Notes on the Altos 586 Computer

Work in progress, November 2023
by Lubomir Rintel

I’ve recently obtained the system board from the Altos 586 computer. Unfortunately, it came without documentation, the hard drive or the controller board.

With the goal of putting the system to some use I’ve decided to examine the hardware and firmware, learn how the system is put together and perhaps eventually substitute the missing parts.

This document contains some notes from my investigation, provided in hope of being useful to other owners of Altos 586 systems.

1. Altos 586 vs. 586T

There seems to be some documentation available for a related system, Altos 586T. While there’s no service manuals or schematics, there’s a reference manual[1] that describes many aspects of the computer architecture and firmware itnerfaces.

The Altos 586 system seems to be largely similar to Altos 586T. The "T" in the latter system’s designation seems to stand for "Tape", as, unlike the 586, 586T includes a tape drive.

The disk controller board used in 586T seems to be vastly different from the controller board used in the Altos 586. While the Altos 586 board is bases on around the 8089 I/O controller and only controls a hard drive, the 586T board includes a Floppy and Tape controllers and utilizes a Z80 processor to handle the I/O.

Considering the above, the 586T manual is largely relevant to the 586, except for when it comes to disk I/O, be it floppy access or hard drive access.

[1] *586T/986T System Reference Manual*, P/N 690-15813-002, April 1985

2. Z80 Peripheral Controller

As on 586T, the peripheral controller is a Z80 controlling a bunch of serial interfaces, timers for baud rate generators and a RTC. It’s equipped with a PIO for generating utility signal lines.

On a 586 it also has an extra duty of controlling the floppy drives, described below.

2.1. Peripherals

The peripherals are mostly the same as in the 586T machine aside from the addition of the Floppy Controller and the related circuitry.

Consult the *586T/986T System Reference Manual* for description of the peripherals common to 586 and 586T.

2.2. I/O Port Map

Address	Name	Part	Description
00h			Bus Address Top Bits Latch
20h – 23h	PIT 0	Intel 8254	Programmable Interval Timer
24h – 24h	PIT 1	Intel 8254	Programmable Interval Timer
28h – 2Bh	SIO 0	Zilog Z8440APS (Z80A SIO/0)	Serial I/O Controller
2Ch – 2Fh	SIO 1	Zilog Z8440APS (Z80A SIO/0)	Serial I/O Controller
30h – 33h	SIO 2	Zilog Z8440APS (Z80A SIO/0)	Serial I/O Controller
34h – 37h	PIO	Zilog Z8420APS (Z80A PIO)	Parallel I/O Controller
38h – 3Bh	FDC	Western Digital FDC1797	Floppy Disk Controller/Formatter
3Ch	DMA	Zilog Z8410APS (Z80A DMA)	Direct Memory Access Controller
40h			DMA Carrier/Parity
80h – 9Fh	RTC	MM58167AN	Microprocessor Real Time Clock

2.3. Floppy Disk Controller/Formatter

The floppy disk controller is the FDC1797 part hooked on in a straightforward manner. The output control signals are interfaced to a standard Shuggart floppy connector via open-collector drivers, the input signals go through a line drivers with external pull up register array. The read data signal is passed through external data separator circuitry to generate clock and data from MFM encoded signal.

Some of the FDC lines involved in the Floppy Drive access are connected to the PIO. In particular, the *#DDEN* pin needs to be set low in order to read double-density (MFM) disks, which is almost certainly the case. The *INTRQ* pin is connected to PIO’s *PA7* input pin. No idea if it’s actually used.

To actually access the floppy drive, the appropriate *Drive Select* signal needs to be pulled low. For reading double-sided floppies, the drive head/side is selected by the *Side Select* signal. The *Drive Select* and *Side Select* signals are also both controlled by the PIO.

For the details about the control lines consult the paragraph on the Parallel I/O Controller below.

2.4. Parallel I/O Controller

For the most part, the pins I've traced out are related to the floppy circuitry. Others are unknown.

#	Pin	Direction	Description
15	PA0		
14	PA1		
13	PA2		
12	PA3		
10	PA4		
9	PA5		
8	PA6		
7	PA7	Input	FDC INTRQ (Pin 39)
27	PB0	Output	"586 sync" sub-board (Pin 7)
28	PB1	Output	"586 sync" sub-board (Pin 8)
29	PB2	Input	Goes into unpopulated connector next to FDD (Pin 10)
30	PB3	Output	Floppy Drive Select 0
31	PB4	Output	Floppy Drive Select 1
32	PB5	Output	Floppy Side Select
33	PB6	Output	FDC #DDEN (Pin 37)
34	PB7	Input	PIT ? OUT2 (Pin 17)

3. Floppy Interface

3.1. Geometry and Format

The Altos 586 computer uses 5¼" diskettes in a drive with following parameters:

Data density	Double Density
Encoding	MFM
Speed	300 RPM
Bit rate	250 kbps
Tracks	80
Heads/Sides	2

The floppies XENIX, CP/M and the firmware use the following format (though the controller permits other formats).

Sector size	512 bytes
Sectors per track	9

Note that while this format is fairly unusual for 5¼" drives and media, it's exactly the same as IBM PC 3½" DD (720 KB) disks use. Regular 3½" drives can be used if they're adapted for the Altos 586 floppy connector pinout (see below).

The regular 1.44 MB floppies can usually be used with DD format, but the density select window needs to be covered with a piece of opaque tape.

3.2. Floppy Connector Pinout

The floppy connector is not PC compatible and therefore usual PC floppy drives can't be used without modifications. It uses straight ribbon cables, not the PC-style ones with a twist.

The pinout (and drive geometry) is fairly close to to what Commodore Amiga computers use. There seems to be a plenty of tutorials on modifying PC drives for Amiga on the Web. The same modificaqtions should work for the Altos 586.

Function	Pin		Direction	Function	Note
GND	1	2	Output	Motor Enable	
	3	4			
	5	6			
	7	8	Input	Sector Index	
	9	10	Output	Disk Select 0	
	11	12	Output	Disk Select 1	
	13	14			
	15	16	Output	Motor Enable	Jumpered with E32
	17	18	Output	Step Direction	
	19	20	Output	Step Pulse	
	21	22	Output	Write Data	
	23	24	Output	Write Gate	
	25	26	Input	Track 0	
	27	28	Input	Write Protect	
	29	30	Input	Read Data	
	31	32	Output	Side/Head Select	Jumpered by E1, closed for double-sided drive
	33	34	Input	Drive Ready	

4. Firmware Interface

The firmware interface is unsurprisingly very similar to 586T. A notable difference is addition of the Floppy Channel, described below.

The *586T/986T System Reference Manual* is not too clear about the offset of various register blocks though they can be for the most part inferred from their respective lengths, since they immediately follow each other.

The layout of register blocks on the 586 is somewhat different due to the inclusion of the *Floppy Drive Interface Registers* block before the *Time of Day Registers*. offsets of each block. It is incidentally also the offset of each command register the firmware looks at, since each block starts with a command register.

Offset	Description
0Ah	Communication Channel Registers (Port 1)
20h	Communication Channel Registers (Port 2)
36h	Communication Channel Registers (Port 3)
4Ch	Communication Channel Registers (Port 4)
62h	Communication Channel Registers (Port 5)
78h	Communication Channel Registers (Port 6)
8Ah	Floppy Drive Interface Registers
D4h	Time of Day Registers

The *Communication Channel Registers* and *Time of Day Registers* are described in the *586T/986T System Reference Manual*, respectively. The rest of this section deals with the Floppy interface exclusively.

4.1. Floppy Drive Interface Registers

This structure is present at offset 8Ah from the command control block (CCB). Not all members are used by all commands.

Offset	Size	Description	Typical Value
+00h	BYTE	Command	87h, 88h, . . .
+01h	BYTE	Status	00h, 48h, 40h, C0h, . . .
+02h – +04h	BYTE[3]	Next Floppy Command Pointer	Pointer to *Floppy Command Queue* entry
+05h	BYTE	Floppy Command Queue Size	02h
+06h	BYTE	Last Floppy Command Index	01h
+07h	BYTE	Next Floppy Command Index	00h
+08h	BYTE	Unused	00h
+09h	BYTE	Unused	00h
+0Ah – +B3h	BYTE[32]	Floppy 1 Parameters	*Floppy Parameters*
+B4h – +D3h	BYTE[32]	Floppy 2 Parameters	*Floppy Parameters*

4.2. Floppy Drive Interface Commands

There factory software has been seen issuing three different command, with two of them having known function:

Command	Description
87h	Set Floppy Drive Parameters
88h	Submit Floppy Command Queue
8fh	Unknown

As with the other command blocks, the most significant bit indicates to the peripheral controller that there's a new command to process. Once the processing is finished, the peripheral controller sets the bit to zero.

