application note

powerful adaptec		
chip set simplifies disk controller design		
G. Venkatesh		
Applications Manager		
August 1984		



Additional copies of this application note, or other Adaptec literature, may be obtained from: Adaptec, Inc. Literature Department 580 Cottonwood Drive Milpitas, CA 95035 The information in this document is subject to change without notice. No part of this document may be copied or reproduced in any form or by any means without prior written consent of Adaptec, Inc.

> ADAPTEC, INC., 580 Cottonwood Drive, Milpitas, California 95035 Copyright 1984

TABLE OF CONTENTS

1.0	Intro	oduction	1
2.0	Data	Buffering and Transferring	5
	2.1	Port A Transfer	
	2.2	Port B Transfer	6
	2.3	Data Transfer Overview	8
	2.4	RAM Buffer Interface	9
3.0	Data	Formatting and Sequencing	3
	3.1	Drive Interface	3
	3.2	Register Overview	1
	٥.2		٦
4.0	Supp	port Processor	7
	4.1	Processor Interface	7
	4.2	Disk Control and Status Monitoring	8
	4.3	Host Bus Access (Registers 50 and 51)	1
	4.4	RAM Buffer Access (Register 70)	1
5.0	Seria	al Data Encoding/Decoding	3
6.0	Sum	mary	7
7.0	App	endices	9
	7.1	Flowcharts	a
	7.2	Sample Routines	
	7.3	Programming 8-Bit ECC Correction	5
	7.4	Typical VCO/PLL Schematic	5
	7.5	Other Microprocessor Interfaces	S S
			(1

1.0 INTRODUCTION

To take advantage of the advances in CPU technology, designers need a high performance rigid disk controller that can effectively break the I/O bottleneck. Unlike disk controllers in the past, new implementations have to be flexible and powerful. In fact, today's design must be capable of being an intelligent I/O subsystem, instead of being a drain on the host CPU. In addition, the disk controllers need to be easily adaptable to recent enhancements in the Winchester disk drive technology.

To be flexible, a controller design must be capable of not only handling the slower drive interfaces such as the ST506 and SA1000, but also enhancements like the ST412HP (High Performance) and other high performance interfaces like ESDI and SMD. In addition, the controller must be able to meet not only the original specifications but also various deviations from them. The controller therefore has to deal with variations in track density, track formatting, data transfer rates, sector sizes and many other things.

The Adaptec disk controller chip set provides the major portion of the hardware necessary to build a high performance rigid disk controller. Three custom LSI circuits make up the set: a single chip controller, an encoder-decoder chip, and a buffer controller. The AIC-100 disk controller chip handles high-speed sequencing of the data, leaving slower drive-control operations to an inexpensive support microprocessor. This approach enables the use of the same chip set with all of the industry-standard disk interfaces (ST506, SA1000, ESDI, SMD and ST412HP).

The AIC-250 LSI encoder-decoder circuit along with a data separator forms all that is necessary for NRZ (non return-to-zero) to MFM (modified frequency modulation) conversion and back. The data separator includes the phase locked loop and the voltage controlled oscillator that are necessary to separate the data from the clock during an MFM read data operation. The chip also performs the address mark generation, address mark detection, and write precompensation functions.

In most systems with shared DMA channels, adequate cycle times are not available to transfer data to and from the disk every 800 nsecs per byte (10 Mbits/sec designs) or every 1.6 µsecs per byte (5 Mbits/sec designs). Without some protection built into the board, even a slow Winchester (ST506) could overrun the DMA with some regularity. The AIC-300 dual ported buffer controller can address up to 64K Bytes of RAM buffer in a true dual-ported buffer configuration that guarantees non-interleaved disk formatting and no host overruns. In addition, the chip greatly simplifies the implementation of the popular SCSI (Small Computer System Interface) bus.

A typical controller block diagram is shown in Figure 1.

By partitioning the function between the different chips, the Adaptec disk controller solution addresses many major growth areas in the disk drive market. Some of the extremely important parameters are:

- Number of Head Selection Lines: To give way to the demand for increased number of heads in the next generation of drives, this is controlled by a low cost support processor.
- Seek Commands: To handle a variety of requirements from step-pulse line for seek control (in ST506 type interfaces) to structured serial output commands (such as direct addressing in ESDI type interfaces), this function is again best handled by the support processor.

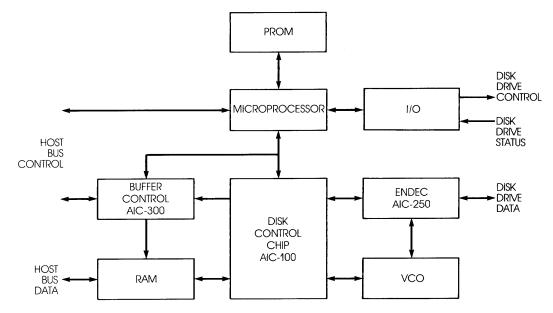


Figure 1. Typical Controller Block Diagram

- Hard or Soft Sector Formatting: The new combination of fixed and removeable drives require hard sector recognition, a function that is handled by the AIC-100 controller chip. A requirement for digital data processing in graphics and sound extends sector lengths to as much as a full track, also supported by the AIC-100.
- Placement of Encoding or Decoding Functions: Drive manufacturers have taken different approaches to the handling of data. Some drive interfaces require MFM data while other's require NRZ data. By separating the NRZ from/to MFM conversions to the AIC-250 ENDEC chip, the AIC-100 disk controller can be used in a variety of controller designs.
- Error Detection and Correction: All data correction should be transparent to the host, so as not to waste valuable processing time. The AIC-100 controller chip offers the best placement for this function.
- Data Buffer Control: This function can be easily yet economically handled by using the AIC-300 buffer controller to convert inexpensive RAM into a truly dual ported FIFO. The size of the buffer can be based on the application, up to a maximum of 64K bytes.

This Application Note discusses the use of the Adaptec disk controller chip set in designing a high performance Winchester disk controller.

Section 2 describes the data transfer between the host and the controller. The performance benefits of using the AIC-300 dual port controller will be discussed.

Section 3 describes the use of the AIC-100 disk controller chip in performing the formatting and high speed sequencing functions.

Section 4 discusses the use of a support processor in monitoring the operations of the AIC-100 and the AIC-300. In addition, the implementation of some popular drive interfaces will also be described.

Section 5 deals with the encoding/decoding of the serial data. Here, the use of the AIC-250 ENDEC chip in the NRZ from/to MFM conversion will be discussed.

Section 7 is the Appendix, and gives some additional detail on the use of the Adaptec chips in the design of a high performance controller.

Section 6 is a summary, and sums up this Application Note.

2.0 DATA BUFFERING AND TRANSFERRING

2.1 PORT A TRANSFER

The effectiveness of a controller design is based on its ability to transfer data to and from the disk as fast as physically possible, and as accurately through Error Detection and Correction (EDAC). Since the drive is continuously spinning during a read or write operation, a byte has to be transferred from/to the AIC-100 disk controller chip every 800 nsec (10 MHz operation) or every 1.6 μ secs (5 MHz operation). The AIC-100 chip either indicates the availability of a byte (during disk read) or requests a byte (during disk write), once every byte time through a signal output on the CLKB line (pin 3). This is shown in Figure 2.

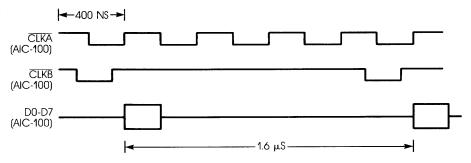


Figure 2a. AIC-300 Port A Data Byte Transfer Timing-5 MHz

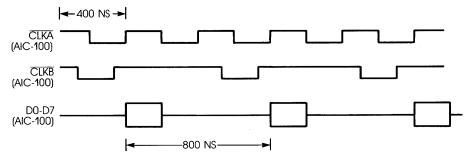


Figure 2b. AIC-300 Port A Data Byte Transfer Timing-10 MHz

This data byte is transferred between the AIC-100 and the RAM buffer and is called a Port A transfer. The appropriate address in the RAM is generated by the AIC-300 buffer controller chip, from a set of pointer registers present in the chip. The interface between the AIC-100 disk controller chip and the AIC-300 buffer controller chip necessary to accomplish this transfer is very simple, and is shown in Figure 3.

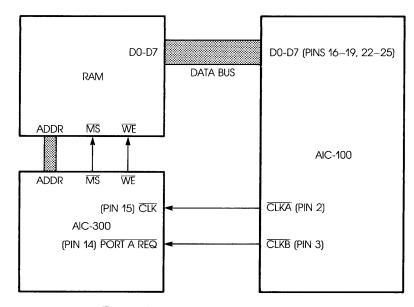


Figure 3. Port A Transfer Interface

2.2 PORT B TRANSFER

The data transfer between the RAM buffer and the host is initiated and controlled by the AIC-300 buffer controller chip, and is referred to as the Port B transfer. The AIC-300 generates the necessary request signal (Port B Req), RAM buffer address and control signals, and the host bus latch control signals. This is shown in Figure 4. The buffer controller uses the Port B Request (pin 39) to initiate the transfer which is completed after a Port B Acknowledge is received (sampled at pin 38).

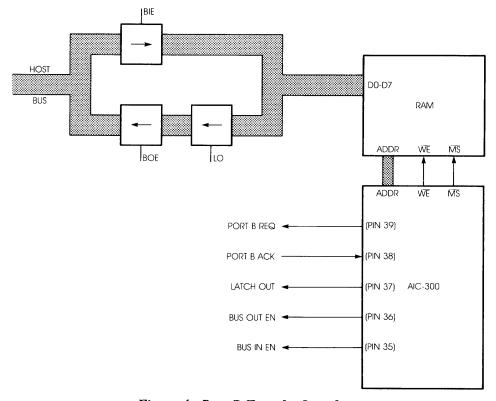


Figure 4. Port B Transfer Interface

The relationship between the different control signals during a Port B transfer are shown in Figure 5.

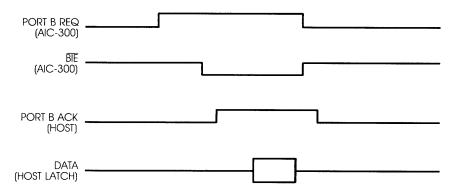


Figure 5a. AIC-300 Port B Data Transfer-Host to Buffer

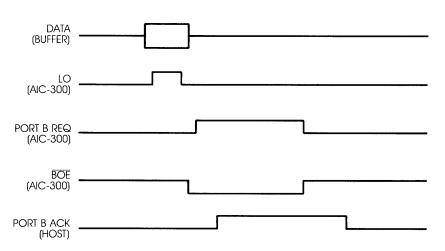


Figure 5b. AIC-300 Port B Data Transfer-Buffer to Host

2.3 DATA TRANSFER OVERVIEW

In a controller design, the AIC-300 provides all the necessary signals to interface to the host and the disk controller chip. The buffer controller also generates the necessary control signals to access the buffer, alternating between Port A and Port B transfers. While the Port A transfer is synchronous in nature (at the data transfer frequency), the Port B transfer is asynchronous and is based on a Req/Ack handshake with the host. Figure 6 shows the timing overview.

