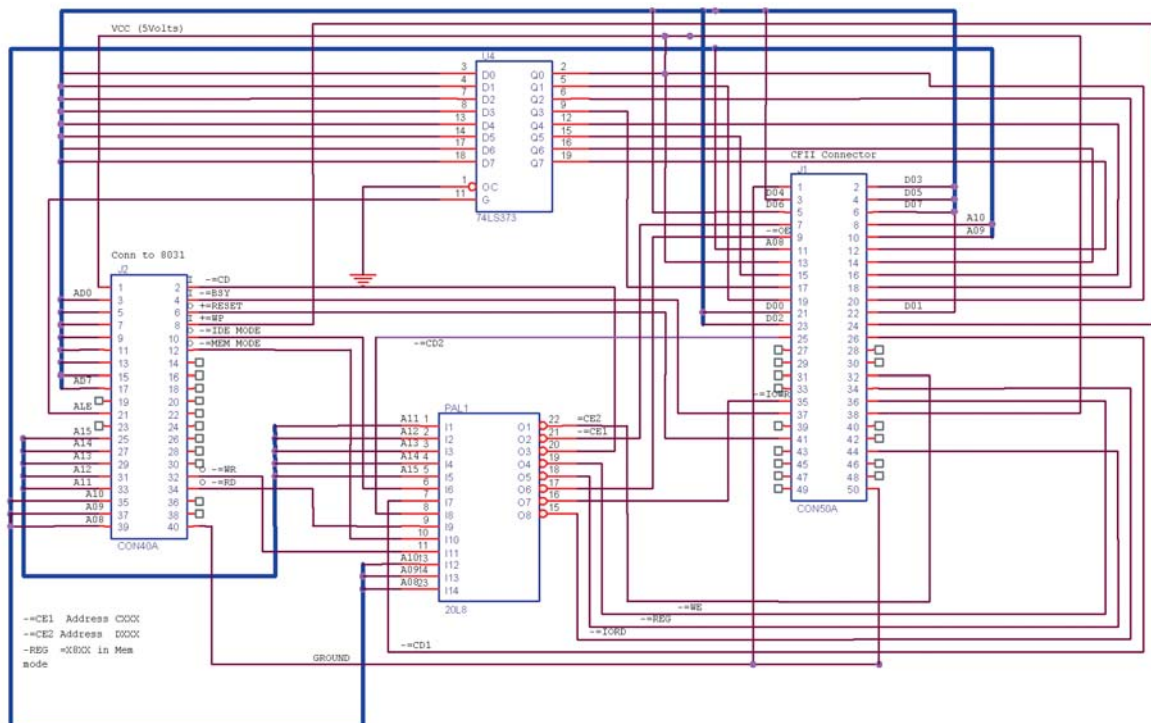


Figure 3. CFII Memory Mode Interface Schematic for 8051

In applications where hot pluggability of the hard drive is required, a data bus transceiver should be employed in the host design. This will help to guard against disturbance of normal processor operations from unwanted transitions during device removal and insertion. Hot pluggability is discussed in the White Paper entitled "Controlling a CF Socket" which is available through the Hitachi Global Storage Technologies web site.

## IDE Mode

In IDE Mode, the hardware set up is similar to Memory Mode with some minor but important differences. The Address Latch functions are the same except that only address lines A0-A2 can be routed to the drive interface. The other address lines (A3-A10) are connected to ground. It is recommended that ATASEL also be grounded by the host as well. The signal called OE in Memory and I/O Mode is used in IDE Mode to instruct the drive that IDE Mode has been selected. The REG signal is only used for DMACK should DMA transfers be desired; if DMACK is not used, REG should be tied high. INPACK is used as DMARQ when DMA transfers are desired.

The drive designation as Master or Slave is set by CSEL. CSEL must be grounded for the drive to operate as a Master or tied high (open) to respond as a Slave. The setting of bit 4 in the CDH register (location X6 hex) is used to differentiate the drive Master or Slave designation. If bit 4 is zero, the destination drive is a Master; if the bit 4 register indicates a 1, the destination drive is a Slave. The Power On default for this bit is zero. In order to communicate with the drive as a Slave, a 1 should be written into bit 4 of the CDH register followed by a status register Read to location X7 hex.

Other IDE Mode set-up differences include the RESET signal setting as low active as compared to high active setting in both Memory and I/O Modes. The PAL is programmed to decode the address and to set the CS0 and CS1 signals. It also routes the processor Read and Write signals to IORD and IOWR, respectively. As in Memory Mode, the PAL still combines the two Card Detect signals into one for the processor to poll.

Data bus widths of either 8 or 16 bits are possible in IDE Mode. The default setting is 16 bits as this setting provides higher data performance. If the 8 bit data bus width is desired, 01h should be written in the Features Register and the Set Features Command issued to the drive.

## I/O Mode

I/O Mode combines settings used in both Memory and IDE Modes. Much of the same address decode of Memory Mode is utilized since an attribute configuration register

must be first modified to place the drive in I/O Mode. This is achieved using Reads and Writes to the configuration register space (the same as Memory Mode), however the CE1 and REG address lines are active.

Once the drive is placed in I/O Mode, access to the command and parameter registers is gained with REG active (low). The CE1 and CE2

settings depend on whether the data bus width is set to transfer either 8 or 16 bits. The factory default setting is 8 bits, where the CE1 signal is active (or low). Selection of the 16 bit data bus width can be accomplished using a combination of CE1, CE2 and A0. This process is detailed in the CF II + specification.

## Host software development pointers

After the access mode has been programmed (Memory, I/O or IDE Mode), the developer can begin to write sub-routines to retrieve data stored on the drive. As covered in the Theory of Operation section, hard drives, like Flash devices, process the data one sector at time. One sector is comprised of 512 bytes. The process of reading or writing to a Hitachi 4GB or 6GB Microdrive consists of specifying the sector to access, issuing the proper command, then reading or writing the data from/to the drive buffer. The software functions covered here involve basic subroutines to read and write sectors on the disk drive, as well as to perform some error handling.

The reading and writing of files in DOS format to the drive requires considerably more software, but the effort may be worthwhile if the programmer wishes to update or download data for a PC system. This requires in depth knowledge on the structure of the Boot Record, File Allocation Tables and the Directory, as the interaction across these structures permit the reading and writing of files with DOS recognizable names. The DOS Technical Reference provides detailed file structure information necessary to perform file access with a FAT 16 file system. Several other documents provide similar information for FAT32 and NTFS file systems.

One of the first tasks the host software must perform is to identify which device is attached. The Identify Device command can be used for this purpose, and will also return valuable information for error prevention and detection. The maximum value of the LBA which is used to determine the size of the FAT and directory areas are indicated by Words 60 and 61.

If there is no desire for portability of the drive to other dissimilar devices, the programmer can determine how data can be read from and written to the hard drive. The programmer must first determine the method by which used or available sectors are identified. For many embed-

ded applications, the freedom afforded by this option is the easiest method for software development, and results in a somewhat proprietary data storage solution.

As mentioned previously, the Memory and IDE Modes are the simplest to implement. In Memory Mode, the configuration registers are accessed as memory locations and the command/data registers as different memory locations. Drive configuration and command execution therefore consists of writing the appropriate parameters to the designated memory locations on the drive, and then sending the execute command request.

In I/O Mode, the host must first access the configuration registers via Memory Mode to place the drive in I/O Mode. All subsequent accesses from this point, other than reconfiguration, are handled by the drive using I/O reads and writes to the sector address specified by the host.

## Summary

System capabilities can be significantly increased once a CF+ Type II device such as the Hitachi Microdrive is embedded on the host processor board. The Microdrive's miniaturization of data storage and large capacity make it useful as a storage device for exclusive access by the host system or by both the host and the system end user. The Microdrive is designed to ensure that minimal hardware integration effort is required. This paper addresses the

fundamental programming parameters necessary in the development of the host software to successfully interface with the Microdrive. The innovation begins with the developer who can utilize the Hitachi Microdrive to design a highly versatile system based on an effective embedded or removable storage solution. The Microdrive's one-inch form factor and ever increasing capacities offer a viable platform from which next generation innovations can arise.

## References:

- 1) Hitachi OEM Hard Disk Drive Specifications for Hitachi Microdrive with CF+ Type II Interface, Rev 1.0
- 2) Compact Flash Association Draft Compact Flash Specification, Rev 1.4e
- 3) An Introduction to FAT 16/FAT 32 File Systems White Paper by Dr. William F. Heybruck, Hitachi Global Storage Technologies
- 4) UDMA Mode Functionality White Paper by Dr. William F. Heybruck, Hitachi Global Storage Technologies
- 5) IBM DOS Technical Reference 5.0 S52G-9932
- 6) HiTech Equipment Corporation, San Diego, C A, [www.hte.com](http://www.hte.com)
- 7) Controlling a CF Slot WhitePaper by Dr. William F. Heybruck, Hitachi Global Storage Technologies