4.2.1. Set Floppy Drive Parameters

The *Set Floppy Drive Parameters* command reads the *Floppy Parameters* structure (see below) and saves it in the controller's SRAM. The other registers in the block are ignored. This command needs to be called to set the sector size before any Floppy I/O.

Failure to set the sector size up will cause the read commands to overflow a data buffer, corrupting the controller SRAM. The main processor firmware calls this command on startup.

4.2.2. Submit Floppy Command Queue

The *Submit Floppy Command Queue* command submits the *Floppy Command Queue* for processing. Said queue is in fact a ring buffer whose entries are the addresses of *Floppy Disk Interface Parameter Block* structures (described below).

Upon submission, the processing starts with the entry on address in *Next Floppy Command Pointer*. an entry, it increments the next entry index, wrapping it around the queue size if necessary. The processing finishes once the next entry index is equal to the last entry index.

Note that the I/O processor modifies the *Next Floppy Command Index*register when it finishes handlin an entry, but keeps the value in *Next Floppy Command Pointer* unchanged, while still using it to calculate the address of the next entry to process. When you submit another batch of commands When you submit another batch of commands you need to adjust it accordingly.

This command doesn't use the *Floppy Parameters* in the register block, though it requires that the parameters are set by the *Set Floppy Drive Parameters* previously.

4.3. Floppy Parameters

Used to issue I/O commands, not used by command.

Offset	Size	Description	Typical Value
+00h	WORD	Unknown	0200h
+02h	WORD	Sector size	0200h
+04h	BYTE	Unknown	06h
+05h – +1fh	BYTE[. . .]	Unknown/Filler (00h)	00h . . .

4.4. Floppy Command Queue

Offset	Size	Description	Typical Value
+00h	DWORD	Floppy Command Block Address	Pointer to *Floppy Command Block*
+04h	DWORD	Floppy Command Block Address	Pointer to *Floppy Command Block*
. . .			
+04h * n	DWORD	Floppy Command Block Address	Pointer to *Floppy Command Block*

4.5. Floppy Disk Interface Parameter Block

Offset	Size	Description	Typical Value
+00h	High 4 bits	Command	10h, 20h, 1Ah, 2Ah, . . .
	Low 4 bits	Retries	
+01h	BYTE	Status	
+02h	BYTE	Drive Number	00h
+03h	BYTE	Track	00h – 28h
+04h	BYTE	Head	00h, 01h
+05h	BYTE	Sector	01h – 09h
+06h – +09h	BYTE[3]	Data Buffer Address	

5. Reading a Floppy from the Firmware ROM Monitor

This section provides a concrete example of using the Floppy Interface of the peripheral controller.

```
FAILED POWER-UP TEST B   Controller board error. I got no board.
Monitor Version a2.2
Enter [1] to boot from Hard Disk
Enter [2] to boot from Floppy Disk
Enter [3] to enter Monitor
Enter option: 3
```

First let's find the Channel Control Block address (CCB). Read it from 1FFFCh:

```
< A, B, D, G, I, K, L, M, O, R, S, X > D 1000:FFFC 4
  1000:FFFC                            16040000   *................*
```

It's 416h. The channel register offsets are calculated from this address. The new command register is at offset 05h from the beginning of the CCB, therefore at address 41Bh.

Now let's prepare a small program that just bumps the command counter so that the controller firmware will know it needs to check for new commands. It then invokes to return to the monitor. We'll place it at address 2000h, which is an arbitrarily chosen address in the DRAM.

It's not strictly necessary here, since each key press results in a serial command and the command registers are checked anyway, but would be necessary for floppy communication outside the monitor.

```
< A, B, D, G, I, K, L, M, O, R, S, X > A 0:2000
  0000:2000  00-fe            incb 0x41b
  0000:2001  00-06
  0000:2002  00-1b
  0000:2003  00-04
  0000:2004  00-cc            int 3
  0000:2005  00-,
```

We can proceed setting the floppy parameters. This is also not strictly necessary if we're using the ROM monitor, because the firmware must have initialized the parameters for us.

Let's do it anyway though, for demonstration purposes. It might also be useful if the floppy format uses a sector of size other than 512 bytes.

The floppy command register is at offset 8Ah from the beginning of the CCB, therefore at address 4A0h.

For reasons explained below, we start setting the bytes from 4A1h and set the command byte (4A0h) last.

```
< A, B, D, G, I, K, L, M, O, R, S, X > A 0:04A1
  0000:04A1  00-00            Status bit, doesn't matter
  0000:04A2  60-60            Buffer pointer, not used
  0000:04A3  09-09
  0000:04A4  00-00
  0000:04A5  02-02
  0000:04A6  00-00            Pointer counter, not used
  0000:04A7  00-00
  0000:04A8  00-00            Unknown
```

```
  0000:04A9  00-00
  0000:04AA  00-00                Floppy 1 parameters
  0000:04AB  02-02
  0000:04AC  00-00
  0000:04AD  02-02
  0000:04AE  06-06
  0000:04AF  00-00
      ...
  0000:04C9  00-00
      ...
  0000:04CA  00-00                Floppy 2 parameters
  0000:04CB  02-02
  0000:04CC  00-00
  0000:04CD  02-02
  0000:04CE  06-06
  0000:04CF  00-00
  0000:04E9  00-00
  0000:04EA  11-,
< A, B, D, G, I, K, L, M, O, R, S, X > A 0:04A0
  0000:04A0  00-87                The "Floppy Parameters" command byte
  0000:04A1  00-,
```

We set the command byte last to make sure the controller firmware doesn't see a valid command until the parameters are all set.

Now we can bump the command counter with the program we entered in at the beginning (though, as stated above, this is not really necessary):

```
< A, B, D, G, I, K, L, M, O, R, S, X > G 0:2000
Break ....
CS:IP FC00:0000  Flags  - - - - S - - - -
  AX    BX    CX    DX    SI    DI    DS    ES    SS    SP    BP
  0000  0000  0000  0000  0000  0000  0000  0000  0000  0FFC  0000
```

Let's enter some actual floppy I/O requests now.

Start with a seek:

```
< A, B, D, G, I, K, L, M, O, R, S, X > A 0:2100
  0000:2100  00-10                Seek command
  0000:2101  00-00                Status
  0000:2102  00-00                Unknown, 00h
  0000:2103  00-05                Track
  0000:2104  00-01                Head
  0000:2105  00-05                Sector
  0000:2106  00-,
```

Seek to a different track, so that we're sure we see the head move:

```
< A, B, D, G, I, K, L, M, O, R, S, X > A 0:2200
  0000:2200  00-10                Seek command
  0000:2201  00-00                Status
  0000:2202  00-00                Unknown, 00h
```

```
  0000:2203  00-25                  Track
  0000:2204  00-01                  Head
  0000:2205  00-05                  Sector
  0000:2206  00-,
```

And then read a sector. The controller will seek back to the appropriate track:

```
< A, B, D, G, I, K, L, M, O, R, S, X > A 0:2300
  0000:2300  00-10                  Read command
  0000:2301  00-00                  Status
  0000:2302  00-00                  Unknown, 00h
  0000:2303  00-05                  Track
  0000:2304  00-01                  Head
  0000:2305  00-05                  Sector
  0000:2306  00-00                  Destination buffer address: 00002500h
  0000:2307  00-25
  0000:2308  00-00
  0000:2309  00-00
  0000:230A  00-,
```

Create a queue with the three commands we created above:

```
< A, B, D, G, I, K, L, M, O, R, S, X > A 0:2400
  0000:2400  00-00                  First command at 00002100h
  0000:2401  00-21                  (See above: seek to track 5)
  0000:2402  00-00
  0000:2403  00-00
  0000:2404  00-00                  First command at 00002200h
  0000:2405  00-22                  (See above: seek to track 25h)
  0000:2406  00-00
  0000:2407  00-00
  0000:2408  00-00                  First command at 00002300h
  0000:2409  00-23                  (See above: read from CHS 5/1/5 to 2500h)
  0000:240A  00-00
  0000:240B  00-00
  0000:240C  00-,
```