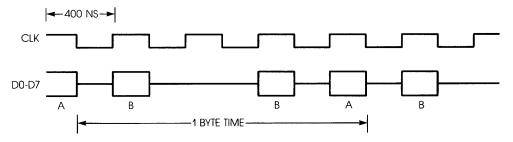


Figure 6a. Dual Port Buffer Timing Overview-5 MHz

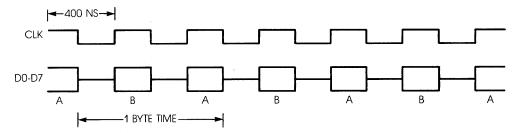


Figure 6b. Dual Port Buffer Timing Overview-10 MHz

Thus during any data transfer operation, even though the access to the buffer is interleaved between Port A and Port B, the AIC-100 disk controller chip should stay one sector ahead of the host. This is necessary to keep error detection and correction transparent to the host. The AIC-300 has an internal stop pointer which is used to prevent overruns. An overview of the buffer operation is shown in Figure 7.

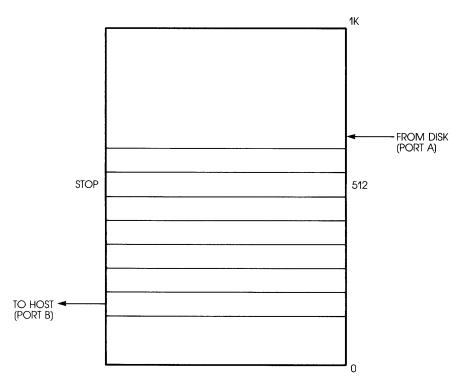


Figure 7. Buffer Operation Overview

Thus during a read operation, while data is being transferred into the buffer from the drive (Port A transfer), at the same time, the previous sector can be transferred to the host (Port B transfer). The stop pointer is set to the end of the sector being transferred to the host, and prevents an overrun. At the conclusion of a successful sector read, the stop pointer can be updated to point to the end of this sector. Now this sector can also be transferred to the host, while yet another sector is read in.

2.4 RAM BUFFER INTERFACE

As mentioned earlier, the buffer controller is capable of managing up to a 64K byte buffer. The chip provides the address, memory select and read/write lines necessary to perform this function. There are two distinct modes of RAM interface: equal to or less than 1K bytes (up to 10 address lines), and greater than 1K bytes (up to 16 address lines).

In the 10 bit or less addressing mode, two special lines supply A8 and A9 address lines. An example is shown in Figure 8.

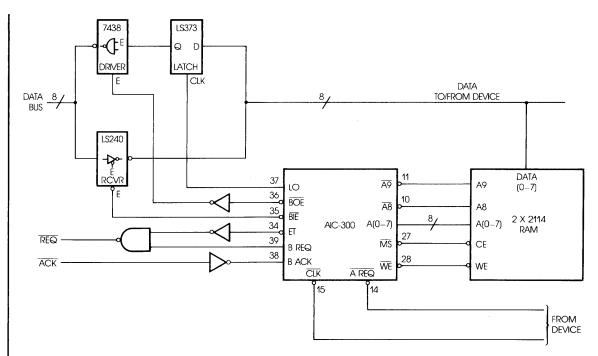


Figure 8. 10-Bit Addressing Application Example

In the 16 bit addressing mode (greater than 10), the higher order address lines (A8-A15) and the lower order address lines (A0-A7) are multiplexed coming out of the chip, on pins A0 to A7. In this mode, the two special lines supply the strobes SDP and SHP used to demultiplex these signals during a disk transfer (port A) and a host transfer (port B) respectively. An example is shown in Figure 9.

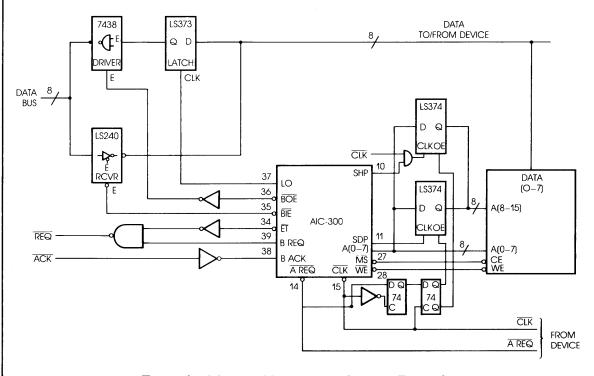


Figure 9. 16-Bit Addressing Application Example

Internal to the AIC-300 are two sets of pointer registers used to generate the RAM addresses. These are the Read Address Pointer (RAP) and the Write Address Pointer (WAP). The actual register used depends on which port is selected and the direction of the data transfer, and is controlled by the value of the ROP/WOP (Read Operation/Write Operation) bit in the DMA control register (Reg 53, bit 4). In addition to this, another pair of registers called the Stop Pointer (SP) is used to control data transfers between the host and the buffer. The different possibilities are shown in Table 1.

Table 1. Buffer Address Generation

ROP/WOP	Source for Address	Direction of Transfer
Read Disk		
1	WAP	Disk to Buffer
1	RAP	Buffer to Host $(RAP < SP)$
Write Disk	•	
0	RAP	Buffer to Disk
0	WAP	Host to Buffer (WAP < SP)

3.0 | DATA FORMATTING AND SEQUENCING

3.1 DRIVE INTERFACE

The AIC-100 Winchester disk controller chip handles all the data formatting and sequencing necessary to interface with the drive.

During a write operation, parallel data is transferred into the chip from the RAM buffer. This data is passed through a 32 bit shift register before it is output as serial data in the NRZ format. During this time, a 4 byte ECC is also computed. After the entire sector is written to the drive, the ECC bytes are also output in a serial fashion.

During a read operation, NRZ data is read into the controller chip and byte parallel data is transferred to the RAM buffer. Error checking is also performed at this time, using the four ECC bytes stored at the end of the sector. If there is an error, the support processor usually initiates retries to determine if the error is correctable. If the error is determined to be correctable, the error syndrome information found in the AIC-100 chip is used to correct the data in the buffer. This process is transparent to the host.

The serial data is synchronized through a read reference clock that is input to the AIC-100 chip. The disk controller chip has the necessary logic to look for index mark and sector mark (on hard sectored drives), in addition to generating the Read Gate and Write Gate signals. The disk controller also controls the writing of the address mark during a write operation, and looks at when the address mark is detected during a read operation. Figure 10 shows the necessary drive interface lines.

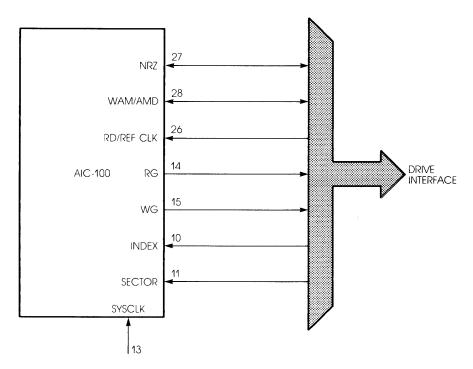


Figure 10. AIC-100 Drive Interface Signals

3.2 REGISTER OVERVIEW

Internally, the AIC-100 disk controller is made up of 17 registers, used to control the high speed sequencing functions. The slower functions such as seek, head selection, drive selection, and drive status monitoring are handled by an external support microprocessor and some I/O port logic. This is described in the following section.

In addition to the 17 internal registers, the AIC-100 chip also decodes accesses to 5 external register addresses which can be used to simplify controller design. The registers can be grouped by function as follows:

- Stored Value Registers
- Command Ste-up Registers
- Command Register, Register 78
- Status/Execution Register, Register 79
- External Registers

Table 2 shows the registers in each group and their function.

The operation of the AIC-100 chip revolves around two of these registers: the command register (Reg 78), and the status register (Reg 79). These registers have to be set up and monitored by the support microprocessor for all data sequencing operations.

Table 2.

Register	Title	R=Read/W=Write	Function
	STORED V	ALUE REGISTERS -	
72	ECC Bits 0-16 "OR"ed and 17-23	R	Syndrome bits.
73	ECC Bits 24-31	R	Error bits.
D0	Gap Length	W	Bit control.
E0	Cylinder Byte	R/W	ID field byte.
E1	Head Byte	R/W	ID field byte.
E2	Sector Byte	R/W	ID field byte.
E3	Flag Byte	R/W	
	COMMAND	SET-UP REGISTERS	•
71	ECC Control	W	Correction control.
74	ECC Polynomial	W	Low order bits.
77	ECC Polynomial	W	High order bits.
7 A	Operation Modifier	R/W	Operation control.
7E	Special I/O	R	Input and data transfer bits.
7 F	POP Stack	R	LIFO stack read.
7F	Clock Control	W	CLKA control.
A 4	Search Bit	W	Enables search.
C4	Sector Length	W	Sector data field length.
	COMM	AND REGISTER	
78	Operation Command	W	Controls the sequencing of the controller chip.
	STATUS/EXE	ECUTION REGISTER	S
79	Chip Status	R	Used to monitor chip status.
79	Start Execution	W	Used to start chip execution.
	EXTER1	NAL REGISTERS —	
50	Host Data Transfer	R/W	Allows processor to R/W data
<i>5</i> 1	Heat Data Township	D/W	directly from the host.
51	Host Data Transfer	R/W	Exactly like Register 50.
70	Buffer Data Transfer	R/W	Allows processor to R/W data
6E	External Line Control	R/W	directly from the buffer. Causes the pins designated RD6E and WR6E to be
6F	External Line Control	R/W	activated by the processor. Same as above, but for pins RD6F and WR6F.

4.0 | SUPPORT PROCESSOR

4.1 PROCESSOR INTERFACE

The AIC-100 chip and the AIC-300 chip are setup and monitored by a support processor. The interface to the support microprocessor is through a multiplexed address/data bus as is found in the Intel 8085 family of microprocessors. This can however be easily adapted to other processors, such as the Z80, through the use of minimal external logic. Some examples of this are shown in Appendix 7.5. For the rest of this controller design application note, an Intel 8085 is used. The basic interface between the 8085 and the Adaptec chips is shown in Figure 11.

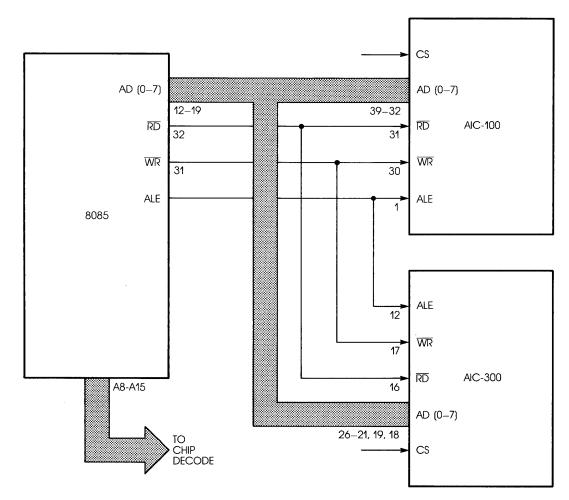


Figure 11. Support Processor Interface

The support processor is used to maintain "loose" synchronization with what is happening in real-time on the disk through the OP command and Sequencer Status registers in the AIC-100 (Reg 78 and Reg 79). Based on the operation, the processor also sets up the registers in the AIC-300 to control the data transfer to and from the buffer.

