HARDWARE DESCRIPTION

THE CORVUS CONCEPT

CORVUS SYSTEMS

DISCLAIMER OF ALL WARRANTIES & LIABILITIES

Corvus Systems, Inc. makes no warranties, either expressed or implied, with respect to this manual or with respect to the software described in this manual, its quality, performance, merchantability, or fitness for any particular purpose. Corvus Systems, Inc. software is sold or licensed "as is." The entire risk as to its quality or performance is with the buyer and not Corvus Systems, Inc., its distributor, or its retailer. The buyer assumes the entire cost of all necessary servicing, repair, or correction and any incidental or consequential damages. In no event will Corvus Systems, Inc. be liable for direct, indirect, incidental or consequential damages, even if Corvus Systems, Inc. has been advised of the possibility of such damages. Some states do not allow the exclusion or limitation of implied warranties or liabilities for incidental or consequential damages, so the above limitation may not apply to you.

Every effort has been made to insure that this manual accurately documents the operation and servicing of Corvus products. However, due to the ongoing modification and update of the software along with future products, Corvus Systems, Inc. cannot guarantee the accuracy of printed material after the date of publication, nor can Corvus Systems, Inc. accept responsibility for errors or omissions.

NOTICE

Corvus Systems, Inc. reserves the right to make changes in the product described in this manual at any time without notice. Revised manuals and update sheets will be published as needed and may be purchased by writing to:

Corvus Systems, Inc.
2029 O'Toole Avenue
San Jose, CA 95131

Telephone: (408) 946-7700
TWX 910-338-0226

FCC WARNING

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. As temporarily permitted by regulation it has not been tested for compliance with the limits for Class A computing devices pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

CORVUS SYSTEMS

The Corvus Concept

Hardware Description

PART NO. : 7100-03291

DOCUMENT NO. : CCC/30-99/1.1

RELEASE DATE : February, 1983

TABLE OF CONTENTS

The Corvus Concept

Hardware Description

CHAPTER ONE

CONCEPT OVERVIEW

1.0 SCOPE OF CHAPTER

This chapter describes the modules and specifications of the Corvus Concept. The Concept was designed in a modular fashion to facilitate servicing and decrease servicing time. Its powerful software and hardware capabilities are outlined in this chapter. Other pertinent Concept publications will be referenced at the appropriate time.

1.1 CONCEPT MODULES

Concept uses the advanced and powerful features of the Motorola MC 68000 microprocessor. It is comprised of a base, a bit mapped video monitor, a detachable keyboard, and an optional floppy disk drive. It contains a standard 256 kbytes of memory with a 512 K option available from Corvus.

The 15-inch monitor is mounted to allow the operator to swivel or tilt the screen for maximum operator comfort. The screen can be operated in a vertical (portrait) or horizontal (landscape) fashion with the display area consisting of 560 x 720 dots.

The keyboard is compact and has ten programmable function keys, a numerical keypad, and SELECTRIC (tm) style alphanumeric keys. The definition of almost all keys is programmatically alterable at will.

The base contains a power supply, fan and a removable tray. The tray contains the processor board, memory board, speaker and calendar battery. There are two RS-232C ports and an Omninet transporter in addition to four 50 pin I/O connectors on the processor board. The memory board contains 256 or 512 Kbytes in 64K dynamic RAMS. This modular design allows easy access to and replacement of the electronic subassemblies of the Concept.

1.1.1 MONITOR

The Ball HD series of monitor is a high resolution display (35MHz). Under DMA, approximately 55K of main memory is ported directly to the 720 pixel by 560 pixel screen. Data is transmitted to the screen at approximately 33 Mbits/sec.

1.1.2 KEYBOARD

The keyboard is compact and manufactured by Keytronics under Corvus specifications. It has Selectric style alphanumeric keys.

- o Selectric Syle Alphanumeric Keys
- o Cursor Movement Keys
- o Numeric Keyboard
- o 10 Programmable Softkeys
- o Special Function Keys
- o Removable Coiled Keyboard Cable

Key definition is software alterable, allowing foreign or special keys to be implemented.