Now invoke the queue the same way as we set the parameters:

```
< A, B, D, G, I, K, L, M, O, R, S, X > A 0:04A1
  0000:04A1  00-ff                  Status
  0000:04A2  60-00                  The queue is at 00002400h
  0000:04A3  09-24
  0000:04A4  00-00
  0000:04A5  02-04                  Queue Length
  0000:04A6  00-03                  Last valid command is 03h commands in the queue
  0000:04A7  00-00                  Start with the command 00h
  0000:04A8  00-00
  0000:04A9  00-00
  0000:04AA  00-,
< A, B, D, G, I, K, L, M, O, R, S, X > A 0:04A0
  0000:04A0  00-c8                  Now set the command byte
  0000:04A1  48-,                   to process the queue
```

```
< A, B, D, G, I, K, L, M, O, R, S, X > G 0:2000
Break ....
CS:IP FC00:0008  Flags  - - - - S - A - -
  AX    BX    CX    DX    SI    DI    DS    ES    SS    SP    BP
  0000  0000  0000  0000  0000  0000  0000  0000  0000  0FF8  0000
```

The command is in. Let's keep checking the status byte until it indicates success:

```
< A, B, D, G, I, K, L, M, O, R, S, X > D 0:04A1 1
  0000:04A1    48                                  *.H.$...........*
```

The status is now 48h. We need it to be 40h. Try some more:

```
< A, B, D, G, I, K, L, M, O, R, S, X > D 0:04A1 1
  0000:04A1    40                                  *.@.$...........*
```

Now the status is 40h, which indicates no error.

The sector data is now at 2500h, let's check it out:

```
< A, B, D, G, I, K, L, M, O, R, S, X > D 0:2500 200
  0000:2500  EB080000 30000000 0002CCEB FD000000   *k...0.....Lk}...*
      ...
  0000:26F0  00000000 00000000 00000000 00000000   *................*
< A, B, D, G, I, K, L, M, O, R, S, X >
```

5.1. Z80 ROM Monitor

The factory firmware on the IP seems to include some sort of debug monitor capability.

Having a firmware monitor is very helpful for investigation of the system. It would allow controlling the Z80 peripherals directly and operating them without the main 8086 processor.

Unfortunately, I didn't discover a way to enter the factory ROM monitor. I've ended up temporarily replacing the ROM with a different one, allowing me to poke around the controller (whilst the main processor is just stuck in an endless loop).

5.1.1. Custom ROM Monitor

My monitor program is a slight modification[2] of a Z80 monitor made by MCook for the ZMC80 computer and adjusted for MPF-1 by F.J.Kraan. It is free software (MIT licensed) and fits the purpose perfectly.

The following capture of the ROM monitor output provides some insight into what functionality the monitor provides:

```
Altos 586 IOP Debug Monitor 0.7
<https://github.com/lkundrak/altos586/>
2015 MCook, 2022 F.J.Kraan, 2022 Lubomir Rintel

         Input ? for command list
>?
ZMC80 Monitor Command List
? - view command list
C - clear screen
D - print 100h bytes from specified location
E - edit bytes in memory
F - fill memory range with value
G - jump to memory value
K - call to memory value
M - copy bytes in memory
P - print port scan (00-FF)
R - monitor reset
S - calculate checksum for memory range
Z - dump user registers (STEP)
I - read I/O port
O - write I/O port
: - download intel hex
+ - print next block of memory
- - print previous block of memory

>
```

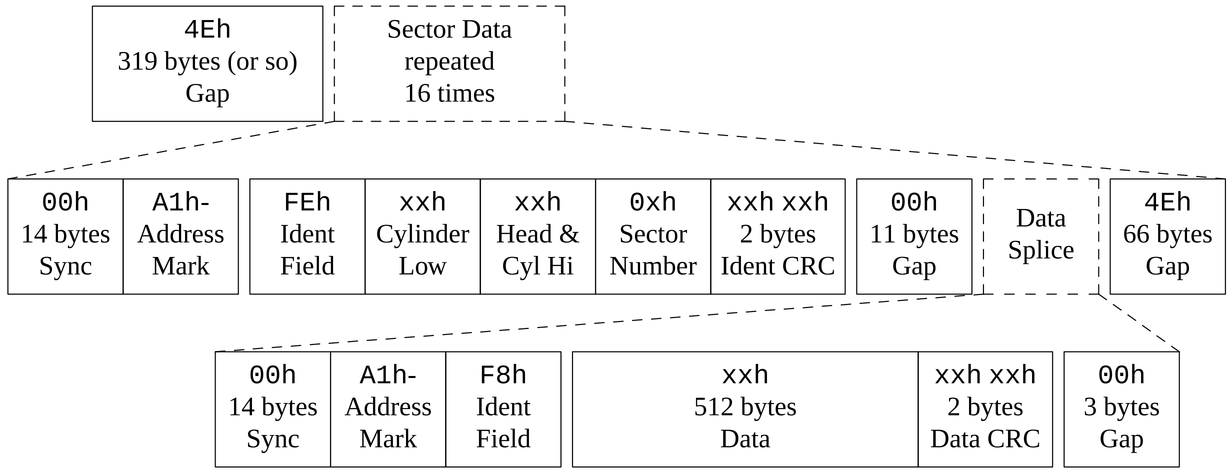
[2] Source code and a binary image are available at *https://github.com/lkundrak/Z80SerialMonitor-586/*

6. Hard Drive Format

The 586T manual is fairly vague and incorrect about the hard drive formatting. The illustration seem to be taken from a hard drive manual and doesn't take specifics of the 586's controller into account. In particular, the gap sizes are wrong and there's a mention of 4-byte ECC where in reality a 16-bit CRC is used.

Other than that, the drive formatting resembles what has been common on drives of the era. It uses MFM encoding as has been common on contemporary hard drives and indeed roughly the same as on floppies since IBM 53FD. In fact, the IBM 53FD manual provides a good description of the overall format.

Here's a more accurate picture (observed on a drive formatted on a 586 system):



Just as with about any MFM-encoded floppy or a hard drive, the `A1h` address mark byte is encoded with an error. It lacks the clock pulse between the fifth and sixth data bit, even though both are zero. This is used to synchronize the decoding logic.

The regular CRC-16-CCITT algorithm[3] is used to calculate the content of the CRC fields. Note that the calculation of the Ident CRC starts from the `FEh` byte, Data CRC is calculated only for the data bytes, not the `F8h` identifiaction field. The `A1h` address mark is not included in either case.

The gaps are quite a bit larger than usual. As a result, less useful data fits on a track—16 sectors of 512 bytes compared to more usual 17.

The falling slope of the (active-low) *Index* signal happens sometime during the first gap.

The overall structure is created during the formatting. On write, the data along with surrounding patterns are spliced between two gaps while the (active-low) *Write Gate* signal is asserted. The timing doesn't have to be completely precise as the surrounding gaps provide some "wiggle room."

---

[3] Initialized with value of 0FFFFh, using the $x^{16} + x^{12} + x^5 + x^0$ generator polynomial, MSB first

6.1. Track dump

Below is a hex dump of decoded MFM stream of a part of a track as it looks immediatelly after formatting. The data bytes are all `E5h`.

The start is arbitrary, not synchronized on the *Index* signal. In fact, it begins right after the last sector passed under the head and a new platter revolution began.