The sequence of the actual steps that have to be followed with the AIC-100 and the AIC-300 for a READ, WRITE, and FORMAT disk operation is shown in the form of flow charts in Appendix 7.1. The actual source code is shown in Appendix 7.2.

DISK CONTROL AND STATUS MONITORING

An important function of the support processor is the management of the low speed drive interface function. By designating this operation to the support processor, the Adaptec chip set can be easily adapted to a variety of drive interfaces and drive types.

In the case of a ST506/412 interface, the support processor has to perform the following functions:

- Recalibrate to Track 000.
- Monitoring Drive Ready and Write Fault.
- Selecting the appropriate drive and head.
- Seeking to the desired cylinder.

This can be easily accomplished through the use of I/O ports and firmware. The AIC-100 chip internally decodes accesses to addresses 6E and 6F. A read or write to these two addresses cause the appropriate pin (pin 4, 5, 6, or 7) on the AIC-100 chip to be asserted. Thus by placing external I/O ports at addresses 6E and 6F, and using this feature of the AIC-100 chip, the designer can eliminate external decode logic. An example of this implementation for a ST506/412 or a ST412HP interface is shown in Figure 12.

Since the support processor is used for drive control, a variation in hardware and a change in the firmware enables the Adaptec chip set to be used for other drive interfaces. An example of the implementation for an SMD interface is shown in Figure 13.

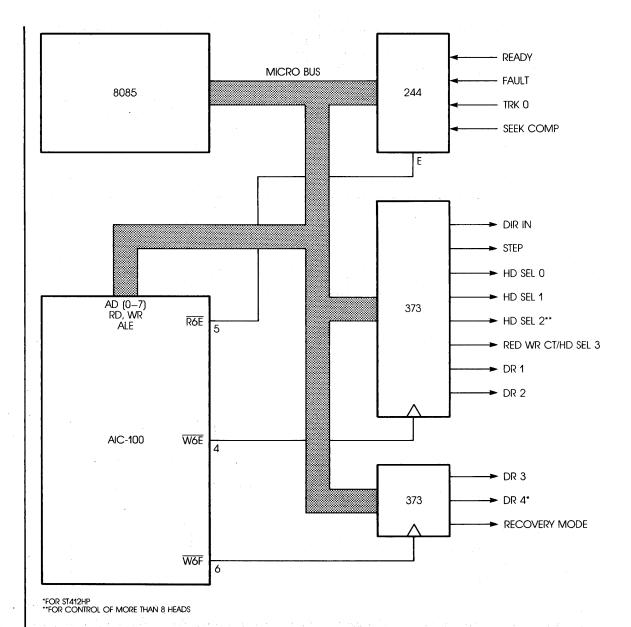
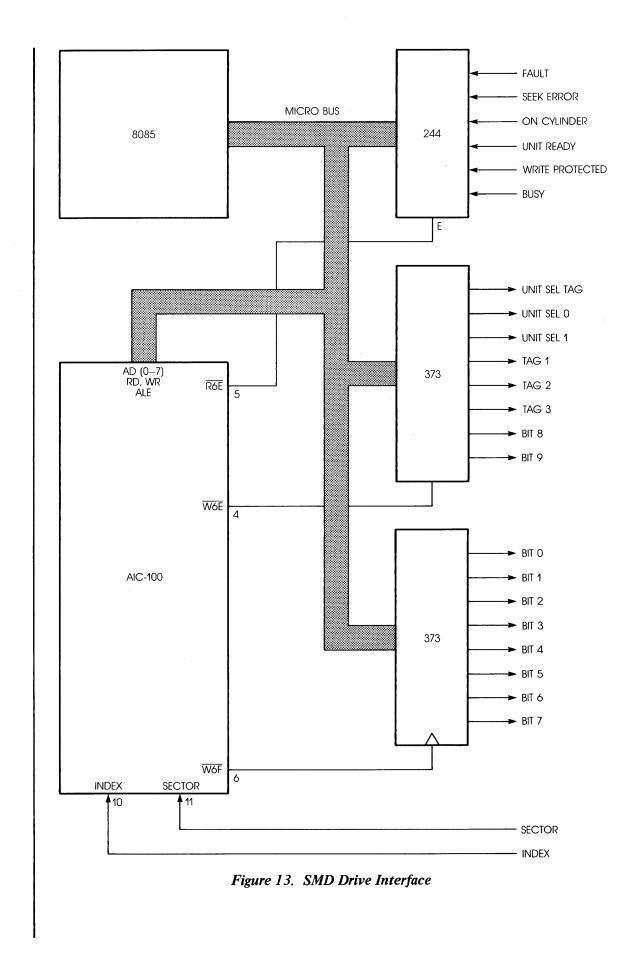


Figure 12. ST506/412 and ST412HP Interface



4.3 HOST BUS ACCESS (REGISTERS 50 AND 51)

In the design of intelligent controllers, a popular host interface used is the SCSI (Small Computer System Interface), previously known as the SASI. Adaptec chip set based designs ease the implementation of the SCSI, since a majority of the functionality needed for this is inherent in the overall architecture and, in particular, in the AIC-300 chip.

While the AIC-300 buffer controller chip can handle data transfers between the host bus and the buffer, it may sometimes be necessary for the support processor to directly access the host data bus. This is the case in an SCSI implementation during the transfer of the Command Data Block (CDB) and message bytes. This can be accomplished by the support processor through an access to Register 50 or 51.

An access to either register is decoded by both the AIC-100 and the AIC-300. The AIC-100 chip internally bridges the support processor data bus and the RAM buffer data bus, thus offering a data path to host latch and receivers. The AIC-300 chip asserts the control signals necessary to access the host bus. During a read, the BIE line from the AIC-300 is asserted. During a write, the LO line will be first asserted, followed by the BOE line. This allows the data to be latched first before being enabled onto the host data bus. During a read or a write to Register 50 or 51, the data is passed through the AIC-100. An example is shown in Figure 14.

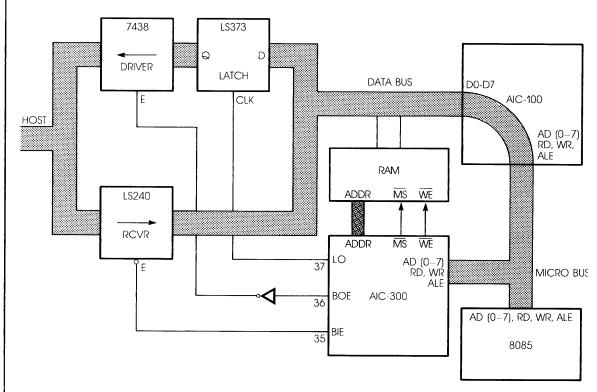


Figure 14. Host Bus Access

4.4 RAM BUFFER ACCESS (REGISTER 70)

During the process of transferring data between the disk and the host, the support processor has to sometimes have access to the sector data. This is especially necessary during a read operation if a correctable ECC error is encountered. Using the error

syndrome information found in the AIC-100 chip, the support processor first calculates an error mask and a displacement. The algorithm necessary for this is shown in Appendix 7.3.

After that, the actual data stored in the buffer has to be read, modified, and written back. This is done through an access to Register 70.

Here again the AIC-100 decodes an access to Register 70 and bridges the processor data bus and the buffer data bus. This allows the transfer of data between the support processor and the RAM.

The AIC-300 decodes an access to Register 70 and asserts memory select and read/write to the RAM. The address selected is the contents of the WAP registers if the ROP/WOP bit (Reg 53, bit 4) is set (Read Disk) and the contents of the RAP registers if the ROP/WOP bit is reset (Write Disk). This path is shown in Figure 15.

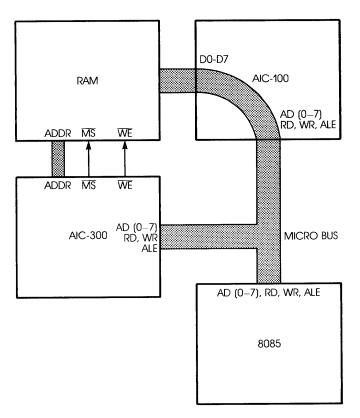


Figure 15. RAM Buffer Access

5.0 | SERIAL DATA ENCODING/DECODING

The AIC-100 controller chip converts parallel data to NRZ serial data and back. In drive interfaces, such as SMD and ESDI, serial data is written to, and read from, the drive interface in NRZ format. However, in drive interfaces like the ST506/412, SA1000, and the ST412HP, the data is transferred in MFM format. In this case, the main difference is the transfer rate, shown in Table 3.

Table 3. MFM Transfer Rates

Interface	Transfer Rate
SA1000	4.34 Mbits/s
ST506/412	5.0 Mbits/s
ST412HP	10.0 Mbits/s

The AIC-250 Encoder/Decoder chip (ENDEC) contains a major portion of the logic necessary to convert data in the NRZ format to/from MFM format. In addition, the chip provides the logic necessary for address mark detection and writing. The chip also has the necessary circuitry for write precompensation which is necessary to compensate for certain bit patterns.

During a read disk operation, an external VCO/PLL is required to separate the data from the clock before it can be converted to NRZ data. A typical VCO/PLL circuit is shown in Appendix 7.4.

An example of the ENDEC interface circuit is shown in Figure 16.

The AIC-250 simplifies ST506/412 and ST412HP interface Winchester disk controller designs due to the fact that it needs very few external passive components, unlike other implementations which require costly delay lines. A typical configuration of the ENDEC for a 5 MHz data rate is shown in Figure 17.