1.1.3 FLOPPY DRIVES

The Concept offers an optional 8-inch floppy drive manufactured by Tandon Corporation.

- o 250 Kilobyte Formatting Capacity which will be expanded to 500 Kilobyte in the future.
- o Single Sided / Single Density IBM 3740 Format

1.1.4 POWER SUPPLY

Power is provided to the Corvus Concept by an AC/DC power supply located in a compartment of the base unit. See power specification outline for further specifications.

1.2 CONCEPT SPECIFICATIONS

1.2.1 OPERATING SYSTEM SOFTWARE

- UCSD Pascal file structure
- ISO Pascal with UCSD extensions
  (native code compiler)
- FORTRAN 77 (native code compiler)
- 68000 Assembler
- EdWord (tm) Wordprocessor
- Corvus LogiCalc (tm) electronic spread sheet

1.2.2 MICROPROCESSOR

- Motorola MC 68000
- 32 bit data registers
- 24 bit memory address bus
- 16 bit data bus

1.2.3 MEMORY

- 256 KB standard
- 512 KB optional

1.2.4 INPUT/OUTPUT

- One Omninet Network Interface
- Two serial asynchronous controllers
- One clock/calendar with battery backup
- One flexible sound generator with speaker
- Two interval timers

1.2.5 WINCHESTER DISK OPTIONS

- o 5.7 MB (formatted) CORVUS Disk System
- o 10.8 MB (formatted CORVUS Disk System
- o 19.7 MB (formatted) CORVUS Disk System

1.2.6 DISKETTE DRIVE OPTIONS

- o 250 KB (formatted) 8-inch diskette drive (to be expanded to 500 KB)
- o 140 KB (formatted) 5 1/4 inch read-only diskette drive

1.2.7 BACKUP OPTION

- o 73 MB (formatted) video cassette recorder

1.2.8 VIDEO DISPLAY

- o 15" CRT (35MHZ)
- o Bit Mapped Display (720 by 560)
- o Vertical Tilt: +17 to -13 degree deflection
- o Horizontal Swivel: 90 degree
- o 720 pixel by 560 pixel screen.
- o 120 characters x 56 lines in landscape mode
- o 90 characters x 72 lines in portrait mode
- o Character Generated by Software

1.2.9 KEYBOARD

- o 91 key detached keyboard
- o Selectric (tm) style keyboard

- o 15 key numeric pad
- o 10 programable function keys
- o cursor control keys

1.2.10 OMNINET

- o 4,000 feet total network length
- o 64 total network devices
- o 1 million bps transfer rate
- o Twisted pair cable transmission medium

1.2.11 ELECTRICAL SPECIFICATIONS

- o 100/120 or 220/240 volts selectable
- o 50/60 Hz ac, 200 Watts

1.2.12 POWER SPECIFICATIONS

o Power Supply provides:

```
+5VDC  @ 8A
+12VDC @ 1.7A
-12VDC @ 1.7A
```

(60Hz, derate 10% for 50Hz)

o Processor and Memory PCB Consumption

```
6A @ 5VDC
130ma @ +12VDC (RS 232C)
170ma @ -12VDC (RS 232C)
```

o I/O Slots (Shared Availability)

```
+5VDC  @ 500mA
+12VDC @ 1.5A
-12VDC @ 1.3A
-5VDC  @ 200mA
```

1.2.13 PHYSICAL CHARACTERISTICS

	HEIGHT IN/CM	WEIGHT LBS/KG	DEPTH IN/CM	WIDTH IN/CM
VIDEO DISPLAY	14/35.6	41/18.6	15/38.1	15/38.1
BASE UNIT	4.5/11.4	24.5/11.2	15/38.1	17/43.2
KEYBOARD	3/7.6	4.7/2.1	8/20.3	17/43.2

CHAPTER TWO

THEORY OF OPERATION

2.0 SCOPE OF CHAPTER

This chapter is intended to deliver a general explaination of the theory of operation of the Corvus Concept. The description is divided between two printed circuit boards that are contained in the Concept: The Processor board and the Memory board. An in-depth description is provided in the Chapter Three.