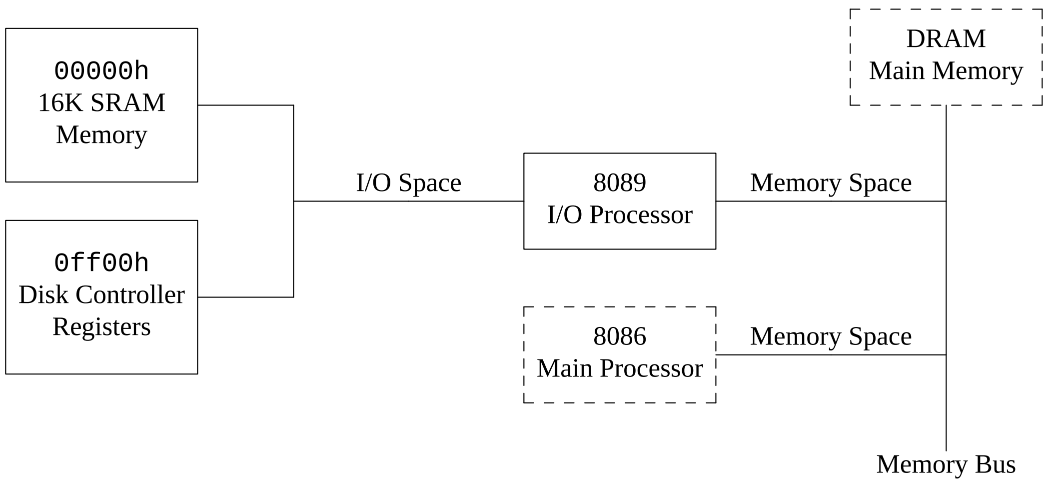
The address mark bytes are highlighted in bold font.

```
0000  4e 4e 4e 4e 4e 4e 4e 4e  4e 4e 4e 4e 4e 4e 4e 4e   Initial gap
*
0130  4e 4e 4e 4e 4e 4e 4e 4e  4e 4e 4e 4e 4e 4e 4e 00
0140  00 00 00 00 00 00 00 00  00a1 fe 00 00 00 b9 d7   The CHS 0,0,0 ident
0150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
0160  a1 f8 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5   Data of CHS 0,0,0 sector
0170  e5 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5
*
0360  e5 e5 08 51 00 00 00 00  4e 4e 4e 4e 4e 4e 4e 4e   Sector end, CRC and gap
0370  4e 4e 4e 4e 4e 4e 4e 4e  4e 4e 4e 4e 4e 4e 00 00
0380  00 00 00 00 00 00 00 00  00 00 00 00 00a1 fe 00   Next sector ident
0390  00 01 a9 f6 00 00 00 00  00 00 00 00 00 00 00 00
03a0  00 00 00 00a1 f8 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5   Sector CHS 0,0,1 data
03b0  e5 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5
*
05a0  e5 e5 e5 e5 e5 e5 08 51  00 00 00 00 4e 4e 4e 4e   Sector CHS 0,0,1 ends
```

7. Hard Drive Controller Board

This hard drive controller supports two hard drives connected via ST-506 compatible interface.

It is based around the Intel 8089 I/O processor. The 8089 is connected to on-board logic and sector buffer SRAM via a private bus, allowing them to be accessed without congesting the system bus. It is also connected to the main memory via the system bus, allowing for efficient DMA transfers of the sector data.



7.1. I/O Programs

The 8089 is controlled with specialized programs placed in the main memory and synchronized with the main CPU with interrupts.

The firmware includes one such program to handle the disk reads and writes. It is parametrized by I/O parameter blocks (IOPB) placed in the main memory. It is described in detail in section below.

So far, I’ve observed two I/O programs for the controller board: one in the firmware and another on a diagnostic floppy. The Xenix operating system apparently uses yet another—I haven’t looked into that one yet.

The following paragraphs give a quick overview of the two I/O programs followed by commented disassembly listings. The programs can be assembled using Eric Smith’s Assembler for the 8089.[4]

7.2. Firmware I/O Handling Program

This is the simple of the two I/O programs. It is provided in the firmware image and run on the 8089 on the hard drive board to service the I/O operations.

It is controlled by the I/O parameter block structure to ultimately transfer data between a main memory buffer and the hard drive. The data is not transferred directly between the controller’s data port and the main memory. Instead, the data is bounced through a a buffer in controller board’s SRAM.

This utilizes the main memory bus more efficiently. The controller data port transfers eight bits of data at a time at a rate determined by the disk rotation speed. These slow accesses are done over a bus that’s private to the controller board, avoiding congestion of the main bus. On the other hand, the SRAMs allows 16-bit accesses at a much faster rate, utilizing the main memory as little as possible.

On reads, the data is first transferred from the data port to the SRAM buffer and then from SRAM to the main memory. On writes, the data is copied from main memory to SRAM buffer and then from SRAM to the data port.

The number of sectors specified in IOPB determines how many transfers are done between the data port and the SRAM. The number of bytes transferred between SRAM and main memory is specified in IOPB directly. These two numbers are related—main memory transfer needs to transfer sufficient number of bytes to include complete sector data.

The I/O program listing[5] follows.