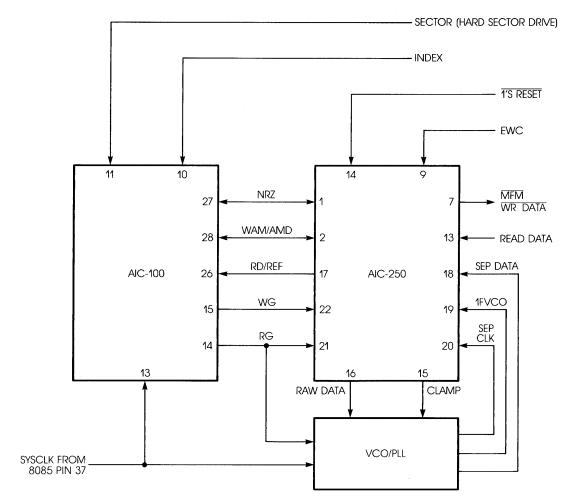


Figure 16. NRZ To/From MFM Conversion

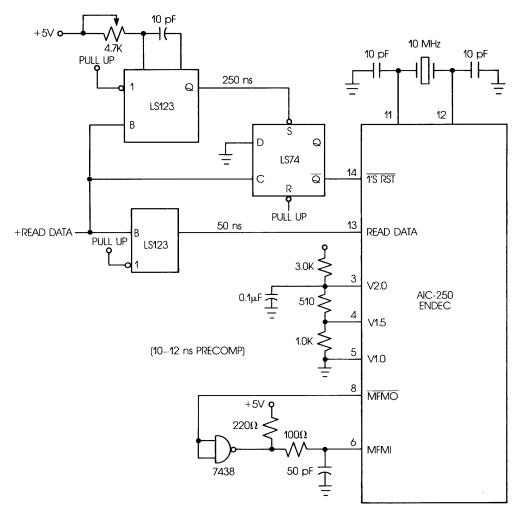


Figure 17. External Component Requirement at 5 MHz Data

6.0 | SUMMARY

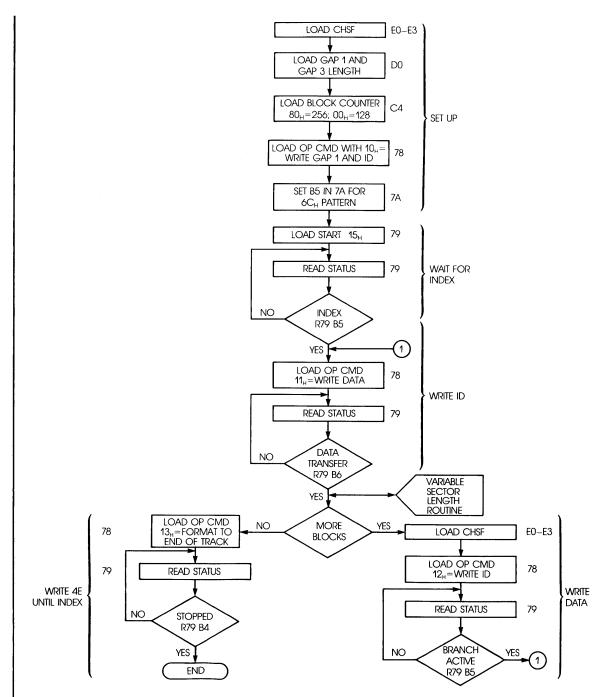
As with all growing technologies, 5¼" Winchester drives will undoubtedly move beyond the capacity of today's units. Thus a designer who is designing a controller board today must consider the needs of tomorrow. To that end, by cleverly partitioning features among various chips in the Adaptec family, and by putting the burden of slow speed disk control and status monitoring on the support processor, future design issues can also be addressed. All of this of course being accomplished without sacrificing performance or increasing cost.

7.0 **APPENDICES**

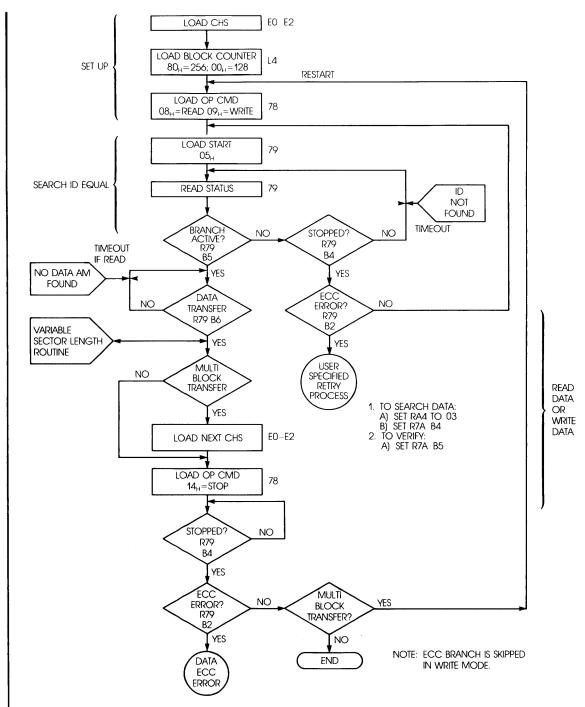
7.1 FLOWCHARTS

In order to read or write data from/to the peripheral, the support processor has to loosely control the operation of the AIC-100 Winchester disk controller chip and the AIC-300 dual ported buffer controller chip. The flowcharts necessary to set up and monitor these two chips are as follows:

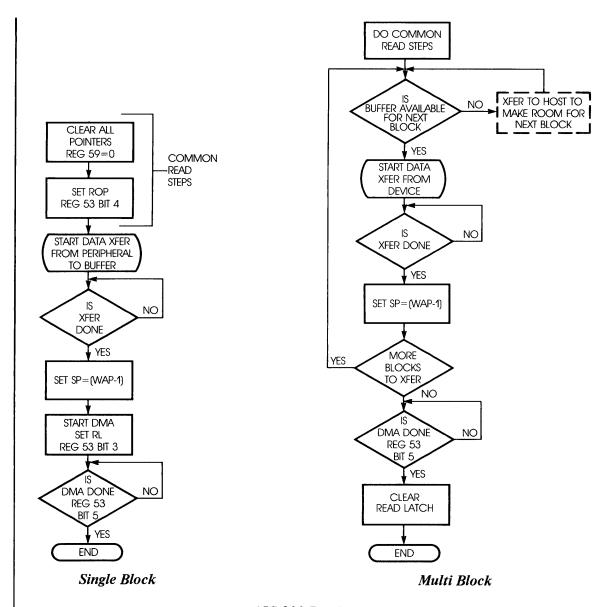
- AIC-100
 - □ Soft Sector Format
 - □ Soft Sector Read/Write (Single and Multi Block)
- AIC-300
 - □ Read Disk (Single and Multi Block)
 - □ Write Disk (Single and Multi Block)



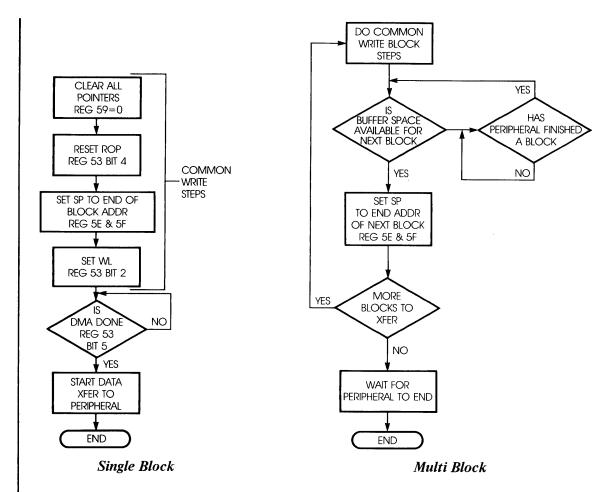
AIC-100 Soft Sector Format



AIC-100 Soft Sector Read/Write (Single and Multi Block)



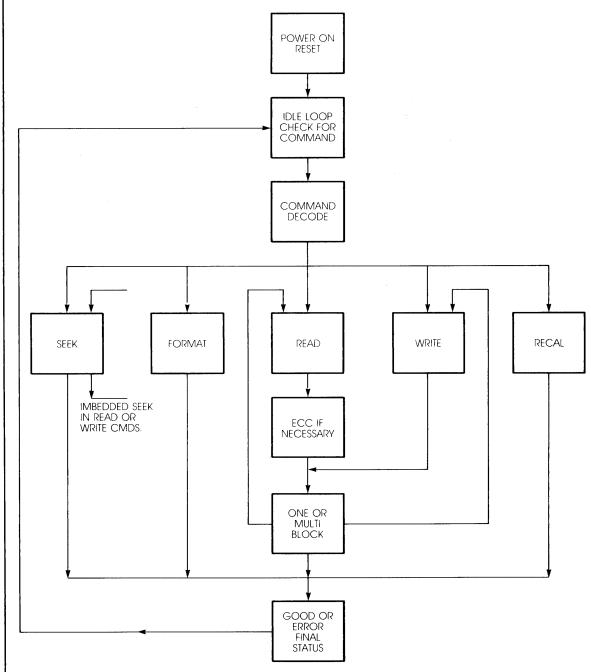
AIC-300 Read Disk



AIC-300 Write Disk

7.2 SAMPLE ROUTINES

In order to control the operation of the disk controller, the support processor has to execute firmware that loosely synchronizes the Adaptec chip set. This section gives sample routines necessary to accomplish some basic functions. These can be modified to meet the specific design requirement. The following flowchart gives an overview of the disk control operation.



Disk Controller Operation Overview

The following is a breakdown of the main functional sections in the disk controller operation.

- Power On Reset: Initialize all known parameters for both host and drive interfaces. Clear or set appropriate flags to indicate unknown parameters (i.e., dirve not formatted). Some initial command may be necessary from the host in order to fully specify all drive or host parameters.
- Idle Loop: Constantly check the host interface for incoming commands. Do any other housekeeping or checking necessary to remain active and alert to the host or drive interface. Seek complete on one of the drives may be checked here.
- Command Decode: Check for incoming commands from the host for correct formats, lengths, and zero check unused fields. Save the host I.D. and the logical unit number of the active unit. Possibly separate command types here and do the seek, if necessary. Jump to the routine to execute the command.
- The Commands: Start and stop command execution, ensure data transfers to or from the host, check I.D. and data fields for validity, and maintain proper positioning on the drive. Please refer to the sample routine listings that follow this text for guidelines.
- Final Status: Report to the host upon completion of each command. Report proper completion or error conditions. Clean up host and drive interfaces, internal flags, etc., and return to the idle loop.

Some typical error conditions are as follows:

Drive Errors	Controller Errors	System Error
NO INDEX SIGNAL NO SEEK COMPLETE WRITE FAULT DRIVE NOT READY TRACK 00 NOT FOUND	I.D. ERROR UNCORRECTABLE DATA ERROR I.D. ADDRESS MARK NOT FOUND DATA ADDRESS MARK NOT FOUND RECORD NOT FOUND SEEK ERROR CORRECTABLE DATA ERROR INTERLEAVE ERROR UNFORMATTED DRIVE SELF TEST ERROR DEFECTIVE TRACK	INVALID COMMAND ILLEGAL BLOCK ADDRESS VOLUME OVERFLOW BAD ARGUMENT INVALID LUN