2.1 CONCEPT PROCESSOR BOARD

The Processor board of the Concept can be divided into the following sections:

- Microprocessor
- ROM and RAM
- Data Communication Ports
- Omninet
- Calendar
- I/O Slots
- Bell, Timer etc.
- Alternate Memory Map Control
- Interrupts

2.1.1 MICROPROCESSOR

The Microprocessor is an 8 MHz MC 68000. Its address lines are buffered to go to many locations. Its data lines are unbuffered to ROM and Static RAM, but buffered to I/O, Omninet and dynamic RAM.

2.1.2 ROM and STATIC RAM

There are four 24 pin and two 28 pin sockets which can accept EPROMs and ROMs up to 64K and static RAMs. 2716s, 2732s and

6116s are some devices which can be used. The ROMs contain boot code, elementary I/O and some self test. The RAM is used to hold system variables, jump tables, etc.

2.1.3 DATA COMMUNICATION PORTS

Two RS-232C ports with independent speeds from 110 to 19200 Baud can be used for an external terminal, modem, printer etc. They share the I/O bus and interrupt structure with other I/O devices.

2.1.4 OMNINET

The processor sends commands to the Omninet transporter through the I/O bus. However, when Omninet does direct memory access (DMA) it takes over the address and data buses going to the memory board. If the MC 68000 attempts to use memory during a DMA cycle it will be held off. Typically there is at maximum one DMA cycle in 8 microseconds during which time there could be a maximum of seven MC 68000 memory accesses. DMA does not slow the MC 68000 appreciably.

2.1.5 CALENDAR

A time clock and calendar is provided. It maintains time while the power is off by using of a NICAD or lithium battery. It also shows day and month but not the year, nor does it interrupt.

2.1.6 50 PIN I/O SLOTS

I/O slots provide access to such devices as Corvus disk systems, parallel printers etc. These slots do not support DMA because they do not share the data and address busses with the dynamic memory. They are connected to the interrupt structure.

2.1.7 BELL, TIMER etc

A Versatile Interface Adapter (VIA) is used for many housekeeping functions. It has two parallel I/O ports which are used for reading and writing sixteen signals.It also has two counters and a shift register which are used for timing functions and bell control.

2.1.8 MEMORY MAPPER

The address space is divided into sections by a memory mapper PROM which examines the state of the address bus and selects the appropriate device. One input is flipflop which can select an alternate mapping.

2.1.9 INTERRUPTS

Six interrupt lines are served by an auto vectored interrupt mechanism. The highest priority (Non Maskable Interrupt NMI) is not used. The two datacomm devices, the keyboard, the VIA and Omninet each have their own interrupt vectors. The collection of data communication control lines shares the lowest vector with the 50-pin bus interrupt.

2.2 CONCEPT MEMORY BOARD

The memory board provides most of the read/write memory for the system. It functions in eight categories.

- o Horizontal Timing
- o Vertical Timing
- o RAM Timing
- o Memory Selection
- o Address Multiplexing
- o Memory Array
- o Memory Buffer
- o Video Shift Registers and Multiplexer

The horizontal counter produces high speed timing for the system, including horizontal synchronisation and blanking pulses for the video monitor. The vertical counter produces vertical synchronization and blanking pulses. RAM timing comprises RAS, MUX, and CAS times for the memory array. RAS is row address select, CAS is column address select and MUX is multiplex address. Memory selection decides which section (upper, lower or both) of the four memory banks shall be read from or written into. The video address counter provides both screen and memory refresh. Data from the memory array is read either into the memory buffers and there to the processor or Omninet, or else to the video shift registers and from there to the screen.

2.2.1 HORIZONTAL TIMING

A 16.364 MHz oscillator is counted down by 2, 16, and larger numbers to produce signals at approximately 8 MHz, 1MHz and 35 KHz. 35 KHz is the horizontal scanning frequency of the monitor. The processor board uses the 16MHz, 8MHz and 1MHz signals to clock various devices. The horizontal timing section comprises an oscillator, three counters (74S163, 74LS163 or similar), a logic array and some flip flops (74LS174) for resynchronisation of signals.