---

[4] Available from *https://github.com/brouhaha/i89*

[5] Assembly source available from *https://github.com/lkundrak/altos586/blob/main/hdc/iop3268.asm*

```
;--------------------------------------------------------------------------
;  8089 Transfer configurations.
;--------------------------------------------------------------------------
PORT_TO_BLK       EQU 08A28H  ; Port GA to memory GB
BLK_TO_PORT       EQU 05628H  ; Memory GB to port GA
BLK_TO_BLK        EQU 0C208H  ; Memory GA to memory GB

;--------------------------------------------------------------------------
;  Program parameters, passed by the user.
;--------------------------------------------------------------------------
IOPB              STRUC
                  DS 4        ; Task Block pointer
IOPB_OP:          DS 1
IOPB_STATUS:      DS 1
IOPB_CYL:         DS 2
IOPB_DRV_HD:      DS 1
IOPB_SEC:         DS 1; Starting sector
IOPB_BYTE_CNT:    DS 2        ; The DMA buffer size
IOPB_DMA_BUF:     DS 4       ; The DMA b  uffer address
IOPB_SEC_CNT:     DS 2       ; Number of sector  s
                              ; The following members are reserved for use by the IOP.
IOPB_IO_SIZE:     DS 2       ; Written by the IOP
IOPB_LAST_CYL:    DS 2        ; Last cylinder we've seeked to
IOPB_LINK_REG:    DS 2        ; Return address of subroutine calls
IOPB              ENDS

;--------------------------------------------------------------------------
;  Bits of IOPB_OP.
;--------------------------------------------------------------------------
OP_BIT_RD         EQU 0       ; Read (0) or write (1)
OP_BIT_FORMAT     EQU 2
OP_BIT_ECC        EQU 3       ; Include the ECC bits on read
OP_BIT_SEL        EQU 5       ; Drive/Head has changed
OP_CMD_BITS       EQU 0FH

;--------------------------------------------------------------------------
;  Bits of IOPB_STATUS.
;--------------------------------------------------------------------------
STATUS_BIT_ERR    EQU 7

;--------------------------------------------------------------------------
;  These are addresses in 8089's I/O space.
;--------------------------------------------------------------------------
HDD_SRAM_ADDR     EQU 00000H
HDD_REGS_ADDR     EQU 0FFD0H

;--------------------------------------------------------------------------
;  Registers at HDD_REGS_ADDR.
;--------------------------------------------------------------------------
PORT_DATA         EQU 00H
PORT_DRV_HD       EQU 02H
PORT_04H          EQU 04H
PORT_CMD_STAT     EQU 06H
```

```
;-------------------------------------------------------------------
;  The entry point.
;-------------------------------------------------------------------
ENTRY:          movi   gc,HDD_REGS_ADDR

                ; Is bit 5 set?
                jnbt   [pp].IOPB_OP,OP_BIT_SEL,NO_SEL

                ; If bit 5 is set, then drive and head might have
                ; Changed and we need ensure correct head is selected.
                movbi  [gc].PORT_CMD_STAT,80H
                movb   [gc].PORT_DRV_HD,[pp].IOPB_DRV_HD
B0:             jnbt   [gc].PORT_CMD_STAT,7,B0

                movbi  [gc].PORT_CMD_STAT,20H
                movi   [pp].IOPB_LAST_CYL,0
C0:             jbt    [gc].PORT_CMD_STAT,0,C0

                ljnbt  [gc].PORT_04H,0,RETURN_81H  ; Error?

NO_SEL:         andbi  [pp].IOPB_OP,OP_CMD_BITS  ; Command bits mask
                ljzb   [pp].IOPB_OP,RETURN_00H  ; Command is zero

                ; Select head
                movb   [gc].PORT_DRV_HD,[pp].IOPB_DRV_HD
B1:             jnbt   [gc].PORT_CMD_STAT,7,B1

                ; Seek to cylinder
                movb   [gc],[pp].IOPB_LAST_CYL  ; [gc].PORT_DATA
                movb   [gc],[pp].IOPB_LAST_CYL+1  ; [gc].PORT_DATA
                movb   [gc].PORT_04H,[pp].IOPB_CYL
                movb   [gc].PORT_04H,[pp].IOPB_CYL+1
                movbi  [gc].PORT_CMD_STAT,10H
                mov    [pp].IOPB_LAST_CYL,[pp].IOPB_CYL
C1:             jbt    [gc].PORT_CMD_STAT,0,C1
D0:             jnbt   [gc].PORT_04H,1,D0

                ; Zero sectors? Nothing to do.
                jzb    [pp].IOPB_SEC_CNT,RETURN_00H

                ; A read?
                jbt    [pp].IOPB_OP,OP_BIT_RD,HDD_SRAM_XFER

                ; Whis is a write. Transfer the data from main memory
                ; DMA buffer to controller board SRAM first
                lpd    ga,[pp].IOPB_DMA_BUF
                movi   gb,PORT_DATA
                mov    bc,[pp].IOPB_BYTE_CNT
                call   [pp].IOPB_LINK_REG,MEM_SRAM_XFER
```

```
;------------------------------------------------------------------
; Transfer between SRAM and the HDD's data register.
; Common to read and write.
;------------------------------------------------------------------
HDD_SRAM_XFER:  movi  gb,HDD_SRAM_ADDR
                movi  mc,0FE80H; Why?
                movi  ga,HDD_REGS_ADDR+PORT_DATA
                movi  [pp].IOPB_IO_SIZE,512

                ; A read? Setup for SRAM (GA) to HDD (GB) transfer.
                jnbt  [pp].IOPB_OP,OP_BIT_RD,XFER_WR
                movi  cc,PORT_TO_BLK
                wid   8,16

                ; Add 5 bytes if ECC requested.
                jnbt  [pp].IOPB_OP,OP_BIT_ECC,XFER_SEC
                movi  [pp].IOPB_IO_SIZE,512+5
                jmp   XFER_SEC

                ; A write. Setup for transfer from SRAM to HDD.
XFER_WR:        movi  cc,BLK_TO_PORT
                wid   16,8
                jnbt  [pp].IOPB_OP,OP_BIT_FORMAT,XFER_SEC
                movi  [pp].IOPB_IO_SIZE,4  ; Sector header

                ; Transfer one sector
XFER_SEC:       mov   bc,[pp].IOPB_IO_SIZE
                movb  [gc],[pp].IOPB_SEC  ; [gc].PORT_DATA
                xfer

                ; Check for errors
                movb  [gc].PORT_CMD_STAT,[pp].IOPB_OP
                jmcne [gc].PORT_CMD_STAT,RETURN_ERR

                ; Are we done or there's more?
                decb  [pp].IOPB_SEC_CNT
                jzb   [pp].IOPB_SEC_CNT,XFER_DONE
                incb  [pp].IOPB_SEC  ; There's more.
                jmp   XFER_SEC

                ; HDD - SRAM transfer done. We're done if it was a write.
XFER_DONE:      jnbt  [pp].IOPB_OP,OP_BIT_RD,RETURN_00H

                ; This was a read. We need to transfer from SRAM to DMA buffer.
                lpd   gb,[pp].IOPB_DMA_BUF
                movi  ga,PORT_DATA
                mov   bc,[pp].IOPB_BYTE_CNT
                call  [pp].IOPB_LINK_REG,MEM_SRAM_XFER

                ; Successful return.
RETURN_00H:     movbi [pp].IOPB_STATUS,0
                jmp   DONE
```

```
;——————————————————————————————————————————————————————————————
; A subroutine that does a block-to-block transfer.
; Used to copy data between the controller SRAM and the DMA buffer in
; main memory, in either direction.
;——————————————————————————————————————————————————————————————
MEM_SRAM_XFER:  wid   16,16
                movi  cc,BLK_TO_BLK
                xfer
                nop
                movp  tp,[pp].IOPB_LINK_REG

;——————————————————————————————————————————————————————————————
; Error return. Communicate the error from the status.
; register to the IOPB.
;——————————————————————————————————————————————————————————————
RETURN_ERR:     movb  [pp].IOPB_STATUS,[gc].PORT_CMD_STAT
                andbi [pp].IOPB_STATUS,7EH
                setb  [pp].IOPB_STATUS,STATUS_BIT_ERR
                movbi [gc].PORT_CMD_STAT,00H
                jmp   DONE

                ; Error 1.
RETURN_81H:     movi  [pp].IOPB_STATUS,(1<<STATUS_BIT_ERR)+1

DONE:           sintr
                hlt
```

7.3. Diagnostic I/O Handling Program

This one is from the diagnostic utility floppy program, that's capable of reading, writing, verifying and formatting the disk.

It is designed for high-performance I/O. The diagnostic program first invokes a routine that copies another I/O program into the controller board SRAM. That way it can be executed without contending for the main memory.

It is also able to enqueue two I/O requests, presumably for submitting requests to two drives concurrently.

The I/O program listing[6] follows.

```
;------------------------------------------------------------
;  Hardware interface.
;------------------------------------------------------------
PORT_TO_BLK     EQU 08A28H  ; Port GA to memory GB
BLK_TO_PORT     EQU 05628H  ; Memory GB to port GA
BLK_TO_BLK      EQU 0C008H  ; Memory GA to memory GB

HDD_SRAM_ADDR   EQU 00000H  ; In I/O space
HDD_REGS_ADDR   EQU 0FFD0H  ; ditto

REG_DATA        EQU 00H
REG_DRV_HD      EQU 02H ; Write Drive & Head select
REG_CYL_ST1     EQU 04H ; Write cylinder, read some status bits
REG_CMD_ST2     EQU 06H ; Write command, read more status bits
REG_28H         EQU 28H

; Write bits of REG_CMD_ST2
CMD_BIT_RD      EQU 0   ; Read Sector Data
CMD_BIT_WR      EQU 1   ; Write Sector Data
CMD_BIT_FMT     EQU 2   ; Format Track
CMD_BIT_LONG    EQU 3   ; Include ID field on read

; Read bits of REG_CYL_ST1
ST1_BIT_TR0     EQU 0   ; Track 0
ST1_BIT_SC      EQU 1   ; Seek Complete

; Read bits of REG_CMD_ST2
ST2_BIT_BSY     EQU 0
ST2_BIT_UNK1    EQU 1   ; Recoverable, retry!
ST2_BIT_BAD_SEC EQU 2   ; Sector flagged bad
;ST2_BIT_NO_SEC EQU 3   ; Record Not found
ST2_BIT_CRC_ERR EQU 4   ; CRC Error
ST2_BIT_UNK5    EQU 5
ST2_BIT_WE      EQU 6   ; Write Error (signalled from drive, latched)
ST2_BIT_RDY     EQU 7   ; Ready (signalled from drive)
```