## Appendix A

### 8051 Assembler Routine for CFII Hitachi Global Storage Technologies Microdrive

```

; 8051 Code to access Compact Flash Card
;
; Change History
; 8/4/99 WFH ADD DIAGNOSTIC SUBROUTINE (RUN_DIAGS), ADD ERROR MESSAGES TO PROCESS ERROR
; 8/5/99 WFH Add Main menu, Mode Menu, Diag menu and associated routines.
; 8/9/99 WFH DEBUG DISPLAY FUNCTIONS
; 8/10/99 WFH ADD COMMAND MENU and Chk pwr, id device, erase sector commands
; 8/11/99 WFH Add idle, re-calibrate, seek, SLEEP MODE, SET FEATURE commands
; 8/16/99 WFH Updates for I/O mode and Base Addresses
; 8/17/99 wfh UPDATE FOR ERROR DETECTION
; 8/18/99 WFH UPDATE FOR READ/WRITE TO AN ADDRESS
; 9/13/99 WFH Update for IDE mode (FORMAT ID DRIVE RESPONSE)
; 10/13/99 WFH FIX ID FORMAT PROBLEMS
; 9/13/01 WFH Sub for ID drive byte swapping
; Copyright 1999, 2001 IBM Corporation
; Equates
;
$MOD51
$NOPAGING
CF_BASE_ADDR EQU 0C000H ;Base address decode for Compact Flash socket -CE1 Activation
CE2_BASE_ADDR EQU 0D000H ;Base address decode for -CE2 activation
REG_BASE_ADDR EQU 0C800H ;Base address decode for -CE1 and -REG
CFG_MFG_CODE EQU 0C822H ;ADDRESS OF CONFIG MFG CODE TPL
CFG_ROT_DEV EQU 0C868H ;ADDRESS OF CONFIG ROTATING DEVICE CODE
CFG_BASE_TPL EQU 0C874H ;Address of Config Base Address TPL (CE2, REG, addr 74)
STATUS_REG EQU 0C007H ;Address of Status Register
COMMAND_REG EQU 0C007H ;Address of Command Register
DH_REG EQU 0C006H ;Address of Device/Head register
SECTOR_COUNT EQU 0C002H ;Address of Sector Count Register for CFLASH
ERROR_REG EQU 0C001H ;Address of Error Register
FEATURE_REG EQU 0C001H ;ADDRESS OF FEATURES REGISTER
CONTROL_REGISTER EQU 0C00EH ;ADDRESS OF DEVICE CONTROL REGISTER FOR SOFT RESET
CARD_DETECT_BIT EQU 01H ;Port 1 bit 0 -=CD Card detect bits lOrdí FROM CARD
BSY_BIT EQU 02H ;Port 1 bit 1 -BSY for Memory Mode From CARD
WRITE_PROTECT EQU 03H ;Port 1 bit 2 +WP from CARD
RESET_BIT EQU 04H ;Port 1 bit 3 +=RESET to CARD
WAIT_BIT EQU 05H ;Port 1 bit 4 -=Wait sig from CARD
P1_INIT EQU 0DBH ;Port 1 initialization setting for MEMORY MODE
Sendstr EQU 03F09H ;Sends a string to the display DP points to string
GET_CHAR EQU 03F03H ;GETS A CHARACTER FROM THE KEYBOARD
PUT_CHAR EQU 03F06H ;Sends a character to the display
A_1 EQU ë1í
A_2 EQU ë2í
A_3 EQU ë3í
A_4 EQU ë4í
A_5 EQU ë5í
A_6 EQU ë6í
A_7 EQU ë7í
A_8 EQU ë8í
A_9 EQU ë9í
A_A EQU ëAí
A_N EQU ëNí
A_CR EQU 00DH
A_LF EQU 00AH; Port Bit Designations
; Port 0 Port 1

```

```

;      Bit 0          ==Card Detect (INPUT)
;      Bit 1          ==Busy      (INPUT)
;      Bit 2          +=Reset (MEM)  ==RESET IDE (OUTPUT)
;      Bit 3          +=Write Protect (INPUT)
;      Bit 4          ==IDE MODE (OUTPUT)
;      Bit 5          ==MEMORY MODE (OUTPUT)
;      Bit 6
;      Bit 7
;
;
; MAIN LINE      READ/WRITE SECTORS from COMPACT FLASH/ MICRODRIVE
;
;
;      ORG      4000H
START:          MOV      A,#P1_INIT
;              MOV      P1,A
;              CALL     INIT_DRIVE          ;CHECK FOR MICRODRIVE
DISP_MAIN:      MOV      DPTR,#MAIN_MENU
;              CALL     SEND_MSG
;              CALL     GET_RESP
;              CJNE    A,#A_1,MAIN_1
;              JMP      DISP_DIAG
MAIN_1:         CJNE    A,#A_2,MAIN_2
;              JMP      DISP_MODE
MAIN_2:         CJNE    A,#A_3,MAIN_3
;              JMP      READ_A_SECTOR
MAIN_3:         CJNE    A,#A_4,MAIN_4
;              JMP      WRITE_A_SECTOR
MAIN_4:         CJNE    A,#A_5,MAIN_5
;              JMP      DISP_BUF_DATA
MAIN_5:         CJNE    A,#A_6,MAIN_6
;              JMP      DISP_DRV_CMDS
MAIN_6:         CJNE    A,#A_7,MAIN_7
;              JMP      0          ;GO BACK TO MONITOR
MAIN_7:         MOV      DPTR,#ERROR8_MSG
;              CALL     SEND_MSG
;              JMP      DISP_MAIN
;
DISP_MODE:      MOV      DPTR,#MODE_MENU
;              CALL     DISP_MENU
;              CALL     GET_RESP
;              CJNE    A,#A_4,DISP_MODE1
;              JMP      DISP_MAIN
DISP_MODE1:     CJNE    A,#A_1,DISP_MODE2
;              CALL     SET_MEM_MODE
;              JMP      DISP_MODE
DISP_MODE2:     CJNE    A,#A_2,DISP_MODE3
;              CALL     SET_IO_MODE
;              JMP      DISP_MODE
DISP_MODE3:     CJNE    A,#A_3,DISP_MODE
;              CALL     SET_IDE_MODE
;              JMP      DISP_MODE
;
DISP_DIAG:      MOV      DPTR,#DIAG_MENU
;              CALL     DISP_MENU
;              CALL     GET_RESP
;              CJNE    A,#A_6,DISP_DIAG1
;              JMP      DISP_MAIN
DISP_DIAG1:     CJNE    A,#A_1,DISP_DIAG2
;              CALL     RUN_DIAGS          JMP      DISP_DIAG
DISP_DIAG2:     CJNE    A,#A_2,DISP_DIAG3
;              CALL     SOFT_RESET
;              JMP      DISP_DIAG
DISP_DIAG3:     CJNE    A,#A_3,DISP_DIAG4
;              CALL     HARD_RESET
;              JMP      DISP_DIAG

```

```

DISP_DIAG4:      CJNE    A, #A_4, DISP_DIAG5
                  CALL    READ_ADDR
                  JMP     DISP_DIAG
DISP_DIAG5:      CJNE    A, #A_5, DISP_DIAG
                  CALL    WRITE_ADDR
                  JMP     DISP_DIAG

READ_A_SECTOR:   CALL    GET_LBA_ADDRESS
                  CALL    READ_SECTOR
                  JMP     DISP_MAIN
WRITE_A_SECTOR:  CALL    GET_LBA_ADDRESS
                  CALL    WRITE_SECTOR
                  JMP     DISP_MAIN
DISP_BUF_DATA:   CALL    DISP_BUFFER
                  JMP     DISP_MAIN
;
; DRIVE COMMAND MENU HANDLING
;
DISP_DRV_CMDS:   MOV     DPTR, #CMD_MENU
                  CALL    DISP_MENU
                  CALL    GET_RESP
                  CJNE    A, #A_9, DRV_CMD_1
                  JMP     DISP_MAIN
DRV_CMD_1:       CJNE    A, #A_1, DRV_CMD_2
                  CALL    CHECK_PWR
                  JMP     DISP_DRV_CMDS
DRV_CMD_2:       CJNE    A, #A_2, DRV_CMD_3
                  CALL    ID_DEVICE
                  JMP     DISP_DRV_CMDS
DRV_CMD_3:       CJNE    A, #A_3, DRV_CMD_4
                  CALL    ERASE_SECTOR
                  JMP     DISP_DRV_CMDS
DRV_CMD_4:       CJNE    A, #A_4, DRV_CMD_5
                  CALL    IDLE_CMD
                  JMP     DISP_DRV_CMDS
DRV_CMD_5:       CJNE    A, #A_5, DRV_CMD_6
                  CALL    RECALIBRATE_CMD
                  JMP     DISP_DRV_CMDS
DRV_CMD_6:       CJNE    A, #A_6, DRV_CMD_7
                  CALL    SEEK_CMD
                  JMP     DISP_DRV_CMDS
DRV_CMD_7:       CJNE    A, #A_7, DRV_CMD_8
                  CALL    SLEEP_MODE
                  JMP     DISP_DRV_CMDS
DRV_CMD_8:       CJNE    A, #A_8, DISP_DRV_CMDS
                  JMP     MORE_CMDS
;
MORE_CMDS:       MOV     DPTR, #MORE_CMD_MENU
                  CALL    DISP_MENU
                  CALL    GET_RESP
                  CJNE    A, #A_9, MORE_CMD_1
                  JMP     DISP_DRV_CMDS
MORE_CMD_1:      CJNE    A, #A_1, MORE_CMDS
                  CALL    SET_FEATURES
;
;
HERE:            JMP     HERE
; LOOP FOR END OF ROUTINE
;
; DISP_MENU
; DISPLAYS THE MENU POINTED TO BY DPTR THEN ADDS THE ENTER NUMBER LINE
;
DISP_MENU:       CALL    SEND_MSG
                  MOV     DPTR, #ENT_NUM_MSG
                  CALL    SEND_MSG
                  RET
;
; GET_RESP
; GET KEYBOARD DATA UP TO CR. PUT IN KBD BUFFER, LAST VALID CHAR IN ACCUM
;