Not Horizontal Blank Previous (NHBlANKP) is clocked into the 74LS174 to produce the horizontal blanking signal NHBLANK. Note that N suffixed to a signal means that the signal is asserted when it has a low voltage.

The memory fetches data for the video shift registers during video-time and can read or write data for the MC 68000 or Omninet during not-video-time. NVIDTIMEP is clocked into the 74LS174 to produce the signal NVIDTIME, as are NLOADVIDP previous and 68K Previous. NLOADVID instructs the video shift registers to load the data from their data lines. 68K allows the memory to start a read or write cycle for the processor. Its timing is so chosen that a memory access will finish before the next video-time. At the video-time a memory access will start and will be complete by the next 68K-time. 1 CYCLE allows one video-time to be between each 68K-time. If 1 CYCLE is not true there will be one video cycle to every three 68K-times. Also during horizontal blank time there will be continuous 68K times and no video times. The 68K time is an enable signal: the memory makes an access when a 68K signal and a MC 68000 read or write request coincide. NHSYNC provides the horizontal synchronisation for the monitor.

2.2.2 VERTICAL TIMING

The vertical timing counter is made out of similar devices to the horizontal timing counter. It produces the Vertical Synchronization (VSYNC) signal to the monitor, the vertical blanking (VBLANK) signal which it combines with the horizontal blank signal to enable video to the monitor. The vertical timing counter is clocked by NHBLANK and resets itself at 60Hz unless the 60Hz jumper is grounded which causes a 50Hz rate to be used. A signal HOME clears the video address counter so that every screen begins to refresh its data at the same point.

2.2.3 RAM TIMING

The Row Address Select (RAS), Column Address Select (CAS) and multiplexing signal are produced by a 74S195 shift register. Input logic allows either video time or the MC 68000 to initiate a memory cycle. As soon as RAS time is asserted (NRASTIME is low) feedback keeps the input low until the CASTIME signal delayed twice through a 74LS174 puts the 74S195 in the "load" condition. Logic high signals are loaded for three clock times assuring that the RAM precharge times are fulfilled.

A 74LS74 flipflop pair controls MC 68000 access to the memory. The first flipflop sets following a MC 68000 memory request (RAMSEL) and the 68K signal. A memory cycle occurs until NCAS goes high, setting the second flipflop and asserting Data Acknowledge (NRAMACK). Eventually MC 68000 will disassert RAMSEL, resetting these two flipflops. Only when the first flipflop is set and the second not set is a read or a write requested by the MC 68000 allowed to happen.

2.2.4 MEMORY SELECTION

The Bank Selects and RAS, CAS and WRITE signals are generated by 74LS139, 74LS08 and 74LS32 devices. A RAS occurs on all banks together during video-time but only on one bank at a time for the MC 68000. The MC 68000 may read the upper, lower or both sections of the bank in a memory access.

2.2.5 VIDEO ADDRESS COUNTER

This counter sequentially addresses all locations of memory displayed on the monitor, including some overlap during horizontal and vertical retrace. The counter increments during horizontal display but does not increment for most of horizontal blank. The memory accesses two banks at a time, and so the counter must increment by two each video time. If four banks of memory are installed, all four banks of data are unloaded into the shift registers at once. Jumper TJ is then set to increment the counter by four.

2.2.6 MEMORY MULTIPLEXING

During video time the video address counter is selected to address the memory, otherwise the MC 68000 or OMNINET addresses the memory. The signal NMUX determines whether the lower or upper part of the address goes to the memory to be strobed by RAS or CAS. Only the lower part of the video counter is necessary to refresh the memory, but all of the counter is necessary to refresh the screen. Jumpers allow selection correctly when two or four memory banks are installed.

2.2.7 MEMORY ARRAY

The memory is divided into 4 banks of 16 (64K) RAMS each. The banks are in turn divided into upper and lower bytes of 8 (64K) RAMS each. For full video to occur at least two banks must be installed.