[6] Assembly source available from *https://github.com/lkundrak/altos586/blob/main/hdc/hd-test.asm*

```
;------------------------------------------------------------------
;  Structures.
;------------------------------------------------------------------
; Parameter Block
PB              STRUC
PB_TB:          DS 4
PB_IN1_ADDR:    DS 4     ; Offset:Segment of first command block
PB_IN2_ADDR:    DS 4     ; Offset:Segment of second command block
PB_OUT1_ADDR:   DS 4    ; Of  fset:Segment of first result block
PB_OUT2_ADDR:   DS 4    ; Of  fset:Segment of second result block
PB              ENDS

; Command Block.
CB              STRUC
CB_OP:          DS 1
CB_STAT:        DS 1
CB_CYL_LO:      DS 1
CB_CYL          EQU CB_CYL_LO
CB_CYL_HI:      DS 1
CB_DRV_HD:      DS 1
CB_SEC:         DS 1
CB_LEN:         DS 2
CB_BUF_ADDR:    DS 4
CB_REG_28H:     DS 1
CB_NUM_SECS:    DS 1
CB_RETRIES:     DS 1
                DS 1
CB_SIZE:
CB              ENDS

; Value following the Command Block in the Working Area.
WA              STRUC
                DS CB_SIZE
WA_CUR_CYL:     DS 2
WA_CB_ADDR:     DS 4; Original addr    ess of this CB
WA_LINK:        DS 4
WA_LINK1:       DS 4
WA_XFER_LEN:    DS 2
WA_TMP_CMD:     DS 1
WA_CUR_DRV_HD:  DS 1
WA_LINK2:       DS 4
WA_ERR_TRIES:   DS 1
WA_ERR_SEC:     DS 1
WA_SIZE:
WA              ENDS
```

```
; Bits of CB_OP.
OP_BIT_RD         EQU 0     ; Read (0) or write (1)
;OP_BIT_WR     EQU 1   ; Diags sets, this, but we don't look
OP_BIT_FMT        EQU 2     ; Format
OP_BIT_LONG       EQU 3     ; Include ID field in reads
OP_BIT_SEL        EQU 5     ; Drive/Head has changed
OP_BIT_NO_RECAL   EQU 6
OP_BIT_UNK7       EQU 7

;--------------------------------------------------------------------
;  The entry point.
;--------------------------------------------------------------------
CODE:
                lpd     ga,[pp].PB_IN1_ADDR
                jzb     [ga].CB_OP,CH1_CHECKED
                movi    gb,WA1
                movi    gc,WA1
                lcall   [gc].WA_LINK,COPY_CB
CH1_CHECKED:

                lpd     ga,[pp].PB_IN2_ADDR
                jzb     [ga].CB_OP,CH2_CHECKED
                movi    gb,WA2
                movi    gc,WA2
                lcall   [gc].WA_LINK,COPY_CB
CH2_CHECKED:

                movi    ga,HDD_REGS_ADDR

                movi    gc,WA1
                jzb     [gc].CB_OP,CH1_SEEKED
                lcall   [gc].WA_LINK,SET_TRACK
                jnbt    [gc].CB_OP,OP_BIT_SEL,SEEK_CH1
                lcall   [gc].WA_LINK1,SEL_DRIVE
                jmp     CH1_SEEKED
SEEK_CH1:       lcall   [gc].WA_LINK1,SEEK_TO_CYL
CH1_SEEKED:

                movi    gc,WA2
                jzb     [gc].CB_OP,CH2_SEEKED
                lcall   [gc].WA_LINK,SET_TRACK
                jnbt    [gc].CB_OP,OP_BIT_SEL,SEEK_CH2
                lcall   [gc].WA_LINK1,SEL_DRIVE
                jmp     CH2_SEEKED
SEEK_CH2:       lcall   [gc].WA_LINK1,SEEK_TO_CYL
CH2_SEEKED:

                movi    gc,WA1
                jzb     [gc].CB_OP,CH1_DONE
                jbt     [gc].CB_OP,OP_BIT_SEL,CH1_OP_DONE
                lcall   [gc].WA_LINK,SET_TRACK
                jbt     [gc].CB_OP,OP_BIT_RD,CH1_WAIT_SEEK
                lcall   [gc].WA_LINK,COPY_FROM_DMA
```

```
                    ; Wait for Seek Complete
CH1_WAIT_SEEK:  jnbt    [ga].REG_CYL_ST1,ST1_BIT_SC,CH1_WAIT_SEEK
                lcall   [gc].WA_LINK2,DO_DISK_CMD
                jnbt    [gc].CB_OP,OP_BIT_RD,CH1_OP_DONE
                lcall   [gc].WA_LINK,COPY_TO_DMA
CH1_OP_DONE:    movi    ga,WA1
                lpd     gb,[pp].PB_OUT1_ADDR
                lcall   [gc].WA_LINK,COPY_CB
CH1_DONE:

                movi    gc,WA2
                jzb     [gc].CB_OP,CH2_DONE
                jbt     [gc].CB_OP,OP_BIT_SEL,CH2_OP_DONE
                lcall   [gc].WA_LINK,SET_TRACK
                jbt     [gc].CB_OP,OP_BIT_RD,CH2_WAIT_SEEK
                lcall   [gc].WA_LINK,COPY_FROM_DMA
                ; Wait for Seek Complete
CH2_WAIT_SEEK:  jnbt    [ga].REG_CYL_ST1,ST1_BIT_SC,CH2_WAIT_SEEK
                lcall   [gc].WA_LINK2,DO_DISK_CMD
                jnbt    [gc].CB_OP,OP_BIT_RD,CH2_OP_DONE
                lcall   [gc].WA_LINK,COPY_TO_DMA
CH2_OP_DONE:    movi    ga,WA2
                lpd     gb,[pp].PB_OUT2_ADDR
                lcall   [gc].WA_LINK,COPY_CB
CH2_DONE:

                sintr
                hlt

;------------------------------------------------------------------------
; Transfer data between the controller SRAM buffer and the controller.
; Handles the situations when the transfer crosses track boundary too.
;------------------------------------------------------------------------
DO_DISK_CMD:    ljzb    [gc].CB_NUM_SECS,DISK_CMD_DONE
                movb    ix,[gc].CB_OP
                andi    ix,7h
                ljz     ix,DISK_CMD_DONE
                movbi   [gc].WA_ERR_SEC,0FFH
                movi    gb,HDD_SECTOR_BUF
                movi    [gc].WA_XFER_LEN,512
                jnbt    [gc].CB_OP,OP_BIT_RD,SETUP_WRITE
                movbi   [gc].WA_TMP_CMD,1<<CMD_BIT_RD
                jnbt    [gc].CB_OP,OP_BIT_LONG,SETUP_READ
                movbi   [gc].WA_TMP_CMD,1<<CMD_BIT_LONG|1<<CMD_BIT_RD
                addi    [gc].WA_XFER_LEN,5  ; Include ID field

SETUP_READ:     movi    cc,PORT_TO_BLK
                wid     8,16
                jmp     SETUP_XFER

SETUP_WRITE:    movbi   [gc].WA_TMP_CMD,1<<CMD_BIT_WR
                movi    cc,BLK_TO_PORT
                wid     16,8
```

```
                jnbt    [gc].CB_OP,OP_BIT_FMT,SETUP_XFER
                movbi   [gc].WA_TMP_CMD,1<<CMD_BIT_FMT
                movi    [gc].WA_XFER_LEN,4h

SETUP_XFER:     movi    mc,0FE80H
DO_XFER:        movb    [ga],[gc].CB_SEC
                movp    [gc].WA_CB_ADDR,gb
                mov     bc,[gc].WA_XFER_LEN
                xfer
                movb    [ga].REG_CMD_ST2,[gc].WA_TMP_CMD
                jmcne   [ga].REG_CMD_ST2,DISK_OP_ERROR
                decb    [gc].CB_NUM_SECS
                jzb     [gc].CB_NUM_SECS,DISK_CMD_DONE

                ; Next sector on same track
                incb    [gc].CB_SEC
                jnbt    [gc].CB_SEC,4,DO_XFER

                ; Next head
                incb    [gc].CB_DRV_HD
                movb    mc,[gc].WA_CUR_DRV_HD
                ori     mc,0F00H
                jmce    [gc].CB_DRV_HD,NEXT_CYL
SECTOR_ZERO:    movb    [ga].REG_DRV_HD,[gc].CB_DRV_HD
                movbi   [gc].CB_SEC,0h
                jmp     SETUP_XFER

                ; Next cylinder
NEXT_CYL:       inc     [gc].CB_CYL
                lcall   [gc].WA_LINK1,SEEK_TO_CYL
                andbi   [gc].CB_DRV_HD,0F0H
                ; Wait for Seek Complete
POLL_OP_SEEK:   jnbt    [ga].REG_CYL_ST1,ST1_BIT_SC,POLL_OP_SEEK
                jmp     SECTOR_ZERO
DISK_CMD_DONE:  jbt     [ga].REG_CMD_ST2,ST2_BIT_BSY,DISK_CMD_DONE
                movp    tp,[gc].WA_LINK2

;---------------------------------------------------------------------
;  Retry on errors.
;---------------------------------------------------------------------
DISK_OP_ERROR:  jbt     [ga].REG_CMD_ST2,5,ERR_STATUS
                jbt     [ga].REG_CMD_ST2,1,ERR_STATUS  ; Retry!
                jbt     [gc].CB_OP,OP_BIT_NO_RECAL,NO_NEED_RECAL
                movb    mc,[gc].CB_SEC
                ori     mc,0FF00H
                jmce    [gc].WA_ERR_SEC,ERR_SEC_SET
                movb    [gc].WA_ERR_SEC,[gc].CB_SEC
                movbi   [gc].WA_ERR_TRIES,0h
ERR_SEC_SET:    jbt     [ga].REG_CMD_ST2,ST2_BIT_BAD_SEC,SKIP_RECAL
                jbt     [ga].REG_CMD_ST2,ST2_BIT_CRC_ERR,SKIP_RECAL
                jnzb    [gc].WA_ERR_TRIES,GIVE_UP
```

```
                ; Seek to cylinder zero to recalibrate
                movbi   [ga].REG_CYL_ST1,0
                lcall   [gc].WA_LINK1,SEL_DRIVE
                lcall   [gc].WA_LINK1,SEEK_TO_CYL
                ; Wait for Seek Complete
POLL_RECAL:     jnbt    [ga].REG_CYL_ST1,ST1_BIT_SC,POLL_RECAL
                jmp     RECAL_DONE
SKIP_RECAL:     jbt     [gc].WA_ERR_TRIES,4,GIVE_UP
RECAL_DONE:     incb    [gc].CB_RETRIES
                incb    [gc].WA_ERR_TRIES
NO_NEED_RECAL:  movb    [gc].CB_STAT,[ga].REG_CMD_ST2
                andbi   [gc].CB_STAT,7EH
                movbi   [ga].REG_CMD_ST2,0h

                ; Try again
                ljnbt   [gc].CB_OP,OP_BIT_NO_RECAL,SETUP_XFER
GIVE_UP:        setb    [gc].CB_STAT,7
                movp    tp,[gc].WA_LINK2

;------------------------------------------------------------------
; Communicate the error and cancel both channels.
;------------------------------------------------------------------
ERR_STATUS:     movb    [gc].CB_STAT,[ga].REG_CMD_ST2
                ljmp    ERR_FINISH

;------------------------------------------------------------------
; Copy the Command Block between the controller SRAM and main memory.
;------------------------------------------------------------------
COPY_CB:        movp    [gc].WA_CB_ADDR,ga
                movi    bc,CB_SIZE
                call    [gc].WA_LINK1,MEMCPY_16
                movp    ga,[gc].WA_CB_ADDR
                movbi   [ga].CB_OP,0h
                movi    ga,HDD_REGS_ADDR
                jbt     [ga].REG_CMD_ST2,1,ERR_STATUS
                movp    tp,[gc].WA_LINK

;------------------------------------------------------------------
; Copy a block of data (sector, CB) between main memory and SRAM.
;------------------------------------------------------------------
MEMCPY_16:      wid     16,16
                movi    cc,BLK_TO_BLK
                xfer
                nop
                movp    tp,[gc].WA_LINK1

;------------------------------------------------------------------
; Set head and a cylinder.
;------------------------------------------------------------------
SET_TRACK:      movb    [ga].REG_DRV_HD,[gc].CB_DRV_HD
                movb    [ga].REG_CYL_ST1,[gc].WA_CUR_CYL+0
                movb    [ga].REG_CYL_ST1,[gc].WA_CUR_CYL+1
                movbi   [gc].CB_STAT,0h
```