```

```

GET_RESP:      MOV     DPTR,#KBUFF
               MOV     R2,#80
GET_RESP1:    CALL    GET_CHAR
               MOV     R3,A
               CALL    PUT_CHAR
               MOV     A,R3
               CJNE   A,#0DH,GET_RESP2
               MOV     A,#0AH      ;LOAD A LINE FEED
               CALL    PUT_CHAR    ;send it to keep input on the display
               MOVX   A,@DPTR
               RET
GET_RESP2:    INC     DPTR
               MOVX   @DPTR,A
;
               INC     DPTR
               DJNZ   R2,GET_RESP1
               MOV     DPTR,#KBD_BUFF_MSG
               CALL    SEND_MSG
               MOV     A,#0
               RET
KBD_BUFF_MSG: DB     ðKEYBOARD BUFFER OVERFLOW (MORE THAN 80 CHARS)í
               DW     0D0AH
               DB     ð$í
;
; HARD_RESET  SEND A HARDWARE RESET TO THE CFII SLOT  (FLIP BIT 4)
;
HARD_RESET:   MOV     A,P1
               ORL    A,#04H ;T/ON RESET BIT
               MOV     P1,A
               ANL    A,#0FBH      ;T/OFF RESET BIT
               MOV     P1,A
               RET
;
;CHECK_CARD   CHECKS FOR THE CARD DETECT BIT TO BE LOW (CARD IN PLACE)
;             IF NOT SENDS MSG TO SCREEN
;
;
CHECK_CARD:   MOV     A,P1
               JB     ACC.0,CHK_CRD0
               MOV     A,#0
               RET
CHK_CRD0:    MOV     DPTR,#ERROR0_MSG      ;Send CARD NOT PRESENT MESSAGE
               CALL    SEND_MSG
               MOV     A,0FH
               RET
;
;SET_MEM_MODE SETÍS CARD INTO MEMORY MODE
;
SET_MEM_MODE: CALL    CHECK_CARD          ;VERIFY A CARD IS INSERTED          JNZ     SET_MEM_1
               MOV     A,P1
               ANL    A,#0DFH      ;SET MEM LINE LOW
               MOV     P1,A
               CALL    WAIT_RDY_STAT
               MOV     DPTR,#REG_BASE_ADDR
               MOV     A,#00
               MOVX   @DPTR,A          ;SET MEM CONFIG IN BASE ADDRESS
SET_MEM_1:   RET
;
;SET_IO_MODE  SETÍS CARD INTO IO MODE
;
SET_IO_MODE: CALL    CHECK_CARD          ;VERIFY A CARD IS INSERTED
               JNZ     SET_IO_1
               MOV     DPTR,#REG_BASE_ADDR
               MOV     A,#01          ;Set I/O mode 01
               MOVX   @DPTR,A
               MOV     A,P1
               ORL    A,#020H      ;SET MEM/IO LINE HIGH
               MOV     P1,A
    
```

```

SET_IO_1:          RET
;
SET_IDE_MODE: CALL CHECK_CARD          ;VERIFY A CARD IS INSERTED
                JNZ   SET_IDE_1
                MOV   A,#0C0H          ;LOAD PORT FOR IDE MODE AND RESET
                MOV   P1,A
                ORL   A,#04H          ;T/OFF RESET (LOW ACTIVE)
                MOV   P1,A
                CALL  WAIT_RDY_STAT
                MOV   DPTR,#FEATURE_REG ;PREPARE TO SET FEATURES FOR 8 BIT TRANSFERS
                MOV   A,#01H
                MOVX  @DPTR,A
                MOV   DPTR,#COMMAND_REG
                MOV   A,#0EFH          ;LOAD SET FEATURES COMMAND
                MOVX  @DPTR,A
                CALL  WAIT_RDY_STAT
                RET
SET_IDE_1:        MOV   DPTR,#SET_IDE_BAD_MSG
                CALL  SEND_MSG
                RET
SET_IDE_BAD_MSG: DB    ěCARD NOT DETECTED, IDE MODE NOT SET $ĭ
;
;
;INIT_DATA      Initialize data variables in RAM
;
INIT_DATA:       MOV   DPTR,#SECTOR_CNT
                MOV   A,#01H
                MOVX  @DPTR,A
                INC   DPTR
                MOV   A,#00H
                MOVX  @DPTR,A
                INC   DPTR
                MOV   A,#03H
                MOVX  @DPTR,A
                INC   DPTR
                MOV   A,#00H
                MOVX  @DPTR,A
                MOV   DPTR,#DATA_BUFFER ;LOAD DATA BUFFER with FF,FE,.....
                MOV   R2,#1
                MOV   R1,#0
INIT_LOOP:       MOV   A,R1
INIT_LOOP1:     MOVX  @DPTR,A
                INC   DPTR
                DJNZ  R1,INIT_LOOP1
                DJNZ  R2,INIT_LOOP
                RET
;
; Soft_RESET
;   Send soft reset to flash card (flip bit 2 address E)
SOFT_RESET:     MOV   DPTR,#CONTROL_REGISTER
                MOV   A,#0CH          ;Set the S/W reset bit
                MOVX  @DPTR,A
                MOV   A,#08H          ;Clear the reset bit
                MOVX  @DPTR,A
                RET
;
;
;
EXIT:           JMP   EXIT
;
; CHECK_PWR
CHECK_PWR:      MOV   A,#0            ;SET UP FOR DRIVE 0
                MOV   DPTR,#DH_REG    ;LOAD ADDRESS OF DH REGISTER
                MOVX  @DPTR,A        ;WRITE A 0 THERE
                MOV   A,#98H          ;LOAD THE CHECK POWER COMMAND
                INC   DPTR
    
```

```

MOVX @DPTR,A ;WRITE TO THE COMMAND REGISTER
CALL WAIT_RDY_STAT ;WAIT FOR READY STATUS
JNZ CHECK_ERR
MOV DPTR,#SECTOR_COUNT
MOVX A,@DPTR ;READ THE SECTOR COUNT
CJNE A,#00,CHECK_PWR1 ;CHECK FOR STANDY OR SLEEP MODE
MOV DPTR,#CHECK_PWR_MSG0 ; LOAD THE MESSAGE sTANDYOR SLEEP MODE
DETECTED
CALL SEND_MSG
RET
CHECK_PWR1:
CJNE A,#0FFH,CHECK_PWR2 ;CHECK FOR IDLE MODE
MOV DPTR,#CHECK_PWR_MSG1 ;LOAD THE MESSAGE idle MODE DETECTED
CALL SEND_MSG
RET
CHECK_PWR2:
MOV DPTR,#CHECK_PWR_MSG2 ;LOAD THE MESSAGE UNKNOWN RESPONSE
CALL SEND_MSG
RET
CHECK_ERR:
CHECK_PWR_MSG0: DB ëSTANDBY OR SLEEP MODE DETECTED $1
CHECK_PWR_MSG1: DB ëIDLE MODE DETECTED $1
CHECK_PWR_MSG2: DB ëUNKNOWN RESPONSE FROM CHECK POWER COMMAND $1
;
;
;ID_DEVICE SEND IDENTIFY DEVICE COMMAND, GET RESULTS IN DATA BUFFER
;
ID_DEVICE:
MOV A,#0 ;SET UP FOR DRIVE 0
MOV DPTR,#DH_REG ;LOAD ADDRESS OF DH REGISTER
MOVX @DPTR,A ;WRITE A 0 THERE
MOV A,#0ECH ;LOAD THE ID DEVICE COMMAND
INC DPTR
MOVX @DPTR,A ;WRITE TO THE COMMAND REGISTER
CALL WAIT_DRQ_STATUS ;WAIT FOR READY STATUS
JNZ ID_ERR
CALL READ_DATA
CALL FORMAT_ID ;FORMAT THE DATA FOR DISPLAY
MOV DPTR,#ID_DRIVE_MSG CALL SEND_MSG ;DISPLAY THE
DATA
ID_ERR: RET
;
;ERASE_SECTOR SENDS THE ERASE SECTOR COMMAND
;
ERASE_SECTOR: CALL GET_LBA_ADDRESS ;GET THE ADDRESS OF THE SECTOR TO BE ERASED
CALL SETUP_SECTOR_ADD ;MOVE TO THE RIGHT REGISTERS
JNZ ERASE_ERR
MOV A,#0C0H ;LOAD THE COMMAND
MOV DPTR,#COMMAND_REG
MOVX @DPTR,A ;WRITE THE COMMAND
CALL WAIT_RDY_STAT
JNZ ERASE_ERR
MOV DPTR,#ERASE_DONE_MSG
CALL SEND_MSG
ERASE_ERR: RET
ERASE_DONE_MSG: DB ëERASE COMMAND COMPLETE $1
;
; IDLE_CMD SEND THE IDLE COMMAND TO THE DRIVE
;
IDLE_CMD:
MOV DPTR,#SECTOR_COUNT
MOV A,#01H ;SET AUTO POWER DOWN TO 5 SECS
MOVX @DPTR,A
MOV DPTR,#DH_REG ;SET THE DRIVE/HEAD TO 0
MOV A,#0
MOVX @DPTR,A
INC DPTR ;POINT TO COMMAND REGISGER
MOV A,#97H ;SET IDLE COMMAND
MOVX @DPTR,A
CALL WAIT_RDY_STAT
JNZ IDLE_ERR