Following are some sample routines for AIC-300 and AIC-100 chips. These routines show how to program normal disk functions. Error routines are left to the user to define. The drive interface defined here may also be different for some users. The chip registers are defined by their locations for easy understanding without having to refer to the equate list.

```
SYSTEM EQUATES
               BUFFER CHIP
                               ;DMA control register ;BUFFER capacity
R53
        EQU
                4Ø53H
R54
        EQU
                4054H
                                RESET
R59
        EQU
                4Ø59H
                405AH
                                ;RAP register
R5A
        EQU
                405CH
                                ;WAP register
R5C
        EOU
R5E
        EQU
                405EH
                                 ;STOP register
               SERDES CHIP
                                ;BUFFER I/O DATA
                4070H
R7Ø
        EQU
                                ;ECC control register ;ECC low byte
                4071H
R71
        EQU
R72
        EQU
                4072H
                                ;ECC high byte
;ECC poly byte FORWARD
;ECC poly byte RECIPROCAL
;BRANCH CONTROL register
R73
        EQU
                4073H
R74
        EQU
                4074H
R77
                4077H
        EQU
R78
                4078H
        EQU
                                ;START register
R79
        EQU
                4079H
                                ;OPERATION CONTROL register ;PUSH/POP I.D. stack/CLOCK DIVIDE
R7A
        EQU
                407AH
R7F
        EQU
                407FH
                                ;CYLINDER area
        EQU
                4ØEØH
REØ
                                ;HEAD area
                4ØE1H
RE1
        EQU
                                ;SECTOR area
RE2
        EQU
                40E2H
                                ;FLAG area
;DATA field length
RE3
        EQU
                4ØE3H
                40C4H
RC4
        EQU
                                ;SEARCH OPERATION cell
;GAP length cell
        EQU
                40A4H
RA4
RDØ
        EOU
                40D0H
                                ; ECC HARDWARE OFFSET
               7
ECCOFF EQU
RESET BUFFER AND SERDES CHIPS
*************
                         disable interrupts;
RESET: DI
                                ;reset serdes
;send to serdes
;RESET OFF
;send to serdes
                A,20H
        MVI
               R71
        STA
                A,ØØH
        MVI
                R71
        STA
                                ;and again
        STA
                R71
        MVI
                A,1
                                ;reset...
                                ;BUFFER CHIP
        STA
                R59
                                 ; RESET OFF
        XRA
                Α
                R59
        STA
                A,42H
        MVT
        OUT
                Ø8ØH
                                ; INITIALIZE PORTS
JMP IDLE ;GO TO IDLE LOOP
           FORMAT ENTIRE DRIVE ROUTINE
**********
FORMAT: MVI B,00 ;clear flag
        MVI
                A,40H
                                ; ENABLE INITIATOR
        STA
               R53
                                ;SELECT THE DRIVE
                GETDRV
        CALL
                                ;recal drive
        CALL
                RECAL
CLRLA:
        XRA
                Α
                                ;zero out
                                ;cylinder...
                REØ
        STA
                REl
                                ;head...
        STA
                                 ;sector...
        STA
                RE2
                                ;and flag
                RE3
        STA
        MVI
                A,ØFFH
                                ;SET 256 BYTE SECTORS
        STA
                RC4
                                ;supress xfer on
                A,20H
        MVI
                                 ;send to op control reg
        STA
                R7A
                                ;32 sectors per track
;gap 1
;store in serdes
                C,32
NEWTRK: MVI
        MVI
                A,ØEH
        STA
                RDØ
                                ;set up value...;for branch register
        MVI
                A,10H
        STA
                R78
                               ;point H/L at start register ;write start value ;get status
                H,R79
        LXI
        MVT
                M,15H
BRACTØ: MOV
                A,M
        ANI
                20H
                                ;test branch active bit
;loop til branch active OR INDEX
                BRACTØ
        JΖ
                                ;WRITE I.D. branch value
        MVI
                A, 11H
                                 ;save in branch register
                R78
        STA
                 40H
                                 ;test xfer active
        ANI
```

```
JZ
                 XFACTØ
                                   ;loop til it is
                 A,ØCH
         MVI
                                   ;gap 3
         STA
                  RDØ
                                   ;save in serdes
         DCR
                                   ;decrement sector counter
                 LSTSEC
                                  ; if last sector, jump out ; otherwise, use 12...
         JΖ
         MVI
                  A,12H
         STA
                  R78
                                   ;as branch
                 INCLA
                                  GET NEXT SECTOR/STORE NEW C.H.S.F
         CALL
                                  ; and loop back to branch active check
         JMP
                  BRACTØ
LSTSEC: MVI
                                  ;stop value
;send to branch register
                  A,13H
         STA
                  R78
LSTWTØ. MOV
                  A.M
                                   ;get SERDES status
         ANI
                  1ØH
                                   ;test the stopped bit
    JZ LSTWTØ ;100p till stopped AT INDEX USERS MUST DEFINE THEIR OWN ROUTINES INCLA AND BUMP.
        INCLA IS INTERNAL HOUSEKEEPING ROUTINE TO KEEP TRACK
              OF CYLINDER AND HEAD UNTIL ENTIRE DISK IS FORMATTED.
              IT ALSO STORES NEW C, H, S, F IN REØ - RE3.
        BUMP WILL SELECT A NEW HEAD AND SEEK, WHEN NECESSARY.
                  INCLA
         CALL
         CALL
                  BUMP
         JNC
                  NEWTRK
                                  ; if not done, loop back
                                  ; if done, zero out...; supress bit
FMDONE: XRA
         STA
         JMP
                  IDLE
             IDLE ; and jump to idle loop
              READ ONE OR MULTIPLE BLOCKS ROUTINE
*****************
         XRA
                                  ;zero out...
         MOV
                 D.A
                                  CLEAR FOR MSB STOP POINTER
         STA
                 R59
                                  ; reset ALL POINTERS on BUFFER CHIP
         MVT
                 A,ØDØH
                                  ;set READ OPERATION
         STA
                 R53
                                  ; send to BUFFER CHIP
         CALL
                 GETDRV
                                  ;select drive
         LDA
                 NBLKS
                                  GET BLOCK COUNT
         MOV
                 C.A
                                   ;save in C
         LDA
                                  GET CYLINDER
                 CYL
         STA
                 REØ
                                  ; INTO SERDES
         STA
                 DESTRK
                                  ;save as desired track
         LDA
                                  GET HEAD
                 HEAD
         STA
                 RE1
                                  ; INTO SERDES
                 DESTHD
         STA
                                  ;save as desired head
         CALL
                 SEEK
                                  ;do the seek
         JC.
                 BADCMD
                                  ;on error, halt
                                  ;zero out..
         XRA
         STA
                 RE3
                                  ;FLAG
         LDA
                 SECT
                                  ;get sector
         STA
                 RE2
                                  ; INTO SERDES
RDRENT: MVI
                 A,08H
                                  ;stuff Ø8
         STA
                 R78
                                  ;into branch register
                 H,R79
         LXI
                                  ;point H/L at start register
        MVI
                 E,Ø3
                                  ;3 revs to find i.d.
                                  ;dummy read to clear index passed
;START SERDES
        LDA
                 R7A
RDSTRT: MVI
                 M,Ø5H
RDDLY: MOV
                 A,M
                                  ;get status
        MOV
                                  ;save a copy in B
                 B,A
        ANI
                 20H
                                  ; check branch active
        JNZ
                 RDDAM
                                  ;branch active = GOOD I.D.-READ DATA
        MOV
                 A,B
                                  ;recover status
        ANI
                 1 Ø H
                                  ;check stopped bit
                                  ;if not stopped, loop back up
;if stopped, get new copy
;look for ecc error
        JΖ
                 RDDLY
        MOV
                 A,B
Ø4H
        ANI
        JNZ
                 RDIDER
                                  ; if ecc blown, jump to error routine
                 R7A
        LDA
                                  ; if not ecc stop, get op ctl
        ANI
                 Ø1
                                  ;test index passed
        JΖ
                 RDSTRT
                                  ; if not passed, loop back
        DCR
                                  ; if passed, drop rev counter
                 E.
        JNZ
                 RDSTRT
                                  ;TRY AGAIN
        JMP
                 SEEKER
                                  ; if zero, ERROR
RDDAM:
        MVI
                 В,0
                                  CLEAR COUNTER
        ANA
        JM
                 RDXFLP
                                  ;TEST FOR DATA TRANSFER ACTIVE
        LDA
                 R79
        ANA
        JM
                 RDXFLP
        DCR
                 В
                                  ; INSURE NO HANG IF NO DATA A.M.
        JNZ
                 AMNACT
                                  ; WAIT FOR DATA TRANSFER
        JMP
                 DAMERR
                                  ; NO DATA A.M. FOUND
```

```
;USE Ø5
RDXFLP: MVI
                A,5
                                ;as branch
                R78
        STA
                                ;DECREMENT COUNTER
RDNXT:
        DCR
                C
                                ;if zero, get out ;GET NEXT SECTOR
        JΖ
                RDBR2
        CALL
                INCLA
                                get status;
RDBR2:
        MOV
                A,M
                                ;save a copy in B ;test branch active and stopped bits
        MOV
                B.A
        ANI
                30 H
                                ;loop till branch active or stopped
        JΖ
                RDBR2
        MVI
                A,08H
                                ;get the next branch address into A
                R78
                                ;and save in branch register
        STA
                                 ;now recover status
        MOV
                A,B
                                 ;test for ecc or compare error
;jump out on error
        ANI
                Ø 5 H
        JNZ
                RDDAER
                                 ;DATA IS GOOD
        LDA
                R53
                Ø8H
                                 ;CHECK READ LATCH
        ANI
                                 ; NOT ON - FIRST SECTOR
                RD4
        JZ
                R53
        T.DA
                                 ;CHECK DMA DONE
                20H
        ANI
                                 ;DMA IS DONE - SEND NEXT SECTOR
        JNZ
                RD2
        MVI
                1FH
        STA
                R79
                                 ;DMA NOT DONE - STOP SERDES
RD1:
                R53
        LDA
                20H
        ANI
                                 ;WAIT FOR DMA DONE ;BUMP IT
        JΖ
                RD1
        INR
                Α
                                 ; INSURE NOT OVER 3
        CPI
        JNZ
                RD3
                                 ;BACK TO Ø
        XRA
                Α
                                 ; SAVE IT
RD3:
        MOV
                D,A
                                 ;PUT IT IN STOP HIGH
        STA
                R5F
                A,ØFFH
RD4:
        MVI
                                 ;SET STOP LOW = FF
        STA
                R5E
                A, ØD8H
        MVI
                                 ;SET READ LATCH AND START DMA
        STA
                R53
                                 CHECK BLOCK COUNT
        MOV
                A,C
        ANA
                STOPRD
                                 :DONE
        JΖ
                                 GO READ NEXT SECTOR
        . MP
                RDRENT
                                 ;stop location..
STOPRD: MVI
                A,14H
                                 ;into branch register
                R78
        STA
                                 GET STATUS
                A.M
STOPBR: MOV
                                 ;look at stopped bit
        ANI
                10H
                STOPBR
                                ;loop till it is
        JΖ
                                 get status again;
        MOV
                A,M
                                 ;test ecc error bit
                Ø4H
        ANI
                                 ;if set, jump to ecc retry routine ;GET STATUS
        JNZ
                RDDAER
STOP1:
        LDA
                R53
        ANI
                20 H
                                 :WAIT FOR DMA DONE
        JΖ
                STOPl
        XRA
                                 ;TURN OFF READ LATCH
                R53
        STA
                                 ;STOP SERDES
        CALL
                KILSER
JMP IDLE ;otherwise, back to idle loop
             WRITE ONE OR MULTIPLE BLOCKS ROUTINE
************
WRITE: XRA
                                ;zero out...
                A
                                 ;CLEAR FOR MSB STOP POINTER
        MOV
                                BUFFER CHIP RESET ;select drive
        STA
                R59
                GETDRV
        CALL
                                GET BLOCK COUNT
        T.DA
                NBLKS
                                 ;save in C
        MOV
                C,A
                                GET CYLINDER
        LDA
                CYL
        STA
                REØ
                                 ; INTO SERDES
                                save as desired track
        STA
                DESTRK
                                 GET HEAD
                HEAD
        LDA
                                 ; INTO SERDES
                RE1
        STA
                                ;save as desired head
        STA
                DESTHD
        CALL
                 SEEK
                                 ;do the seek
                BADCMD
                                 ;on error, halt
        JC
        XRA
                                 ;zero out..
                 Α
                                 ;buffer pointers
        STA
                 R59
                                 ;zero out..
        XRA
                 Α
        STA
                 RE3
                                 ;FLAG
        LDA
                 SECT
                                 ;get sector
        STA
                 RE2
                                 ; INTO SERDES
                                 ;stuff 09 (start address)...
WRRENT: MVI
                 A, Ø9H
                                 ;into branch register
                 R78
        STA
```

```
LXI
                 H,R79
                                   ;point H/L at start register
        MVT
                 E,03
                                   ;3 revs to find i.d.
        LDA
                 R7A
                                   ;dummy read to clear index passed
        MOV
                                   GET MSB STOP POINTER VALUE
                 A,D
         INR
                 A
                                   ;BUMP IT
        CPI
                 4
                                   ; INSURE NOT OVER 3
        JNZ
                 WD1
        XRA
                 Α
                                   ;BACK TO Ø
WD1:
                                   ; SAVE VALUE
        MOV
                 D,A
        STA
                 R5F
                                   ; SET STOP POINTER HIGH
        MVI
                 A,ØFFH
        STA
                                   ;STOP POINTER LOW = FF
                 R5E
                 A,ØC4H
        MVI
        STA
                 R53
                                   ;SET WRITE LATCH - START DMA
WD2:
        LDA
                 R53
                                   GET STATUS
        ANI
                 20H
        J 7.
                 WD2
                                   ;WAIT FOR DMA DONE
WRSTRT: MVI
                 M,Ø5H
                                   ;DATA IS IN BUFFER - START SERDES
WRDLY:
        MOV
                 A,M
                                   ;get status
        MOV
                 B.A
                                   ;save a copy in B
        ANI
                 20H
                                   ;check branch active
        JNZ
                 WRDAM
                                   ; if branch active, jump
        MOV
                 A,B
                                   ;recover status
        ANI
                 10H
                                   ;check stopped bit
        JNZ
                 WRRENT
                                   ;if stopped, re-enter
        T.DA
                 R7A
                                   ; if not stopped, get op ctl
        ANI
                 Ø1
                                   ;test index passed
        JΖ
                 WRDLY
                                   ; if not passed, loop back
                                   ;if passed, drop rev counter;TRY AGAIN
        DCR
                 E
        JNZ
                 WRDLY
                                   ; if zero, find out why
        JMP
                 SEEKER
WRDAM:
        MOV
                 A,M
                                   ;get status
        MOV
                 B,A
                                   ;save a copy in B
        ANI
                                   ;stopped ??
                 10H
        JNZ
                 WRRENT
                                   ; if stopped, re-enter to re-start
        MOV
                 A,B
                                   ;recover status
        ANI
                 40H
                                   ;transfer active ??
        JΖ
                 WRDAM
                                   ; if no, loop back
WRXFLP: MVI
                 A, Ø5H
                                   ;use Ø5..
        STA
                 R78
                                   ;as branch
        XRA
                 Α
                                   ;zero out...
        STA
                 WRFLAG
                                   ;write flag
WRNXT:
        DCR
                                   ;decrement counter
        JΖ
                 STOPWR
                                   ; if zero, get out ;GET NEXT SECTOR
        CALL
                 INCLA
        MVI
                 D,09H
                                   ;next branch (09) into D to tighten code
WRBR2:
        MOV
                 A,M
                                   ;get status
        MOV
                 B,A
                                   ;save a copy in B
        ANI
                 30H
                                   ;test branch active and stoped bits
        JZ
                 WRBR2
                                   ;loop till branch active or stopped
        MOV
                 A,D
                                   ;get the next branch address into A
                                   ;and save in branch register
        STA
                 R78
        MOV