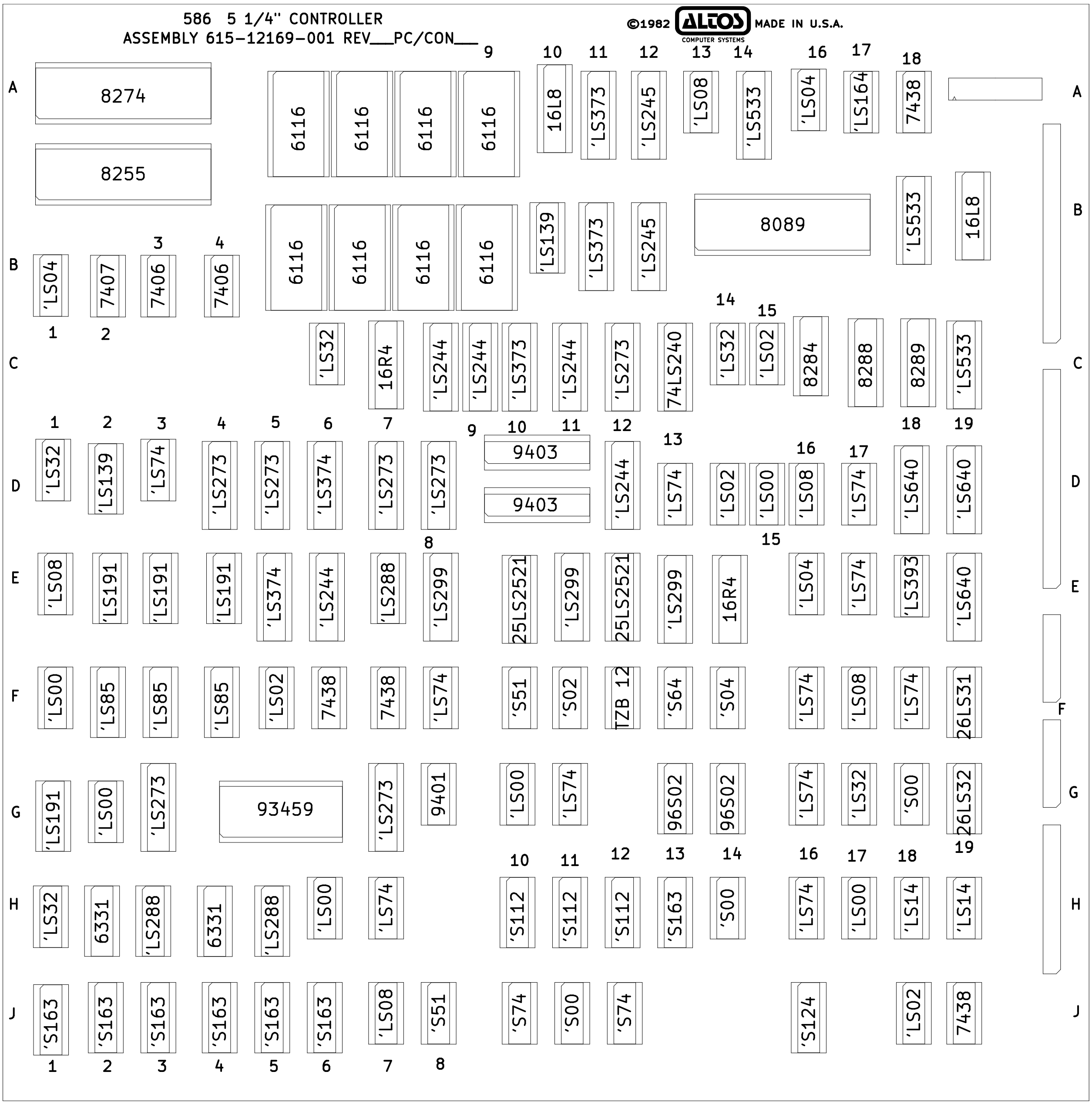
```
                ; Wait for bit 7 to come up for up to FFh attempts
                movi    bc,0h
POLL_DRV_RDY:   dec     bc
                jz      bc,ERR_FINISH
                jnbt    [ga].REG_CMD_ST2,ST2_BIT_RDY,POLL_DRV_RDY
                movp    tp,[gc].WA_LINK

;--------------------------------------------------------------------
;  Seek to a cylinder.
;--------------------------------------------------------------------
SEEK_TO_CYL:    movb    [ga],[gc].WA_CUR_CYL+0
                movb    [ga],[gc].WA_CUR_CYL+1
                movb    [ga].REG_CYL_ST1,[gc].CB_CYL_LO
                movb    [ga].REG_CYL_ST1,[gc].CB_CYL_HI
                movbi   [ga].REG_CMD_ST2,10H
                mov     [gc].WA_CUR_CYL,[gc].CB_CYL
POLL_SEEK:      jbt     [ga].REG_CMD_ST2,ST2_BIT_BSY,POLL_SEEK
                movp    tp,[gc].WA_LINK1

;--------------------------------------------------------------------
;  Select a disk drive unit.
;--------------------------------------------------------------------
SEL_DRIVE:      movbi   [ga].REG_CMD_ST2,80H
                movbi   [gc].CB_STAT,0h
                jnbt    [gc].CB_OP,OP_BIT_UNK7,NO_BIT_7
                movb    [gc].WA_CUR_DRV_HD,[gc].CB_DRV_HD
NO_BIT_7:       movb    [ga].REG_DRV_HD,[gc].CB_DRV_HD

                ; Wait for drive to become ready for up to FFh attempts
                movi    bc,0h
POLL_READY:     dec     bc
                jz      bc,ERR_FINISH
                jnbt    [ga].REG_CMD_ST2,ST2_BIT_RDY,POLL_READY

                movbi   [ga].REG_CMD_ST2,20H
                movi    [gc].WA_CUR_CYL,0h
POLL_SEL:       jbt     [ga].REG_CMD_ST2,ST2_BIT_BSY,POLL_SEL
                movp    tp,[gc].WA_LINK1

;--------------------------------------------------------------------
;  Indicate errors in both channels.
;--------------------------------------------------------------------
ERR_FINISH:     movi    ga,WA1
                orbi    [ga].CB_STAT,81H
                lpd     gb,[pp].PB_OUT1_ADDR
                lcall   [gc].WA_LINK,COPY_CB
                movi    ga,WA2
                orbi    [ga].CB_STAT,81H
                lpd     gb,[pp].PB_OUT2_ADDR
                lcall   [gc].WA_LINK,COPY_CB

                sintr
                hlt
```

```
;------------------------------------------------------------------------
; Copy data indicated by CB from main memory to controller SRAM.
;------------------------------------------------------------------------
COPY_FROM_DMA:  jz      [gc].CB_LEN,COPIED_IN
                setb    [gc].CB_REG_28H,4
                movb    [ga].REG_28H,[gc].CB_REG_28H

                ; Copy from DMA buffer to sector buf
                lpd     ga,[gc].CB_BUF_ADDR
                movi    gb,HDD_SECTOR_BUF
                mov     bc,[gc].CB_LEN
                lcall   [gc].WA_LINK1,MEMCPY_16

                movi    ga,HDD_REGS_ADDR
                movbi   [ga].REG_28H,10H
                ljbt    [ga].REG_CMD_ST2,1,ERR_STATUS
COPIED_IN:      movp    tp,[gc].WA_LINK

;------------------------------------------------------------------------
; Copy data indicated by CB to main memory from controller SRAM.
;------------------------------------------------------------------------
COPY_TO_DMA:    jz      [gc].CB_LEN,COPIED_OUT
                setb    [gc].CB_REG_28H,4
                movb    [ga].REG_28H,[gc].CB_REG_28H

                ; Copy sector buf to DMA buffer
                lpd     gb,[gc].CB_BUF_ADDR
                movi    ga,HDD_SECTOR_BUF
                mov     bc,[gc].CB_LEN
                lcall   [gc].WA_LINK1,MEMCPY_16

                movi    ga,HDD_REGS_ADDR
                movbi   [ga].REG_28H,10H
COPIED_OUT:     movp    tp,[gc].WA_LINK

;------------------------------------------------------------------------
; Command block copies and sector data copies in controller SRAM.
;------------------------------------------------------------------------
WA1:            DSWA_SIZE
WA2:            DSWA_SIZE
HDD_SECTOR_BUF:
```

```
LOADER_OFFSET   EQU 00400H ; In main memory

                ; Pad the loader to 0400H
                DS LOADER_OFFSET-HDD_SECTOR_BUF

;--------------------------------------------------------------------
; The bootstrap routine. Copies the IOP to controller SRAM so that it
; can be executed without main memory contention.
;--------------------------------------------------------------------
LOADER:
                lpd     gc,[pp]; [pp].PB_TB
                lpd     ga,[pp]; [pp].PB_TB
                movi    gb,HDD_SRAM_ADDR

                ; Move 400 bytes back to set BC=IOP base
                movi    bc,LOADER
                not     bc
                inc     bc

                ; Initialize CH2 area
                mov     [gb],bc
                add     ga,[gb]
                add     gc,[gb]
                addi    gc,WA2
                lpd     gb,[pp].PB_IN2_ADDR
                movb    [gc].WA_CUR_DRV_HD,[gb].CB_DRV_HD
                movi    gb,HDD_SRAM_ADDR

                ; Initialize CH1 area
                movp    [gb],ga
                movp    gc,[gb]
                addi    gc,WA1
                lpd     gb,[pp].PB_IN1_ADDR
                movb    [gc].WA_CUR_DRV_HD,[gb].CB_DRV_HD
                movi    gb,HDD_SRAM_ADDR

                ; Relocate the IOP
                movi    bc,HDD_SECTOR_BUF
                lcall   [gc].WA_LINK1,MEMCPY_16

                movi    ga,HDD_REGS_ADDR
                movbi   [ga].REG_28H,10H

                sintr
                hlt
```