```

```

                MOV     DPTR,#IDLE_DONE_MSG
                CALL    SEND_MSG
IDLE_ERR:
IDLE_DONE_MSG: DB     òIDLE COMMAND COMPLETE $í
;
;RECALIBRATE_CMD          SEND THE RECALIBRATE COMMAND
;
RECALIBRATE_CMD: MOV     A,#0E0H           ;SET THE DATA FOR THE DH REG
                MOV     DPTR,#DH_REG
                MOVX    @DPTR,A
                INC     DPTR
                MOV     A,#10H           ;LOAD THE RECALIBRATE COMMAND
                MOVX    @DPTR,A
                CALL    WAIT_RDY_STAT
                JNZ     RECAL_ERR
                MOV     DPTR,#RECAL_DONE_MSG
                CALL    SEND_MSG
RECAL_ERR:
RECAL_DONE_MSG: DB     òRE-CALIBRATE COMPLETE $í
;
;SEEK_CMD                SEND THE SEEK COMMAND
;
SEEK_CMD:        CALL    GET_LBA_ADDRESS
                CALL    SETUP_SECTOR_ADD ;SEND TO DEVICE REGISTERS
                JNZ     SEEK_ERR
                MOV     DPTR,#COMMAND_REG
                MOV     A,#071H         ;SET SEEK COMMAND
                MOVX    @DPTR,A         ;SEND IT
                CALL    WAIT_RDY_STAT
                JNZ     SEEK_ERR             MOV     DPTR,#SEEK_DONE_MSG
                CALL    SEND_MSG
SEEK_ERR:
SEEK_DONE_MSG:  DB     òSEEK COMPLETE $í
;
;SLEEP_MODE             PUT THE DRIVE IN SLEEP MODE
;
SLEEP_MODE:     MOV     A,#00H         ;SET THE DATA FOR THE DH REG
                MOV     DPTR,#DH_REG
                MOVX    @DPTR,A
                INC     DPTR
                MOV     A,#99H         ;LOAD THE SLEEP MODE COMMAND
                MOVX    @DPTR,A
                CALL    WAIT_RDY_STAT
                JNZ     SLEEP_ERR
                MOV     DPTR,#SLEEP_DONE_MSG
                CALL    SEND_MSG
SLEEP_ERR:
SLEEP_DONE_MSG: DB     òSLEEP MODE COMPLETE $í
;
;SET_FEATURES          GET THE FEATURE FROM OPERATOR, PUT IN FEATURES REG AND SEND COMMAND
;
SET_FEATURES:   MOV     DPTR,#REQ_FEATURE_MSG
                CALL    SEND_MSG
                CALL    GET_RESP
                MOV     DPTR,#KBUFF
                MOVX    A,@DPTR         ;GET FIRST CHARACTER
                CALL    CONV_TO_HEX
                RL      A               ;MOVE TO HIGH NIBBLE
                RL      A
                RL      A
                RL      A
                MOV     R2,A           ;STORE IN R2
                MOV     DPTR,#KBUFF+1 ;GET SECOND BYTE
                MOVX    A,@DPTR
                CALL    CONV_TO_HEX
                ORL     A,R2           ;OR IT WITH HIGH DIBBLE
    
```

```

MOV     DPTR,#FEATURE_REG
MOVX   @DPTR,A
MOV     DPTR,#DH_REG
MOV     A,#0
MOVX   @DPTR,A
INC     DPTR
MOV     A,#0EFH           ;LOAD SET FEATURES COMMAND
MOVX   @DPTR,A
CALL   WAIT_RDY_STAT
MOV     DPTR,#SET_FEAT_DONE
CALL   SEND_MSG
RET

REQ_FEATURE_MSG:  DB     ðENTER FEATURE IN HEX (2 CHARACTERS) E.G. 01 = 8 BIT DATA TRANSFER $1
SET_FEAT_DONE:   DB     ðSET FEATURES COMMAND COMPLETE $1
;
;INIT_DRIVE  Routine to CHECK FOR MICRODRIVE, AND  SETUP BASE ADDRESS
;
;
INIT_DRIVE:      MOV     DPTR,#CFG_MFG_CODE   ;LOAD ADDRESS OF MFG CODE
MOVX   A,@DPTR           ;GET THE CODE
CJNE   A,#0A4H,NOT_MICRODRIVE
INC     DPTR              ;MOVE TO NEXT BYTE (2 IN CFG AREA)
INC     DPTR
MOVX   A,@DPTR           CJNE   A,#00,NOT_MICRODRIVE
MOV     DPTR,#CFG_ROT_DEV ;CHECK FOR ROTATING IBM DEVICE
MOVX   A,@DPTR
CJNE   A,#08H,NOT_MICRODRIVE
MOV     DPTR,#CFG_BASE_TPL ;Load address of CIS Base Address TPL
MOVX   A,@DPTR           ;Get the base address (should be 0200)
MOV     R1,A             ;Temp Store in R1 (first the 00)
INC     DPTR
INC     DPTR
MOVX   A,@DPTR
MOV     R2,A             ;Temp store in R2 ( now the 02)
MOV     DPTR,#REG_BASE_ADDR ;Add the base address to the CF address (was C800)
MOV     A,DPL
ADD     A,R1
JNC    NINCDPL
INC     DPH              ;Bump DPH if carry in previous add
NINCDPL:         MOV     DPL,A
MOV     R1,A
MOV     A,DPH
ADD     A,R2
MOV     DPH,A
MOV     R2,A
MOV     DPTR,#CONFIG_REG_BASE ;Load address of storage space
MOV     A,R1
MOVX   @DPTR,A
INC     DPTR
MOV     A,R2
MOVX   @DPTR,A           ;Save the address at location CONFIG_REG_BASE
MOV     A,#00           ;LOAD NORMAL STATUS
RET

NOT_MICRODRIVE: MOV     DPTR,#NOT_MICRO_MSG
CALL   SEND_MSG
MOV     DPTR,#CONFIG_REG_BASE
MOV     A,#00H
MOVX   @DPTR,A
MOV     A,#0CAH
MOVX   @DPTR,A
MOV     A,#0FFH
RET

NOT_MICRO_MSG:  DB     ðNOT A MICRODRIVE, LOADED CA00 $1
;
; LOAD_CFG_ADDR ROUTINE TO LOAD DPTR WITH THE BASE REGISTER ADDRESS.
;

```

```

LOAD_CFG_ADDR:      MOV     DPTR,#CONFIG_REG_BASE
                   MOVX    A,@DPTR
                   MOV     R1,A
                   INC     DPTR
                   MOVX    A,@DPTR
                   MOV     DPH,A
                   MOV     DPL,R1
                   RET

;
;READ_STAT Routine to read the status byte of the CF drive
;
READ_STAT:         MOV     DPTR,#STATUS_REG      ;Load address of status Register
                   MOVX    A,@DPTR              ;Load the status
                   RET

;
;WAIT_RDY_STAT      Routine to wait for READY status, FLAG errors
;                   RETURNS WITH A=0 FOR NO ERROR OR ERROR CODE IN A
;WAIT_RDY_STAT:    CALL    READ_STAT             ;Read the Status
                   JB     ACC.0,ERROR1          ;ERROR BIT SET
                   ANL    A,#0F0H             ;Mask the 2 ready bits
                   CJNE   A,#50H,WAIT_RDY_STAT ;Wait for a 50h without error
                   MOV     A,#00H             ;Load a ZERO for return status
                   RET

ERROR1:           CALL    PROCESS_ERROR
                   RET

;
PROCESS_ERROR:     MOV     DPTR,#ERROR_REG      ;Load address of ERROR REGISTER
                   MOVX    A,@DPTR              ;Load the error register
                   MOV     R1,A
                   ANL    A,#0F0H             ;MASK OUT HIGH SIDE
                   RR     A
                   RR     A                   ;SHIFT TO LOW SIDE
                   RR     A
                   RR     A
                   ORL    A,#30H              ;OR 30 TO MAKE A NUMBER
                   MOV     DPTR,#ERROR2_CODE
                   MOVX    @DPTR,A             ;SAVE THE NUMBER IN THE MESSAGE AREA
                   MOV     A,R1
                   ANL    A,#0FH             ;CONVERT THE LOW SIDE
                   ORL    A,#30H
                   INC    DPTR
                   MOVX    @DPTR,A             ;SAVE IT IN THE MESSAGE AREA
                   MOV     DPTR,#ERROR2_MSG
                   CALL    SEND_MSG
                   MOV     A,R1               ;RESTORE THE ERROR CODE
                   JB     ACC.0,ERROR3
                   JB     ACC.2,ERROR4
                   JB     ACC.4,ERROR5
                   JB     ACC.6,ERROR6
                   JB     ACC.7,ERROR7

CLEAR_ERROR:      CALL    SOFT_RESET           ;CLEAR THE ERRORS
                   MOV     A,R1               ;restore the error code
                   RET

ERROR3:           MOV     DPTR,#ERROR3_MSG
                   CALL    SEND_MSG
                   JMP    CLEAR_ERROR

ERROR4:           MOV     DPTR,#ERROR4_MSG
                   CALL    SEND_MSG
                   JMP    CLEAR_ERROR

ERROR5:           MOV     DPTR,#ERROR5_MSG
                   CALL    SEND_MSG
                   JMP    CLEAR_ERROR

ERROR6:           MOV     DPTR,#ERROR6_MSG
                   CALL    SEND_MSG
                   JMP    CLEAR_ERROR

ERROR7:           MOV     DPTR,#ERROR7_MSG