                                   ;now recover status
                 A,B
        ANI
                 10H
                                   ;check stopped bit
        JNZ
                 WRRENT
                                   ;re-enter if stopped
        JMP
                 WRDAM
                                   ;otherwise, loop back up
STOPWR: MVI
                                   ;STOP LOCATION
                 A,14H
        STA
                 R78
                                   ;into branch register
WRSTBR: MOV
                 A.M
                                   ;get status
        ANI
                 1 ØH
                                   ;look at stopped bit
        JΖ
                 WRSTBR
                                   ;loop till it is
        XRA
        STA
                 R53
                                   ;TURN OFF WRITE LATCH
        CALL
                 KILSER
                                  STOP SERDES
        JMP
                 IDLE
                                   ;otherwise, back to idle loop
```

```
RECAL ROUTINE DOES A ONE-STEP-AT-
       A-TIME RECAL WHILE LOOKING FOR THE
       TRACK Ø SIGNAL.
* NOTE: USERS INTERFACE MAY BE DIFFERENT THAN
       SHOWN BELOW FOR PIN ASSIGNMENTS FOR
* STATUS, SEEK PULSES AND SEEK DIRECTION. *
                               ;temp save A/PSW
RECAL: PUSH
               PSW
                               ;select drive 1, direction=out
       MVI
               A,ØlH
       OUT
               82H
                               ;send to port
RDYCK1: IN
               83H
                              get status;
               Ø8H
       ANI
                               ;test ready
                               ;loop till ready
               RDYCK1
       JΖ
TRØØCK: IN
               83H
                               get status;
       ANI
               Ø2H
                               ;test trk Ø
       JNZ
               RETØ
                              ;if track 0, get out
       MVI
               A,41H
                               ;step pulse
                               ;send it
       OUT
               82H
               A,Ø1H
                              ;un-step
       MVI
       OUT
               82H
                               ;send it
SKCPLP: IN
               83H
                               get drive status
       ANI
               ØlH
                               ;look at seek complete
               SKCPLP
                               ;loop till seek complete
       JZ
                               ;then check for track 00
               TRØØCK
       JMP
RETA:
                               ;zero out
       XRA
        STA
               CURTRK
                               ;current track
       POP
               PSW
                               ;recover A/PSW
                               ;return
*****************
       SEEK TO TRACK AND HEAD IN DESTRE AND DESTHD,
       WHICH CONTAIN CYLINDER AND HEAD, RESPECTIVELY.
       USER TO SET THESE UP FROM COMMAND DATA.
*************
               PSW
       PUSH
                               ;temp save A/PSW
SEEK:
                               ;temp save B/C
       PUSH
               В
               CURTRK
       LDA
                               get current track;
       MOV
               B,A
                               ;save in B
                              get destination track
       LDA
               DESTRK
       MOV
               C,A
                               ;save in C
                              ;compare them
       CMP
                              ; if they are the same, jump around
               DOHEAD
        JΖ
                               ; if desired is less than current, jump
        JC
               GOOUT
GOIN:
       MOV
               A,C
                              get destination track;
        SUB
               В
                               ;subtract smaller current track
               C,A
                              ;put result in C
       MOV
                              ;get control port
;set dir in
               82H
        IN
        ORT
               8 Ø H
               DOSEEK
        JMP
                              ;and jump to common code
                               ;get current track
GOOUT:
       MOV
               A,B
                              ;subtract smaller destination track
        SUB
                              ;put result in C
       MOV
               C,A
               82H
                               ;get control port
       ΙN
                               ;set dir out
       ANT
               7FH
DOSEEK: OUT
               82H
                               ;set up direction
SEEKLP: ORI
               4ØH
                               ;set step pulse
                              ;send it
       OUT
               82H
       ANI
               ØBFH
                               ;un-step
                               ;send it
       OUT
               82H
WTSKC1: IN
               83H
                               ;get drive status
                               ;look at seek complete
               ØlH
        ANI
               WTSKCl
                               ;loop till seek complete
        .77
       DCR
               С
                               ;drop seek counter
        JNZ
               SEEKLP
                               ;loop till C=Ø
       LDA
                               ;get derired track
               DESTRK
                               ;save as current track
               CURTRK
       STA
    NOTE: USERS INTERFACE MAY BE DIFFERENT THAN SHOWN BELOW
          FOR DRIVE AND HEAD SELECTION.
                               ;get desired head
DOHEAD: LDA
                               ;shift left...
       RAL
                               ;to put head number...
       RAL
        RAL
                               ;in correct position...
                               ;to send to I/O port
        RAL
        ANI
               3ØH
                               ;zap any garbage
                               ;save in B
       MOV
               B,A
                               ;get current control port
       ΤN
               82H
        ANI
               ØCFH
                               ;clear head select
```

```
;OR in head
          ORA
                    82H
          OUT
                                        ;output result
          POP
                    PSW
                                        ;recover A/PSW
                     ; and return
                     SELECT DRIVE ROUTINE
                     *******
                                 ;temp save A/PSW
GETDRV: PUSH PSW
                                      ;get current port value
;zap select bits
;select drive
;send it out
;recover A/PSW
                    82H
          IN
          ANI
                    ØFCH
          ORI
                    Øl
          OUT
                    82H
          POP
                    PSW
          RET
                                         ; and return
       ***********
          STOPS THE SERDES AND THEN RETURNS
***********
KILSER: PUSH PSW ;temp save A/PSW
MVI A,1FH ;then load the stop location..
STA R79 ;to the start register to halt serdes
HLCKLP: LDA R79 ;get serdes status
ANI 10H ;check the stopped bit
JZ HLCKLP ;loop till halted
POP PSW ;if stopped, recover A/PSW
          RET
                                        ;and then return
  ****************
                    ROUTINE READS THE SYNDROME INTO ECC1
***************
                               ;DISABLE FEEDBACK
;TO SERDES
;point H/L at serdes ecc register
;point D/E at ram to save syndrome
;get msb of syndrome
;save in ram
RDSYND: MVI A,04H
          STA
                    R71
          LXI
                    H,R73
          LXI
                   D,ECC1
          MOV
                    A,M
                                       ;save in ram ;bump ram pointer
          STAX
                    D
          INX
                    D
                                       three more bytes to do; 8 bits per byte
                    C,03
          MVI
BYTLOP: MVI
                    B,08
                                      ;8 bits per byte
;inhibit feedback and shift value
;force shift of syndrome
;subtract 1 from bit counter
;loop 8 times (1 byte)
;load the next byte of the syndrome
;save it in ram
;and bump ram pointer
;drop byte count
          IVM
                    A,06H
SHFT:
          STA
                    R71
          DCR
                    В
          JNZ
                    SHFT
          MOV
                    A,M
          STAX
                    D
          INX
                    D
                                      drop byte count
          DCR
                    C
     JNZ BYTLOP ;loop 3 times
          ATTEMPT TO CORRECT ECC ERROR
*****************
                 A, 08H ;CLEAR ECC COMMAND
FIXIT: MVI
                                       ;TO SERDES ;RESET OFF
          STA
                    R71
          IVM
                   A,04H
          STA
                    R71
                                      ;H/L = FIRST SYNDROME BYTE
          LXI
                    H,ECC1+3
                                       ; TO ENTER 4 BYTES
                    C,04
          MVI
                                       ;8 BITS PER BYTE ;GET BYTE TO LOAD
NXTBYT: MVI
                    B,08
          MOV
                    D,M
                                       ;BYTE TO LOAD IN A ;LSB BIT TO CARRY
SHFBYT: MOV
                    A,D
          RAR
                                        ; SAVE SHIFTED BYTE
                                      ;SAVE SHIFTED BYTE
;LSB BIT TO BIT Ø
;ONLY ONE BIT
;ADD SHIFT COMMAND
;STUFF BIT VIA ECC CONTROL REG
;DECREMENT BIT COUNT
;LOOP 8 TIMES
;TO NEXT HIGHER SYNDROME BYTE
;DECREMENT BYTE COUNT
;LOOP 4 TIMES
;POINT TO REVERSE BOLVNOWIELD
          MOV
                    D,A
          RAL
          ANI
                    ø٦
          ORI
                    Ø6H
          STA
                    R71
          DCR
                    В
          JNZ
                    SHFBYT
          DCX
                    Н
          DCR
                    NXTBYT
          JNZ
                                       ; POINT TO REVERSE POLYNOMIAL
STFRCP: LXI
                    H, REVERS
                                       ; INTO POLYNOMIAL REGISTERS
;GET CURRENT BLOCK SIZE
          CALL
                    LD POLY
FIGLEN: LHLD
                    ABLKSZ
                                       ; MOVE POINTER
; INTO B/C
          PUSH
                    н
          POP
                    В
                                       ; NUMBER OF BITS PER BYTE
; NUMBER OF BITS PER BLOCK
          MVI
                    A,08
                    MULT
          CALL
                                        ;4 ECC + 2 GAP BYTES TIMES 8
                    H,3CH
          LXT.
          DAD
                    В
                                         ;H/L = TOTAL BITS PER BLOCK
```

```
SHLD
                 TEMP
                                  ; SAVE TOTAL BIT COUNT
         XCHG
                                  ; PUT IN D/E
         MVI
                 A,00
                                  ;DISABLE ECC SHIFTS
         STA
                                  ;TO ECC CONTROL REG
                 R71
                                  ;H/L = ERROR TEST REG
        T.X.T
                 H,R72
SHFTST: MVI
                 A,Ø2H
                                 ;TO SHIFT ECC
         STA
                 R71
                                  ;TO CONTROL REG
         DCX
                                  ;DECREMENT BIT COUNT
        MOV
                                  ;GET ERROR INDICATOR
                 A,M
        ANA
                                  ;TEST FOR ZERO
                 GOTDIS
                                 ;Ø = ERROR FOUND
         J7.
        MOV
                 A,E
                                  ;LSB BIT COUNTER
         ORA
                                  ;CHECK FOR ZERO
                                  ; NON ZERO = SHIFT AGAIN
; DONE HERE = HARD ERROR
         JNZ
                 SHFTST
        JMP
                 ECCERR
GOTDIS: CALL
                                  ;D/E = MINUS COUNT
                 COMPD
        LHLD
                 TEMP
                                  ;H/L = TOTAL BIT COUNT
        DAD
                                  ;TOTAL BITS MINUS COUNT DOWN
                                  ; ECC HARDWARE OFFSET
        LXI
                 D,-ECCOFF
        DAD
                                  ;SUBTRACT HARDWARE OFFSET
                 D
        LXI
                 D,-32
                                  ;FIRST 32 = ECC OR GAP
        DAD
                 D
                                  ;SUBTRACT 32
         JNC
                 FIXED
                                  ; ERROR IN ECC = DATA O.K.
        SHLD
                 TEMP1
                                  ;SAVE COUNT
        MOV
                                  ;TO DIVIDE BYTE ADDRESS BY 8
                 в,03
RR3LOP: MOV
                 A,H
                                  ;HIGH BYTE OF BIT COUNT
        ANA
                 Α
                                  ;CLEAR CARRY
        RAR
                                  ;LSB BIT TO CARRY
        MOV
                 H,A
                                  ; SAVE ROTATED VALUE
                                  GET LOW BYTE
        MOV
                 A,L
                                  ;CARRY TO BIT 7
        RAR
        MOV
                 L,A
                                  ;SAVE SHIFTED LOW BYTE
        DCR
                                 ;DECREMENT SHIFT COUNTER
        JNZ
                 RR3LOP
                                  ;LOOP 3 TIMES
                                  ;HIGH BYTE OF SHIFTED ADDRESS
        MOV
                 A,H
                                  ;CLEAR 3 HIGH BITS
;SAVE IT IN H
        ANI
                 1FH
        MOV
                 H,A
        XCHG
                                  ;SHIFTED ADDRESS TO D/E
        CALL
                 COMPD
                                  ;D/E = MINUS ADDRESS
                                  ;UPPER WRITE LIMIT POINTER (WAP)
        LHLD
                 R5C
                                  ;LAST DATA BYTE
        DCX
                 Н
        DAD
                 D
                                  ;SUBTRACT POINTER FROM TOP OF DATA
CORECT: LDA
                 R73
                                  GET ERROR MASK DATA
        MOV
                 D,A
                                  ;D = ERROR MASK
                                  CLEAR E
        MVI
                 E,Ø
                                  ;BIT COUNT = ERROR OFFSET IN 2 BYTES
        LDA
                 TEMPl
                                  ;ONLY 3 BITS
        ANI
                 Ø7
        INR
                                 ;PLUS ONE = SHIFT COUNT
                 Α
                                 ;B = SHIFT COUNT
        MOV
                 B,A
RRXLP:
        DCR
                                 ;DECREMENT SHIFT COUNT
        JZ
                 MASKOK
                                 ;0 = MASK SHIFTED TO PROPER BITS
                                  GET HIGH BYTE OF MASK
        MOV
                 A,D
                                  ;CLEAR CARRY
        ANA
                 Α
        RAR
                                  ;BIT ZERO TO CARRY
        MOV
                 D,A
                                 ;RESTORE SHIFTED COPY
        MOV
                                  GET LOW BYTE
                 A,E
        RAR
                                 ;CARRY TO BIT 7
        MOV
                E,A
                                 RESTORE LOW BYTE
                                 RESTORE HIGH BYTE
        MOV
                A,D
        ANT
                Ø7FH
                                 ;CLEAR BIT 7
        MOV
                D,A
                                 ;SAVE HIGH BYTE
        JMP
                RRXLP
                                 ;LOOP AGAIN
MASKOK: MOV
                A,E
                                 GET LOW BYTE OF MASK
                                 ;SWAP END FOR END
        CALL
                MIRROR
        MOV
                                 ;SAVE IN E
                E,A
        MOV
                A.D
                                 ;GET HIGH BYTE
        CALL
                MIRROR
                                 ;SWAP END FOR END
        VOM
                D,A
                                 ;SAVE IN D
        LDA
                R53
                                 CURRENT DMA CONTROL BYTE
        ANI
                 ØF7H
                                 ;KILL READ LATCH
        STA
                R53
        PUSH
                Н
                                 ;SAVE ERROR POINTER
        LHLD
                R5C
                                 GET CURRENT WAP
                                 ;SAVE IN MEMORY
        SHLD
                 TEMP
                                 ; RESTORE ERROR POINTER
        POP
                H
                R5C
        SHLD
                                 :PUT IN WAP
        LDA
                R7Ø
                                 ;GET DATA FROM BUFFER
        XRA
                E
                                 ;CORRECT LOW BYTE
                R7Ø
                                 ;CORRECTED BYTE TO BUFFER
        STA
        INX
                                  ; POINTER TO HIGH BYTE
```

```
; PUT IN WAP ;GET DATA FROM BUFFER
        SHLD
               R5C
       LDA
               R7Ø
                             CORRECT HIGH BYTE
        XRA
               ח
               R7Ø
        STA
                              ; CORRECTED BYTE TO BUFFER
        LHLD
               TEMP
                              ; RECOVER SAVED WAP
                               ; RESTORE WAP
       SHLD
               R5C
               F,FORWRD
FIXED: LXI
                              ;H/L = FORWARD POLYNOMIAL
               LDPOLY
                              ; RELOAD FORWARD POLYNOMIAL
       CALL
       MVI
               A,00
                              ;STANDARD ECC COMMAND
        STA
                              ; TO ECC CONTROL REG
       RET
     **************
        LOAD ECC POLYNOMIAL
*************
             PSW ; H/L = POLY CONSTANT
B ; SAVE B/C
LDPOLY: PUSH
              В
       PUSH
        PUSH
              D
                             ; SAVE D/E
               B,04
       MVI
                             ;TO LOAD 4 BYTES
;D/E = FIRST POLY REG
;LOAD POLYNOMIAL
               D,R74
       LXI
        CALL
               MOVBYT
        POP
        POP
               В
        POP
               PSW
        RET
     ********
        COMPLEMENT THE D REGISTER
COMPD: PUSH
             PSW
        VOM
               A,D
        CMA
        MOV
               D,A
       MOV
               A,E
        TNX
               D
                              ;TWO'S COMPLEMENT
        POP
               PSW
        RET
       *************
        MULTIPLY A BY B/C
***************
MULT:
       PUSH
             Н
        ANA
               A
                              ;CLEAR CARRY
                             ;SHIFT MULTIPLIER
;GO IF NO BIT
SHFTA:
       RAR
        LNC
               SHFTB
       DAD
                              ; ACCUMULATE FINAL VALUE
               В
        RC
                              ; RETURN IF OVERFLOW
SHFTB:
       PUSH
               PSW
                              ; SAVE MULTIPLIER
       MOV
               A,C
                             ;SHIFT MULTIPLICAND
        RAL
        MOV
               C,A
       MOV
               A,B
        RAL
        MOV
               B,A
        POP
               PSW
                              ; RESTORE MULTIPLIER
                             CHECK IF DONE CONTINUE IF NOT
        ANA
               Α .
               SHFTA
        JNZ
        VOM
               B,H
                              ; RESULTS TO B/C
        MOV
        RET
    ;SAVE H/L
MIRROR: PUSH
               Н
                             ;SAVE D/E
;BIT COUNT = 8
       PUSH
               D
       MVI
               н,8
                             START VALUE INTO D
       MOV
               D,A
MIRLOP: MOV
               A,D
                              GET INITIAL DATA
       RAR
                              ;BIT Ø TO CARRY
       MOV
                              ; SAVE SHIFTED VALUE
               D,A
               A,E ; GET RESULTING VALUE

; LSB BIT INTO LOW END SHIFTING HIGHER

E,A ; SAVE RESULTS IN E

H ; DECREMENT BIT COUNTER

MIRLOP ; CONTINUE UNTIL DONE

A,E ; PUT RESULTS IN A
       MOV
       RAL
       MOV
       DCR
       JNZ
       MOV
       POP
               D
       POP
               Н
       RET
```