2.2.8 MEMORY BUFFERS

The data from the memory is latched into buffers at the end of each 68K memory cycle, and can be read from the buffers if the MC 68000 is requesting a read. If the MC 68000 is not reading, the buffers are tristate.

2.2.9 VIDEO SHIFT REGISTERS

The video data stream requires a data rate of 32 MHz. To achieve this 32 bits (two words) are loaded from 2 banks each microsecond (or four words alternate micro second if four banks are installed). The shift register is clocked at 16MHz but is split in two. Bits are taken alternately from each half during a clock cycle, thus doubling the data rate.

CHAPTER THREE

DETAILED TECHNICAL DESCRIPTION

3.0 SCOPE OF CHAPTER

The various circuits that make up the processor board and the memory board are described in detail in the following chapter.

3.1 PROCESSOR BOARD

The processor board is made up of the following elements:

- MICROPROCESSOR
- BUFFERS
- ADDRESS MAPPING
- DATA ACKNOWLEDGE
- STATIC RAM & ROM
- I/O & I/O BUFFERS
- OMNINET & MEMORY BUFFERS
- INTERRUPTS
- KEYBOARD COMMUNICATION
- OSCILLATOR

3.1.1 MICROPROCESSOR AND BUS BUFFERS

The CONCEPT is based on the Motorola MC 68000 8MHz microprocessor which has a 16 bit bidirectional data bus, a 24 bit address bus, and sixteen 32 bit internal registers. The 64 pin design eliminates the need for data and address multiplexing by giving each data and address line a seperate pin.

The address bus is buffered from the processor by three 74LS244s at locations U507, U508, U407. If the address bus is shorted together these buffers could possibly be damaged. The data bus is unbuffered to the on board ROMs and static RAMs. The data bus is buffered to the I/O ports, OMNINET, and dynamic RAMs. If for any reason the address or data bus becomes defective the processor will assert the HALT signal and abort all operations.

The control signals (WRITE, UPPER & LOWER DATA STROBE, and ADDRESS STROBE) are buffered by a single 74LS244 at location U307. The function codes lines are decoded to determine supervisor mode and interrupt acknowledge. This interrupt acknowledge is connected to the Valid Peripheral Address Pin to indicate auto vectoring interrupt mode. For further information on the processor, please refer to Motorola's 68000 users handbook.

3.1.2 ADDRESS MAPPING

An 82S181 bipolar PROM examines the address lines to produce enable signals for I/O and memory. Additional inputs are NZERO, ALTMAP, and NSUPERVISOR. The first signal (NZERO) is output from a set of gates (U409, U510) which detect that the lower address bits are at zero. If the higher address bits are at zero and (ALTMAP) is zero, then ROM0 will be selected for a power on boot.

If no device is selected, CYCLE ROM is asserted to prevent the processor from hanging up.

The I/O address space is divided into eight blocks by U605, most of which are further subdivided.

Blocks 0 - 4 (See I/O Slots Section 3.1.15)

Block 5 - Allows reading of the NMI and IRQ lines from each of the slots on a single operation.

Block 6 - Allows reading of the Clock Calendar

Block 7 - Allows reading or writing to the I/O Ports.

Block 7 can further be subdivided into:

0	NKBP	Keyboard
1	NSR0	Data Comm Port 0
2	NSR1	Data Comm Port 1

3	NVIA	Versatile Interface Adapter
4	NCALM	Clock calendar address and Strobe register
5	NOMNI	Omninet Strobe
6	NOMOFF	Reset Omninet Interrupt Flip Flop
7	NIOSTRB	External I/O ROM Strobe

3.1.3 STATIC RAM AND ROMS

There are four 24 pin sockets and two 28 pin sockets mounted on the processor board. Two sockets (locations U707 and U711) are intended to hold 2k x 8 static RAMs. The recommended static RAM device number is 6116s. The RAM can contain system variables, jump tables, and data storage during testing. The speed of the RAM can be accounted for by jumpers K4, K5. The ROM0 sockets (locations U706 and U710) can hold such devices as 2716 or 2732. This ROM pair contains the boot code, initial self test, setup data for I/O, simple keyboard map, a character set, etc. The ROM1 sockets (locations U708 and U709) are 28 pin sockets. These sockets can hold such devices as 2716, 2732, 2764 and other pin compatible ROMs and RAM. These ROMs may be used in loading Motorola's MACSbug for bringup and diagnostic purposes or for installing user's firmware.