```

```

        CALL    SEND_MSG
        MOV     DPTR,#CRLF
        JMP    CLEAR_ERROR
;
;SEND_MSG      Send a message and follow with CR LF
;
SEND_MSG:      CALL    SENDSTR
               MOV     DPTR,#CRLF
               CALL    SENDSTR
               RET
;
; RUN_DIAGS    Run the diagnostics on the flash card and flag any errors to display.;
RUN_DIAGS:     CALL    WAIT_RDY_STAT      ;Wait for Ready Status
               JNZ     RUN_D_ERROR        ;IF ERROR BYPASS ROUTINE
               MOV     DPTR,#COMMAND_REG
               MOV     A,#090H           ;LOAD THE DIAGNOSTIC COMMAND
               MOVX    @DPTR,A           ;SEND IT TO THE DEVICE
RUN_DIAGS1:    CALL    READ_STAT
               ANL     A,#0F0H           ;MASK OUT READY AND BUSY BITS FROM STATUS
               CJNE    A,#050H,RUN_DIAGS1 ;WAIT FOR A READY STATUS
               CALL    READ_STAT
               JB      ACC.0,RUN_DIAGS_ERROR ;CHECK FOR ERROR STATUS
               MOV     DPTR,#DIAGS_MSG0  ;SEND DIAGS COMPLETE MESSAGE
               CALL    SEND_MSG
RUN_D_ERROR:   RET
RUN_DIAGS_ERROR: MOV    DPTR,#ERROR_REG
               MOVX    A,@DPTR
               CJNE    A,#01H,RUN_DIAGSE1
               MOV     DPTR,#DIAGS_MSG1
               CALL    SEND_MSG
               RET
RUN_DIAGSE1:   CJNE    A,#02H,RUN_DIAGSE2
               MOV     DPTR,#DIAGS_MSG2
               CALL    SEND_MSG
               RET
RUN_DIAGSE2:   CJNE    A,#03H,RUN_DIAGSE3
               MOV     DPTR,#DIAGS_MSG3
               CALL    SEND_MSG
               RET
RUN_DIAGSE3:   CJNE    A,#04H,RUN_DIAGSE4
               MOV     DPTR,#DIAGS_MSG4
               CALL    SEND_MSG
               RET
RUN_DIAGSE4:   CJNE    A,#05H,RUN_DIAGSE5
               MOV     DPTR,#DIAGS_MSG5
               CALL    SEND_MSG
               RET
RUN_DIAGSE5:   ANL     A,#080H           ;MASK IDE ERROR
               CJNE    A,#080H,RUN_DIAGSE6
               MOV     DPTR,#DIAGS_MSG6
               CALL    SEND_MSG
               RET
RUN_DIAGSE6:   MOV     DPTR,#DIAGS_MSG7
               CALL    SEND_MSG
               RET
;GET_ADDR      GETS A 4 CHARACTER ADDRESS AND CONVERTS TO HEX
;
GET_ADDR:      MOV     DPTR,#GET_ADDR_MSG
               CALL    SEND_MSG
               CALL    GET_RESP
               MOV     DPTR,#KBUFF+1    ;SET POINTER TO FIRST CHARACTER
               MOVX    A,@DPTR
               INC     DPTR
               CJNE    A,#0DH,GET_ADDR1
               JMP     GET_ADDR_ERR
    
```

```

GET_ADDR1:      CALL    CONV_TO_HEX
                RL     A
                RL     A
                RL     A
                RL     A
                MOV    R3,A
                MOVX   A,@DPTR                INC    DPTR
                CJNE  A,#0DH,GET_ADDR2
                JMP    GET_ADDR_ERR
GET_ADDR2:      CALL    CONV_TO_HEX
                ORL    A,R3
                MOV    R3,A
                MOVX   A,@DPTR
                INC    DPTR
                CJNE  A,#0DH,GET_ADDR3
                JMP    GET_ADDR_ERR
GET_ADDR3:      CALL    CONV_TO_HEX
                RL     A
                RL     A
                RL     A
                RL     A
                MOV    R4,A
                MOVX   A,@DPTR
                INC    DPTR
                CJNE  A,#0DH,GET_ADDR4
                JMP    GET_ADDR_ERR
GET_ADDR4:      CALL    CONV_TO_HEX
                ORL    A,R4
                MOV    R4,A
                MOV    A,#00
                RET
GET_ADDR_ERR:  MOV    DPTR,#GET_ADDR_ER_MSG
                CALL   SEND_MSG
                MOV    A,#0FFH
                RET
GET_ADDR_ER_MSG: DB    ðERROR IN ADDRESS, CR RECEIVED TOO SOON $1
GET_ADDR_MSG:  DB    ðENTER ADDRESS, 4 CHARACTERS IN HEX $1
;
;GET_DATA
;
GET_DATA:      MOV    DPTR,#GET_DATA_MSG
                CALL   SEND_MSG
                CALL   GET_RESP
                MOV    DPTR,#KBUFF+1          ;SET POINTER TO FIRST CHARACTER
                MOVX   A,@DPTR
                INC    DPTR
                CJNE  A,#0DH,GET_DATA1
                JMP    GET_DATA_ERR
GET_DATA1:     CALL    CONV_TO_HEX
                RL     A
                RL     A
                RL     A
                RL     A
                MOV    R5,A
                MOVX   A,@DPTR
                INC    DPTR
                CJNE  A,#0DH,GET_DATA2
                JMP    GET_DATA_ERR
GET_DATA2:     CALL    CONV_TO_HEX
                ORL    A,R5
                MOV    R5,A
                RET
GET_DATA_ERR:  MOV    DPTR,#GET_DATA_ER_MSG
                CALL   SEND_MSG
                JMP    GET_DATA
    
```

```

GET_DATA_MSG: DB      ðENTER DATA (2 HEX CHARACTERS $1
GET_DATA_ER_MSG: DB      ðDATA ERROR TRY AGAIN $1;
;WRITE ADDR
;
WRITE_ADDR:      CALL    GET_ADDR
                 MOV     DPTR,#WRITE_ADR_STO
                 MOV     A,R3
                 MOVX   @DPTR,A
                 INC    DPTR
                 MOV     A,R4
                 MOVX   @DPTR,A
                 CALL   GET_DATA
                 MOV     DPTR,#WRITE_ADR_STO
                 MOVX   A,@DPTR
                 MOV     R3,A
                 INC    DPTR
                 MOVX   A,@DPTR
                 MOV     R4,A
                 MOV     DPL,R4
                 MOV     DPH,R3
                 MOV     A,R5
                 MOVX   @DPTR,A
                 RET
WRITE_ADR_STO:   DW      0
;
;READ_ADDR      READS CONTENTS OF AN ADDRESS
;
READ_ADDR:      CALL    GET_ADDR
                 MOV     DPH,R3
                 MOV     DPL,R4
                 MOVX   A,@DPTR
                 CALL   DISP_HEX
                 RET
;
;READ CFLASH SECTOR , 512 BYTES FROM LBA ADDRESS IN MEMORY;
READ_SECTOR:    CALL    SETUP_SECTOR_ADD
                 JNZ     READ_SECT_ERROR
                 MOV     DPTR,#COMMAND_REG      ;LOAD ADDRESS OF COMMAND REGISTER (C007)
                 MOV     A,#20H                 ;LOAD A îREADî COMMAND
                 MOVX   @DPTR,A
                 CALL   WAIT_DRQ_STATUS         ;WAIT FOR DATA REQUEST FROM CARD INDICATING DATA IS
READY TO READ
                 JNZ     READ_SECT_ERROR         ;BYPASS READ ON ERROR
                 CALL   READ_DATA               ;READ THE DATA FROM THE DEVICE
READ_SECT_ERROR: RET
;
WAIT_DRQ_STATUS: CALL    READ_STAT
                 JB     ACC.0,WDRQS_ERROR      ;IF AN ERROR, GO PROCESS IT
WAIT_DRQ_LOOP:  MOVX   A,@DPTR                 ;RELOAD STATUS
                 ANL    A,#0F8H                ;MASK APPROPRIATE BITS (BSY,RDY DWF,DSC,DRQ,X,X,X)
                 CJNE  A,#58H,WAIT_DRQ_LOOP    ;LOOK FOR RDY,DSC,DRQ
                 MOV     A,#00                 ;GOOD STATUS RETURN
                 RET
WDRQS_ERROR:    CALL    PROCESS_ERROR
                 RET
;
;
; READ_DATA     READ 512 BYTES FROM THE DEVICE
;
READ_DATA:      MOV     DPTR,#DATA_BUFFER
                 CALL   READ_256               ;READ 256 BYTES FROM THE DEVICE
                 CALL   READ_256               RET
;
READ_256:       MOV     R3,#0                 ;RESET BYTE COUNT
READ_256_LOOP:  PUSH    DPH
                 PUSH    DPL