FORWRD: REVERS:		00,00,00,00 00,00,00,00	;FORWARD ECC POLYNOMIAL ;REVERSE ECC POLYNOMIAL
*		RAM ASSIG	**************************************
CURTRK:	DB	ØØH	;current track
DESTRK:	DB	ØØН	;destination track
DESTHD:	DB	ØØН	;destination head
ERRTYP:	DB	ØØН	;error type
RDFLAG:	DB	ØØН	;read flag
WRFLAG:	DB	ØØН	;write flag
CYL:	DB	ØØН	;INPUT CYLINDER
HEAD:	DB	ØØH	;INPUT HEAD
SECT:	DB	ØØН	;INPUT SECTOR
NBLKS:	DB	ØØH	;INPUT NUMBER OF BLOCKS
TEMP:	DB	Ø ØН	;TEMPORARY STOGAGE
TEMP1:	DB	ØØН	;TEMPORARY STORAGE
ECCl:	DS	4	;ECC location
ABLKSZ:	DS	2	;BLOCK SIZE
	END		

7.3 PROGRAMMING 8-BIT ECC CORRECTION

After each read date operation, a read error may have occurred. This may be determined by reading Register 79. If bit 2 is set, an error did occur and the following procedure is employed to determine if the error is correctable. Note that the majority of read errors are soft (i.e., caused by noise) and that the correction algorithm is time consuming. It is recommended that the record be re-read before attempting correction.