7.4. Chip Locations

The drawing below describes the placement of parts on the controller board. Note that only ICs are included.



7.5. Bill of Materials

Currently, only ICs are listed below.

Locations	Part	Description
2A	D8274	Multi-Protocol Serial Controller
2B	7407N	Hex Buffers/Drivers, Open-Collector
2E, 3E, 4E, 1G	DM74LS191N	4-Bit Counter
2F, 3F, 4F	SN74LS85N	4-Bit Magnitude Comparator
2H, 4H	6331-1N	32 x 8-Bit PROM
3A	P8255A-5	Programmable Peripheral Interface
3B, 4B	DM7406N	Hex Buffer, Inverting, Open-Collector
3D, 13D, 17D, 17E, 8F, 16F, 18F, 11G, 16G, 7H, 15H	DM74LS74AN	Dual D-Latch, Positive Edge, with Preset
4G	93459	Programmable Logic Array
5A, 7A, 8A, 9A, 5B, 7B, 8B, 9B	HM6116P-3	2048 x 8-Bit Static RAM
6C, 14C, 1D, 17G, 1H	SN74LS32N	Quad OR Gate
6D, 5E	DM74LS374N	Octal D-Latch, Positive Edge
7C, 14E	PAL16R4CN	Programmable Logic Array
7E, 3H, 5H	DM74LS288N	32 x 8-Bit PROM
8C, 9C, 11C, 12D, 6E	DM74LS244N	Dual 4-Bit Buffer
8E, 11E, 13E	SN74LS299N	Shift Register
8G	N9401N	CRC generator
9D, 10D	N9403N	64-Bit FIFO Buffer Memory
10A, 19B	PAL16L8CN	Programmable Logic Array
10B, 2D	SN74LS139N	Monostable Multivibrator
10E, 12E	AM25LS2521PC	8-bit Equal-To Comparator
10F, 8J	SN74S51N	Dual Configurable Gate
10H, 11H, 12H	DM74S112N	Dual J-K Flip-Flop
10J, 12J	DM74S74N	Dual D-Latch, Positive Edge, with Preset
11A, 11B, 10C	DM74LS373N	Octal D-Latch
11F	DM74S02N	Quad NOR Gate
12A, 12B	DM74LS245N	Octal Bus Transceiver
12C, 4D, 5D, 7D, 8D, 3G, 7G	SN74LS273N	Octal D-Latch, Positive Edge
12F	TZB 12-10	Delay Line
13A, 16D, 1E, 17F, 7J	DM74LS08N	Quad 2-Input AND Gate
13B	D8089-3	Programmable I/O Processor
13C	74LS240N	Dual Inverting 4-Bit Buffer
13F	DM74S64N	Quad Configurable Gate
13G, 14G	96S02PC	Monostable Multivibrator
13H, 1J, 2J, 3J, 4J, 5J, 6J	DM74S163AN	4-Bit Counter, Positive Edge

Locations	**Part**	**Description**
14A, 18B, 19C	SN74LS533N	Octal Transceiver, Inverting
14F	DM74S04N	Hex Inverter
15C, 14D, 5F, 18J	SN74LS02N	Quad NOR Gate
15D, 1F, 2G, 10G, 6H, 16H	SN74LS00N	Quad 2-Input NAND Gate
16A, 1B, 16E	SN74LS04N	Hex Inverter
16C	D8284A	Clock Generator
16J	SN74S124N	Dual Voltage Controlled Oscillator
17A	DM74LS164N	8-Bit Shift Register
17C	D8288	Bus Controller
18A, 6F, 7F, 19J	7438N	Quad NAND Gate, Open-Collector
18C	P8289	Bus Arbiter
18D, 19D, 19E	SN74LS640ND	Octal Bus Transceiver
18E	SN74LS393N	Dual 4-Bit Binary Counter, Negative Edge
18G, 14H, 11J	DM74S00N	Quad 2-Input NAND Gate
18H, 19H	SN74LS14N	Hex Inverter, Schmitt Trigger
19F	AM26LS31CN	Quad RS-422 Differential Line Driver
19G	AM26LS32CN	Quad RS-422 Differential Line Interface