```

```

MOV     DPTR,#CF_BASE_ADDR
MOVX   A,@DPTR           ;GET A BYTE OF DATA
POP    DPL
POP    DPH
MOVX   @DPTR,A           ;SAVE IT IN THE BUFFER AREA
INC    DPTR
DJNZ   R3,READ_256_LOOP
RET

;
;SETUP_SECTOR_ADDRESS   MOVE VALUES FROM GET_SECTOR ADDRESS AREA TO DEVICE REGISTERS
;
SETUP_SECTOR_ADD:      CALL    WAIT_RDY_STAT           ;WAIT FOR READY STATUS
JNZ    SET_SEC_ERR     ;BYPASS ON ERROR
MOV    DPTR,#SECTOR_CNT
MOVX   A,@DPTR
MOV    R4,A
MOV    DPTR,#SECTOR_LBA_ADDRESS
MOVX   A,@DPTR
MOV    R7,A
INC    DPTR
MOVX   A,@DPTR
MOV    R6,A
INC    DPTR
MOVX   A,@DPTR
MOV    R5,A
MOV    DPTR,#SECTOR_COUNT
MOV    A,R4           ;LOAD SECTOR COUNT
MOVX   @DPTR,A       ;WRITE IT TO CFLASH REGISTER
INC    DPTR
MOV    A,R5           ;LOAD LBA 0-7
MOVX   @DPTR,A       ;WRITE IT TO REGISTER
INC    DPTR
MOV    A,R6           ;LOAD LBA 8-15
MOVX   @DPTR,A
INC    DPTR
MOV    A,R7           ;LOAD LBA 16-23
MOVX   @DPTR,A
INC    DPTR
MOV    A,#0E0H       ;SET THE LBA BIT
MOVX   @DPTR,A
MOV    A,#00         ;CLEAR A FOR GOOD STATUS
SET_SEC_ERR:         RET
;
;
;WRITE_SECTOR
WRITE_SECTOR: CALL  SETUP_SECTOR_ADD
MOV    DPTR,#COMMAND_REG ;LOAD ADDRESS OF COMMAND REGISTER (C007)
MOV    A,#30H         ;LOAD A îWRITE SECTORî COMMAND
MOVX   @DPTR,A
CALL   WAIT_DRQ_STATUS ;WAIT FOR DATA REQUEST FROM CARD INDICATING DATA IS
READY TO READ
JNZ    WRITE_ERROR
CALL   WRITE_DATA     ;READ THE DATA FROM THE DEVICE
WRITE_ERROR:         RET
;
;WRITE_DATA          WRITES 512 BYTES TO THE DRIVE;
WRITE_DATA:         MOV    DPTR,#DATA_BUFFER
CALL   WRITE_256      ;WRITE 256 BYTES TO THE DEVICE
CALL   WRITE_256
RET
;
WRITE_256:         MOV    R3,#0           ;RESET BYTE COUNT
WRITE_256_LOOP:    MOVX   A,@DPTR       ;GET THE DATA BYTE FROM THE BUFFER
INC    DPTR
PUSH   DPL
PUSH   DPH

```

```

MOV     DPTR,#CF_BASE_ADDR
MOVX   @DPTR,A           ;WRITE THE DATA BYTE TO THE DEVICE
POP     DPH
POP     DPL
DJNZ   R3,WRITE_256_LOOP
RET

;
; DISPLAY CHARACTER
;
DISP_CHAR:      MOV     R5,A           ;Save the character
                PUSH   DPH
                PUSH   DPL
                MOV     DPTR,#CHAR_MSG+18
                MOVX   @DPTR,A
                MOV     DPTR,#CHAR_MSG
                CALL   SEND_MSG
                POP    DPL
                POP    DPH
                MOV     A,R5
                RET
CHAR_MSG:      DB     ¨The Character is:  $1
;
;DISP_HEX
;
DISP_HEX:      MOV     R5,A
                PUSH   DPH
                PUSH   DPL
                MOV     DPTR,#HEX_MSG_CHAR
                RR     A
                RR     A
                RR     A
                RR     A
                ANL   A,#0FH
                CALL   HEX_CHAR
                MOVX   @DPTR,A
                INC    DPTR
                MOV     A,R5
                ANL   A,#0FH
                CALL   HEX_CHAR
                MOVX   @DPTR,A
                MOV     DPTR,#HEX_MSG
                CALL   SEND_MSG
                POP    DPL
                POP    DPH
                MOV     A,R5
                RET
HEX_MSG:      DB     ¨THE CHARACTER IN HEX IS:  ¨
HEX_MSG_CHAR: DW     0
                DB     ¨$1
HEX_CHAR:      ADD     A,#01           MOVC   A,@A+PC
                RET
                DB     ¨0123456789ABCDEF1
;
;GET LBA ADDRESS
;
GET_LBA_ADDRESS:  MOV     DPTR,#LBA_ADDR_MSG           ;DISPLAY GET LBA ADDRESS MESSAGE
                CALL   SEND_MSG
                CALL   GET_RESP                   ;GET THE RESPONSE
                MOV     DPTR,#KBUFF+1             ;ADDRESS OF RESPONSE
                MOV     R3,#3                     ;NUMBER OF BYTES (6 ASCII CHARACTERRS)
GET_LBA4:      MOVX   A,@DPTR                   ;GET CHARACTER
                INC    DPTR
                CJNE   A,#0DH,GET_LBA1           ;IF CR, SEND ERROR MESSAGE
                JMP    GET_LBA2
GET_LBA1:      CALL   CONV_TO_HEX                ;CONVERT FIRST ASCII CHARACTER TO HEX
                RL     A                          ;MOVE TO HIGH NIBBLE
    
```

```

        RL      A
        RL      A
        RL      A
        MOV     R5,A
        MOVX    A,@DPTR          ;GET NEXT ASCII CHAR
        INC     DPTR
        CJNE    A,#0DH,GET_LBA3  ;IF CR SEND ERROR MESSAGE
        JMP     GET_LBA2
GET_LBA3: CALL     CONV_TO_HEX          ;CONVERT FROM ASCII TO HEX (LOW NIBBLE)
        ORL     A,R5              ;OR IT WITH HIGH NIBBLE
        PUSH    DPL
        PUSH    DPH
        MOV     R5,A
        MOV     A,#3
        SUBB    A,R3
        MOV     DPTR,#SECTOR_LBA_ADDRESS ;PUT IT IN THE SECTOR ADDRESS AREA
        ADD     A,DPL
        MOV     DPL,A
        MOV     A,R5
        MOVX    @DPTR,A
        POP     DPH
        POP     DPL
        DJNZ    R3,GET_LBA4      ;CONTINUE FOR ALL 3 BYTES
        RET
GET_LBA2: MOV     DPTR,#LBA_ERR_MSG
        CALL    SEND_MSG
        JMP     GET_LBA_ADDRESS
LBA_ERR_MSG: DB     ðEND OF LINE DETECTED BEFORE 6 VALID CHARACTERS $1
LBA_ADDR_MSG: DB  ðENTER LBA SECTOR ADDRESS IN HEX 6 CHARACTERS, E.G. 0004DC $1
;
;CONV_TO_HEX CONVERTS CHARACTER IN ACC TO LOWORDER
;
CONV_TO_HEX: MOV     R2,A
        ANL     A,#0F0H
        CJNE    A,#30H,CONV_TXT
        MOV     A,R2
        ANL     A,#0FH
        JMP     CONV_TO_HEX0
CONV_TXT:  CJNE    A,#40H,CONV_ERR
        MOV     A,R2
        ANL     A,#0FH
        ADD     A,#09H
CONV_TO_HEX0: RETCONV_ERR:    PUSH    DPL
        PUSH    DPH
        MOV     DPTR,#CONV_ERR_MSG
        CALL    SEND_MSG
        POP     DPH
        POP     DPL
        MOV     A,#0
        JMP     CONV_TO_HEX0
CONV_ERR_MSG: DB  ðINVALID HEX DIGIT ENTERED$1
;
;DISP_BUFFER ROUTINE TO DISPLAY THE I/O BUFFER OF ADDRESS 5000-52FF (512 BYTES)
;
DISP_BUFFER: MOV     DPTR,#DATA_BUFFER
DISP_BUFFER5: MOV     R2,#16          ;LOAD LINE COUNT
DISP_BUFFER2: MOV     R3,#16          ;LOAD CHARACTER COUNT PER LINE
        MOV     A,DPH
        MOV     R4,A
        MOV     R6,A          ;SAVE IN R6 FOR ASCII ROUTINE
        MOV     A,DPL
        MOV     R5,A
        MOV     R7,A          ;SAVE IN R7 FOR ASCII ROUTINE
        MOV     DPTR,#DISP_BUF_MSG
        MOV     A,R4
        CALL    MOV_A_TO_BUFF    ;PUT HIGH ADDRESS INTO MESSAGE
    
```

```

MOV     A,R5
CALL    MOV_A_TO_BUFF      ;PUT LOW ADDRESS INTO MESSAGE
INC     DPTR                ;MOVE OVER COLON
INC     DPTR                ;MOVE OVER SPACE AFTER COLON
DISP_BUFF1:
PUSH    DPL
PUSH    DPH
MOV     DPH,R4              ;SET DPTR TO POINT TO DATA BUFFER
MOV     DPL,R5
MOVX   A,@DPTR             ;LOAD THE CHARACTER FROM THE BUFFER
INC     DPTR
MOV     R4,DPH              ;SAVE FOR LATER USE
MOV     R5,DPL
POP     DPH                 ;RESTORE POINTER TO MESSAGE AREA
POP     DPL
CALL    MOV_A_TO_BUFF      ;PUT THE 2 CHAR IN THE MESSAGE
DJNZ   R3,DISP_BUFF1
INC     DPTR                ;MOVE OVER SPACE FOR ASCII AREA
INC     DPTR                ;MOVE OVER SEMICOLON
INC     DPTR                ;MOVE OVER SPACE
; HERE INSERT CODE TO DISPLAY THE SAME BYTES IN ASCII
;     WHEN IT'S DONE THEN...
DISP_BUFF6:
MOV     R3,#16
PUSH    DPH                 ;SAVE MSG AREA POINTER
PUSH    DPL
MOV     DPH,R6
MOV     DPL,R7
MOVX   A,@DPTR             ;LOAD THE CHARACTER IN ASCII FROM DATA BUFFER
SUBB   A,#020H             ;CHECK FOR LESS THAN 20
JC     DISP_BUFF7         ;REPLACE CHAR WITH PERIOD
SUBB   A,#5FH             ;CHECK FOR GREATER THAN 7F
JNC    DISP_BUFF7
MOVX   A,@DPTR
JMP    DISP_BUFF8
DISP_BUFF7:
MOV     A,#02EH           ;REPLACE WITH PERIOD
DISP_BUFF8:
INC     DPTR
MOV     R6,DPH             MOV     R7,DPL
POP     DPL
POP     DPH
MOVX   @DPTR,A            ;SAVE IN MSG AREA
INC     DPTR
DJNZ   R3,DISP_BUFF6     ;DPO FOR ALL 16 CHARACTERS
MOV     DPTR,#DISP_BUF_MSG ;DISPLAY THE LINE
CALL    SEND_MSG
MOV     DPH,R4            ;RELOAD POINTER TO THE BUFFER AREA
MOV     DPL,R5
DJNZ   R2,DISP_BUFF2     ;DO FOR 16 LINES PER PAGE
DISP_BUFF4:
MOV     DPTR,#DISP_BUF_MSG1 ;DISPLAY MESSAGE A=aBORT OR N=NEXTÍ
CALL    SEND_MSG
CALL    GET_RESP
CJNE   A,#A_A,DISP_BUFF3 ;IF IT'S AN A THEN ABORT
RET
DISP_BUFF3:
CJNE   A,#A_N,DISP_BUFF4 ;IF IT'S AN N THEN DO ANOTHER 16 LINES OF 16 BYTES
PER LINE
MOV     DPH,R4            ;RELOAD POINTER TO THE BUFFER AREA
MOV     DPL,R5
JMP    DISP_BUFF5
;
DISP_BUF_MSG: DB     ë5XXX: 112233445566778899AABCCDDEEFF      ;123456789ABCDEF      $Í
DISP_BUF_MSG1: DB     ë ENTER A TO ABORT OR N FOR NEXT PAGE $Í
;
;MOV_A_TO_BUFF CONVERTS FROM HEX TO ASCII THEN STORES IN LOACTIONS POINTED TO BY DPTR
;
MOV_A_TO_BUFF:
SETB   RS1
NOP
MOV     R6,A
ANL    A,#0F0H