The general flow of the algorithm for 8-bit correction is as follows:

- 1. Off-load the 32-bit syndrome into local RAM.
- 2. Shift the syndrome back into the ECC register in reverse order, swapping the syndrome end for end.
- 3. Change the ECC polynomial from forward to reciprocal.
- 4. Shift the ECC until all bits except the high order (24-31) bits are zero (correctable) or the number of shifts are greater than the number of bits in the record (uncorrectable).
- 5. If correctable, the number of shifts represent the displacement of the error from the end of the record (the last bit of the ECC). The error pattern is located in bits 24-31 of the ECC register. This pattern is exclusive ORed with the appropriate bits in memory to correct the error.

Detailed Programming Steps

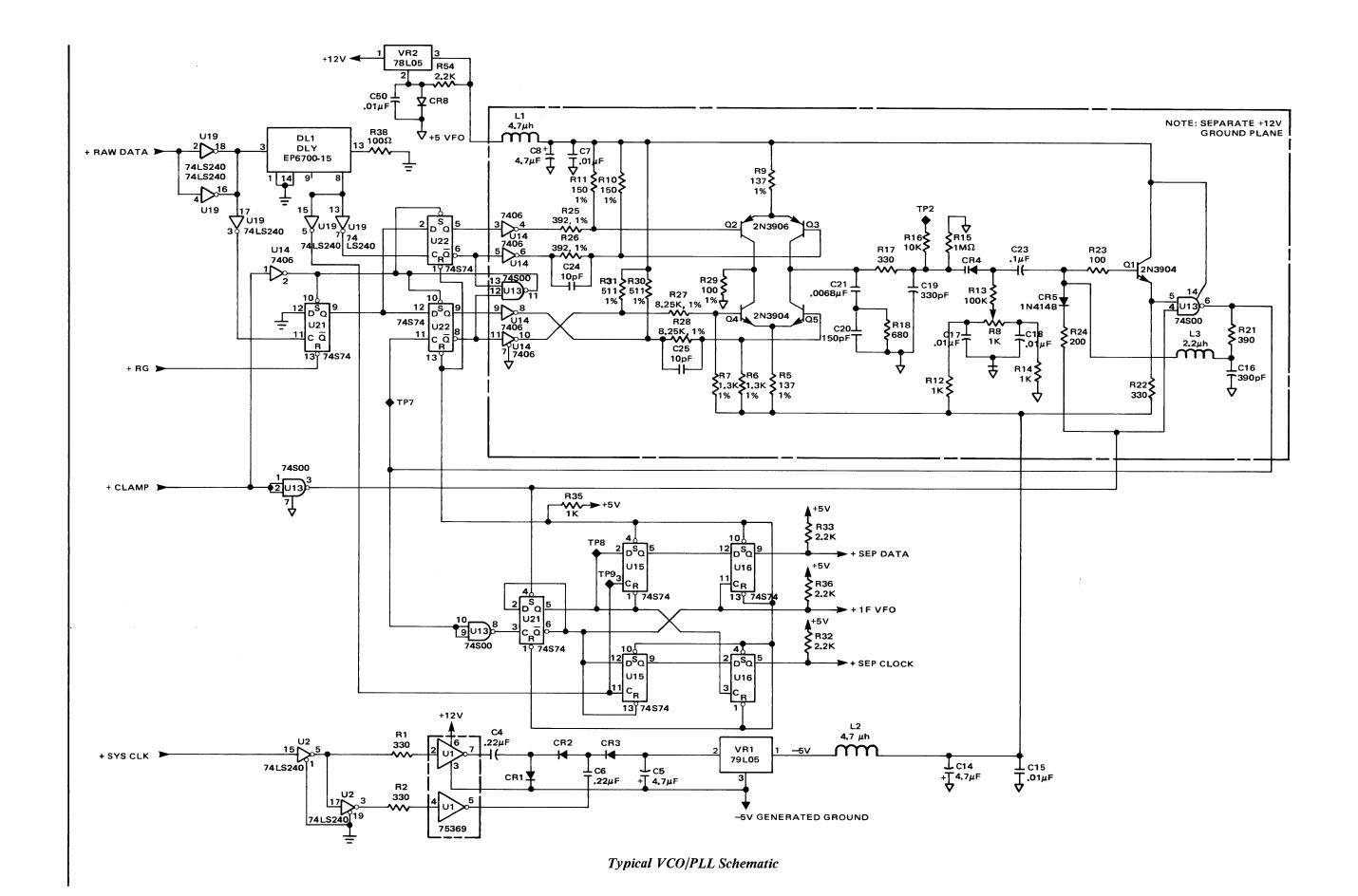
- 1. After a read error is detected, disable feedback by setting R71 = 04_H.
- 2. Store contents of R73 in RAM (x).
- 3. Shift ECC 8 times by setting $R71 = 06_H$ eight times.
- 4. Store contents of R73 in RAM (x + 1).
- 5. Shift ECC 8 times by setting $R71 = 06_H$ eight times.
- 6. Store contents of R73 in RAM (x + 2).
- 7. Shift ECC 8 times by setting $R71 = 06_H$ eight times.
- 8. Store contents of R73 in RAM (x + 3).
- 9. Clear ECC and disable feedback by setting R71 to 08 and then 04.
- 10. Right rotate location RAM (x + 3) and test if carry is set (i.e., test bit 0): If set, then load R71 = 07_H.
 If not set, then load R71 = 06_H.
 Repeat operation 7 more times to load entire byte.
- 11. Repeat step 10 for RAM locations x + 2, x + 1, and x until all 32 bits of the syndrome are loaded into the ECC in reverse order.
- 12. Load $R74 = 00_H$ and $R77 = 01_H$ to enable the reciprocal polynomial and disable the forward polynomial.
- 13. Compute record length in bits:
 # of bits per data field = ECC + Data + AM and SYNC
 For a 256 byte record length in bits = 4 * 8 + 256 * 8 + 2 * 8 = 2096
- 14. Enable feedback by setting $R71 = 00_H$.
- 15. Shift ECC once by setting $R71 = 02_H$ and increment a software counter.

- 16. Test to see if the software counter is greater than the record length: If yes, the error is uncorrectable. Re-enable the forward polynomial and end operation.
- 17. Test to see if R72 = 00_H: If yes, go to step 18. If no, go to step 15.
- 18. Subtract hardware offset of 7 from the shift count. If a correctable error is located within the ECC or the SYNC and AM bytes (the shift count < = 32), the data field is good and no further action is required. Subtract 32 from the shift count.
- 19. The bit displacement (shift count) must now be converted to a byte offset by right shifting the count 3 times. The value of the shift count equals the bit displacement from end of the record.
- 20. R73 is the mirror image of the error pattern. Form the error mask data (2 bytes) by R73 concatinating with a zero byte.
- 21. Get the shift count (E#) for error mask data by extracting the lower 3 bits from the shift count obtained in step 18.
- 22. Right shift the error mask data with MSB (bit 15) set to zero. Repeat E#-1 times more.
- 23. Mirror the error mask data byte by byte.

7.4 TYPICAL VCO/PLL SCHEMATIC (DATA SEPARATOR CIRCUIT)

In this section, a typical data separator circuit is shown. This circuit is used in the data encode/decode portion of the disk controller design. During a read operation, the AIC-250 chip receives data in the MFM format. In this format, both clock and data information are encoded together. The data separator is used to obtain separate data and separate clock, from the MFM encoded raw data.

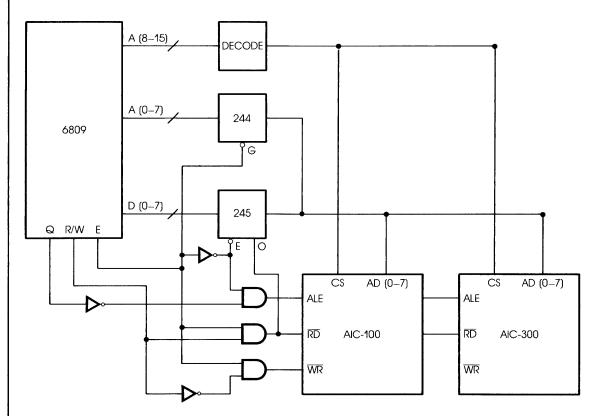
This schematic is commonly used by Adaptec and other disk controller manufacturers. The user has the choice of using this or any other design.



OTHER MICROPROCESSOR INTERFACES

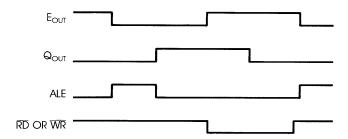
The Adaptec chip set is intended for use in a system with a multiplexed address/data bus, such as is found in the Intel 8085 family of microprocessors. The NSC 800 can also be very easily used since, in the Z80 emulation mode, the external interface is through an Intel 8085 type multiplexed address/data bus.

Other microprocessors, however, can be easily adapted for use in the disk controller design as the support processor. The basic design involves multiplexing the address and data busses, through external latches, and generating an Address Latch Enable signal (ALE). In addition to this, a separate read and write control signal is required. The following schematic shows the use of a Motorola MC6809 as the support processor in the design of the disk controller.



MC6809 Based Support Processor Interface

In the case of the 6809, as can be seen from the figure, the E_{OUT} and Q_{OUT} signals are used to generate the ALE signal. The generation of RD and WR is fairly simple also. The timing relationships for these signals are shown below.



MC6809 Interface Timing Relationship

In the case of a Z80 microprocessor, the interface is fairly similar. The lower order address and data lines have to be multiplexed in a similar fashion. However, separate RD and WR signals are already available, coming out of the processor chip. The ALE signal can be derived from the IORQ signal which is generated by the Z80 during input or output cycles (IN or OUT class of instructions).