```

```

RR      A
RR      A
RR      A
RR      A
CALL    CONV_TO_ASCII
MOVX    @DPTR,A
INC     DPTR
MOV     A,R6
ANL     A,#0FH
CALL    CONV_TO_ASCII
MOVX    @DPTR,A
INC     DPTR
CLR     RS1
NOP
RET

;
;CONV_TO_ASCII
;CONVERTS LOW NIBBLE IN A TO ASCII, RETURNS IN A
;
CONV_TO_ASCII:  JB     ACC.3,CONV_HIGH
CONV_LOW1:     ORL     A,#30H
              RET
CONV_HIGH:     JNB     ACC.1,CONV_LOW
              JMP     CONV_HIGH1
CONV_LOW:      JNB     ACC.2,CONV_LOW1
CONV_HIGH1:    SUBB    A,#09
              ORL     A,#40H
              RET

;
;FORMAT_ID; FORMATS THE RESPONSE FROM ID DRIVE COMMAND AND DISPLAYS RESULT
;
FORMAT_ID:     MOV     DPTR,#DATA_BUFFER+3      ;PROCESS NUMBER OF CYLINDERS
              MOVX    A,@DPTR
              MOV     DPTR,#ID_DR_NUM_CYL+2
              CALL    MOV_A_TO_BUFF
              MOV     DPTR,#DATA_BUFFER+2
              MOVX    A,@DPTR
              MOV     DPTR,#ID_DR_NUM_CYL
              CALL    MOV_A_TO_BUFF
              MOV     DPTR,#DATA_BUFFER+6      ;PROCESS NUMBER OF HEADS
              MOVX    A,@DPTR
              MOV     DPTR,#ID_DR_NUM_HEAD
              CALL    MOV_A_TO_BUFF
              MOV     DPTR,#DATA_BUFFER+9      ;PROCESS NUMBER OF BYTES PER TRACK
              MOVX    A,@DPTR
              MOV     DPTR,#ID_DR_NUM_BPT
              CALL    MOV_A_TO_BUFF
              MOV     DPTR,#DATA_BUFFER+8
              MOVX    A,@DPTR
              MOV     DPTR,#ID_DR_NUM_BPT+2
              CALL    MOV_A_TO_BUFF
              MOV     DPTR,#DATA_BUFFER+11     ;PROCESS NUMBER OF BYTES PER SECTOR
              MOVX    A,@DPTR
              MOV     DPTR,#ID_DR_NUM_BPS
              CALL    MOV_A_TO_BUFF
              MOV     DPTR,#DATA_BUFFER+10
              MOVX    A,@DPTR
              MOV     DPTR,#ID_DR_NUM_BPS+2
              CALL    MOV_A_TO_BUFF
              MOV     DPTR,#DATA_BUFFER+13     ;PROCESS NUMBER OF SECTORS PER TRACK
              MOVX    A,@DPTR
              MOV     DPTR,#ID_DR_NUM_SPT
              CALL    MOV_A_TO_BUFF
              MOV     DPTR,#DATA_BUFFER+12
              MOVX    A,@DPTR
              MOV     DPTR,#ID_DR_NUM_SPT+2

```

```

CALL    MOV_A_TO_BUFF
MOV     R3,#20
MOV     R4,#0
MOV     DPTR,#DATA_BUFFER+20
;PROCESS SERIAL NUMBER (move 20 bytes)
FORMAT_ID2:
MOVX   A,@DPTR
MOV     R1,A
INC     DPTR
PUSH   DPL
PUSH   DPH
MOV     DPTR,#ID_DR_SERIAL
MOV     A,DPL
ADD     A,R4
MOV     DPL,A
JNC    FORMAT_ID1
INC     DPH
FORMAT_ID1:
INC     R4
MOV     A,R1
MOVX   @DPTR,A
POP     DPH
POP     DPL
DJNZ   R3,FORMAT_ID2
MOV     R3,#10
MOV     DPTR,#ID_DR_SERIAL
;Load count for 10 words
;Load Pointer
CALL   SWAP_BA_AB
;Swap the bytes
MOV     R3,#8
MOV     DPTR,#DATA_BUFFER+46
;PROCESS FIRMWARE VERSION (MOVE 8 BYTES)
FORMAT_ID4:
MOVX   A,@DPTR
MOV     R1,A
INC     DPTR
PUSH   DPL
PUSH   DPH
MOV     DPTR,#ID_DR_FW_VER
MOV     A,DPL
ADD     A,R4
MOV     DPL,A
JNC    FORMAT_ID3
INC     DPH
FORMAT_ID3:
INC     R4
MOV     A,R1
MOVX   @DPTR,A
POP     DPH
POP     DPL
DJNZ   R3,FORMAT_ID4
MOV     R3,#4
MOV     DPTR,#ID_DR_FW_VER
;Load count for 4 words
CALL   SWAP_BA_AB
;Swap the bytes
MOV     R3,#40
MOV     R4,#0
MOV     DPTR,#DATA_BUFFER+46
;PROCESS MODEL NUMBER (MOVE 40 BYTES)
FORMAT_ID6:
MOVX   A,@DPTR
MOV     R1,A
INC     DPTR
PUSH   DPL
PUSH   DPH
MOV     DPTR,#ID_DR_MODEL
MOV     A,DPL
ADD     A,R4
MOV     DPL,A
JNC    FORMAT_ID5
INC     DPH
FORMAT_ID5:
INC     R4
MOV     A,R1
MOVX   @DPTR,A
POP     DPH
POP     DPL
DJNZ   R3,FORMAT_ID6
    
```

```

MOV     R3,#20                               ;Load count for 20 words
MOV     DPTR,#ID_DR_MODEL
CALL    SWAP_BA_AB                             ;SWap the bytes
MOV     DPTR,#DATA_BUFFER+109                 ;PROCESS CURRENT NUMBER OF CYLINDERS
MOVX    A,@DPTR
MOV     DPTR,#ID_CUR_NUM_CYL
CALL    MOV_A_TO_BUFF
MOV     DPTR,#DATA_BUFFER+108
MOVX    A,@DPTR
MOV     DPTR,#ID_CUR_NUM_CYL+2
CALL    MOV_A_TO_BUFF
MOV     DPTR,#DATA_BUFFER+111                 ;PROCESS CURRENT NUMBER OF HEADS
MOVX    A,@DPTR
MOV     DPTR,#ID_CUR_NUM_HDS
CALL    MOV_A_TO_BUFF
MOV     DPTR,#DATA_BUFFER+110
MOVX    A,@DPTR                               MOV     DPTR,#ID_CUR_NUM_HDS+2
CALL    MOV_A_TO_BUFF
MOV     DPTR,#DATA_BUFFER+113                 ;PROCESS CURRENT NUMBER OF SECTORS PER
TRACK
MOVX    A,@DPTR
MOV     DPTR,#ID_CUR_NUM_SPT
CALL    MOV_A_TO_BUFF
MOV     DPTR,#DATA_BUFFER+112
MOVX    A,@DPTR
MOV     DPTR,#ID_CUR_NUM_SPT+2
CALL    MOV_A_TO_BUFF                             ;PROCESS CURRENT CAP IN SECTORS
MOV     DPTR,#DATA_BUFFER+116
MOVX    A,@DPTR
MOV     DPTR,#ID_CUR_CAP_SEC
CALL    MOV_A_TO_BUFF
MOV     DPTR,#DATA_BUFFER+115
MOVX    A,@DPTR
MOV     DPTR,#ID_CUR_CAP_SEC+2
CALL    MOV_A_TO_BUFF
MOV     DPTR,#DATA_BUFFER+114
MOVX    A,@DPTR
MOV     DPTR,#ID_CUR_CAP_SEC+4
CALL    MOV_A_TO_BUFF
;
;
;
;
MOV     DPTR,#ID_CUR_CAP_SEC+6                 ;PROCESS LBA MODE SECTORS
CALL    MOV_A_TO_BUFF
MOV     DPTR,#DATA_BUFFER+122
MOVX    A,@DPTR
MOV     DPTR,#ID_TOT_LBA_SEC
CALL    MOV_A_TO_BUFF
MOV     DPTR,#DATA_BUFFER+121
MOVX    A,@DPTR
MOV     DPTR,#ID_TOT_LBA_SEC+2
CALL    MOV_A_TO_BUFF
MOV     DPTR,#DATA_BUFFER+120
MOVX    A,@DPTR
MOV     DPTR,#ID_TOT_LBA_SEC+4
CALL    MOV_A_TO_BUFF
RET
;
;
;SWAP_BA_AB
;
;   SWAPS BYTES SO TEXT DATA LOOKS RIGHT
;
SWAP_BA_AB:   NOP
RST_SWAP:    MOVX    A,@DPTR           ;LOAD ONE BYTE
              MOV     R1,A
              INC     DPTR
    
```

```

MOVX  A,@DPTR      ;LOAD SECOND BYTE
MOV   R2,A
MOV   A,R1
MOVX  @DPTR,A     ;Save First byte at second location
DEC   DPL
MOV   A,R2
MOVX  @DPTR,A     ;Save Second byte at first location
INC   DPTR
INC   DPTR
DJNZ  R3,RST_SWAPEOSWAP:      RET

;
;  MESSAGES AREA
;
MAIN_MENU:        DB   ëMAIN MENUí
                  DW   0D0AH
                  DB   ë  1. DIAGNOSTICS MENUí
                  DW   0D0AH
                  DB   ë  2. SET MODE MENUí
                  DW   0D0AH
                  DB   ë  3. READ A SECTORí
                  DW   0D0AH
                  DB   ë  4. WRITE A SECTORí
                  DW   0D0AH
                  DB   ë  5. DISPLAY BUFFER DATAí
                  DW   0D0AH
                  DB   ë  6. DRIVE COMMANDS MENUí
                  DW   0D0AH
                  DB   ë  7. EXIT THE PROGRAM$í
ENT_NUM_MSG:     DB   ë ENTER NUMBER: $í
DIAG_MENU:       DB   ëDIAGNOSTICS MENUí
                  DW   0D0AH
                  DB   ë  1 RUN DIAGS$í
                  DW   0D0AH
                  DB   ë  2 SOFT RESETí
                  DW   0D0AH
                  DB   ë  3 HARD RESETí
                  DW   0D0AH
                  DB   ë  4 READ AN ADDRESSí
                  DW   0D0AH
                  DB   ë  5 WRITE AN ADDRESSí
                  DW   0D0AH
                  DB   ë  6 RETURN TO MAIN MENU$í
MODE_MENU:       DB   ëMODE MENUí
                  DW   0D0AH
                  DB   ë  1. SET MEMORY MODEí
                  DW   0D0AH
                  DB   ë  2. SET IO MODEí
                  DW   0D0AH
                  DB   ë  3. SET IDE MODEí
                  DW   0D0AH
                  DB   ë  4. RETURN TO MAIN MENU$í
CMD_MENU:        DB   ëCommand Menuí
                  DW   0d0aH
                  DB   ë  1. CHECK POWER MODEí
                  DW   0D0AH
                  DB   ë  2. IDENTIFY DEVICEí
                  DW   0D0AH
                  DB   ë  3. ERASE SECTORí
                  DW   0D0AH
                  DB   ë  4. IDLEí
                  DW   0D0AH
                  DB   ë  5. RECALIBRATEí
                  DW   0D0AH
                  DB   ë  6. SEEKí
                  DW   0D0AH

```

```

DB      ë      7. SET SLEEP MODE$í
DW      0D0AH
DB      ë      8. MORE COMMMANDS$í
DW      0D0AH          DB      ë      9. RETURN TO MAIN MENY $í
;
MORE_CMD_MENU:
DB      ëMORE DRIVE COMMANDS MENU$í
DW      0D0AH
DB      ë      1.SET FEATURES COMMAND ë
DW      0D0AH
DB      ë      9. RETURN TO DRIVE COMMAND MENU$í
;
ERROR0_MSG:
DB      ëCARD NOT PRESENT$í
ERROR1_MSG:
DB      ëDRIVE NOT READY$í
ERROR2_MSG:
DB      ëStatus Error Code ë
ERROR2_CODE:
DB      00H
DB      00H
DB      ë$í
ERROR3_MSG:
DB      ëGENERAL ERROR $í
ERROR4_MSG:
DB      ëABORT DUE TO WRITE FAULT, NOT READY OR BAD COMMAND $í
ERROR5_MSG:
DB      ëSECTOR ID CANNOT BE FOUND $í
ERROR6_MSG:
DB      ëUNCORRECTABLE ERROR $í
ERROR7_MSG:
DB      ëBAD BLOCK DETECTED $í
ERROR8_MSG:
DB      ëInvalid entry $í
CRLF:
DB      0DH
DB      0AH
DB      ë$í
;
DIAGS_MSG0:
db      ëDIAGNOSTICS COMMAND COMPLETED $í
DIAGS_MSG1:
DB      ëNO ERROR DETECTED$í
DIAGS_MSG2:
DB      ëFORMATTER DEVICE ERROR$í
DIAGS_MSG3:
DB      ëSECTOR BUFFER ERROR$í
DIAGS_MSG4:
DB      ëECC CIRCUITRY ERROR$í
DIAGS_MSG5:
DB      ëCONTROLLING MICROPROCESSOR ERROR$í
DIAGS_MSG6:
DB      ëSLAVE ERROR IN TRUE IDE MODE$í
DIAGS_MSG7:
DB      ëError code not listed$í
;
; ID DRIVE DISPLAY INDICATING DRIVE PARAMETERS
ID_DRIVE_MSG: DB      ëIDENTIFY DRIVE RESULT$í
DW      0D0AH
DB      ëMODEL NUMBER OF DRIVE: ë
ID_DR_MODEL:
DB      ë
DW      0D0AH
DB      ëNUMBER OF CYLINDERS ë
ID_DR_NUM_CYL:
DW      0000H
DB      ë      NUMBER OF HEADS ë
ID_DR_NUM_HEAD:
DB      00H
DB      ë ë
DW      0D0AH
DB      ëNUMBER OF BYTES PER TRACK ë
ID_DR_NUM_BPT:
DW      0000H
DB      ë      NUMBER OF BYTES PER SECTOR ë
ID_DR_NUM_BPS:
DW      0000H
DB      ë ë
DW      0D0AH
DB      ëNUMBER OF SECTORS PER TRACK ë
ID_DR_NUM_SPT:
DB      ë ë
DW      0D0AH
DB      ëSERIAL NUMBER ë
ID_DR_SERIAL: DB      ë
DB      ë      FIRMWARE VERSION: ë
ID_DR_FW_VER: DB      ë
DW      0D0AH
DB      ëCURRENT NUM OF CYLINDERS ë

```

# Need More System Storage? Embed a Hitachi Microdrive!

```
ID_CUR_NUM_CYL:    DB      0      0      0      0      DB      0      0      CURENT NUM OF HEADS 0
ID_CUR_NUM_HDS:    DB      0      0      0      0      DW      0D0AH
                  DB      0      0      0      0      DB      0      0      CURENT SECTORS PER TRACK 0
ID_CUR_NUM_SPT:    DB      0      0      0      0      DB      0      0      CURENT CAPACITY IN SECTORS 0
ID_CUR_CAP_SEC:    DB      0      0      0      0      DW      0D0AH
                  DB      0      0      0      0      DB      0      0      TOTAL LBA MODE SECTORS : 0
ID_TOT_LBA_SEC:    DB      0      0      0      0      DW      0D0AH
                  DB      0      0      0      0      DB      0      0      $1
; RESERVED MEMORY LOCATIONS FOR DATA MEMORY
CONFIG_REG_BASE:   DB      0,0      ;Memory space to hold Config Base Address
SECTOR_LBA_ADDRESS: DB      0,3,0    ;LOCATION FOR SECTOR ADDRESS DEFAULT 0300
SECTOR_CNT:        DB      1      ;LOCATION FOR SECTOR COUNT FOR READ/WRITE DEFAULT 1
                  DSEG
                  ORG      5400H    ;AREA FOR DATA
DATA_BUFFER:       DS      512
KBUFF:             DS      80

END
```

Hitachi Global Storage Technologies trademarks are intended and authorized for use only in countries and jurisdictions in which Hitachi Global Storage Technologies has obtained the rights to use, market and advertise the brand. The Travelstar trademark is authorized for use in the Americas, EMEA, and the following Asia-Pacific countries and jurisdictions: Australia, Hong Kong, Japan, New Zealand, South Korea and Taiwan. Contact Hitachi Global Storage Technologies for additional information. Hitachi Global Storage Technologies shall not be liable to third parties for unauthorized use of this document or unauthorized use of its trademarks.

References in this publication to Hitachi Global Storage Technologies' products, programs or services do not imply that Hitachi Global Storage Technologies intends to make these available in all countries in which it operates.

Product specifications provided are sample specifications and do not constitute a warranty. Information is true as of the date of publication and is subject to change. Actual specifications for unique part numbers may vary. Please visit the Support section of our website, [www.hitachigst.com/support](http://www.hitachigst.com/support), for additional information on product specifications. Photographs may show design models.

© 2007 Hitachi Global Storage Technologies

Hitachi Global Storage Technologies  
3403 Yerba Buena Road  
San Jose, CA 95135 USA

Produced in the United States 11/07.  
All rights reserved.

Microdrive® and Adaptive Battery Life Extender™ (ABLE) are registered trademarks of Hitachi Global Storage Technologies.