3.1.4 DATA ACKNOWLEDGE

The 68000 is an asynchronous machine for memory and I/O. It asserts a memory request (derived from DATA STROBE and ADDRESS STROBE) and waits for memory acknowledge (DTACK). Memories and I/Os of different speeds are accommodated by delaying data acknowledge (DTACK) until the access is complete. In the Concept the DTACK is provided by either a state machine (dynamic RAM, I/O) or by the combination of the device select (ROM, static RAM) and a delay. The circuitry that is involved consist of U411, U511 and U510. The delay is from a shift register (U412). The shift register clocks in ones unless cleared by DATA STROBE being negated. It has outputs at intervals of 61 nsec up to about 490 nsec which can be selected by a jumper for each of the ROM and RAM pairs. This allows for access times of zero to about 600 nsec to be used. The delay from the state machines varies by up to a microsecond (dynamic RAM) or up to 2 microseconds (I/O). Gates U411 , U510 and U511 form the delayed signal into NDTACK (TP 20) Data Strobe DS can be seen at TP 19 and Not Address Strobe at TP 11. If the MC 68000 encounters illegal conditions (e.g. instructions), it will halt with a low at TP 14.

3.1.5 I/O AND I/O BUFFERS

An I/O request is generated when the MC 68000 addresses a particular address space. This address space is further decoded to select individual devices. Most devices are allowed sixteen locations, but the slots are allowed 256. I/O is attached only to one byte of the bus, and so all addresses are odd. All I/O except OMNINET and the clock/calendar is based on the operation of 6502 peripheral devices. These are synchronous, and expect an address to become stable at one edge of the 1MHz clock. At the other edge of the clock they generate data. The data becomes invalid soon after the clock edge.

By contrast the MC 68000 is asynchronous. To allow I/O devices and the MC 68000 to operate in their own enviroment, a state machine and a pair of latched buffers are used. When the MC 68000 writes to I/O the first flip flop (U604) accepts the I/O request at the next rising 1MHz clock edge. Data is latched into the write buffer and a data acknowledge produced on the next rising 1MHz clock edge. When the 68000 negates the data strobe the flip flops are cleared in preparation for another I/O cycle. When MC 68000 reads I/O similar latching occurs, holding the data until the MC 68000 completes its cycle, even though the data from the 6502 peripheral may have become invalid. The maximum possible I/O rate is 0.5 MHz, but is normally lower because the MC 68000 will not produce one I/O request as soon as one microsecond after another I/O request.

To select I/O the address mapper sets line NIO low. This, together with I/O acknowledge flip flop being reset, enables a 1 of 8 decoder (U605). With the inputs A9,A10, and A11 this decoder seperates I/O space into 512 byte blocks. Only half of these bytes are accessible because the I/O bus is attached only to the odd byte data bus.

3.1.6 OMNINET AND MEMORY BUFFERS

OMNINET is a self contained unit on the processor board, and can be given commands from the processor via the I/O bus. The processor places a byte of data on the bus and strobes NOMNI. The data consists of three bytes containing an address where OMNINET is to find its command in memory. The processor checks VIA port A, bit 0 to see if OMNINET is ready to receive another byte of address. OMNINET has no other connection with the processor. It talks directly to memory, preempting the processor by means of the memory arbiter (See Section 3.1.7).

OMNINET'S 6801 (U302) and monochip (U104) control Direct Memory Access. The Asynchronous Data Link Controller (ADLC, U301) takes care of the serial data transfer through a pair of transmitter and receivers (U201) to a balanced twisted pair cable (RS422). The serial transfer occurs at 1 Mbit per second. Parallel transfer by byte has a DMA rate of 125 kbytes per second.

To begin the DMA of a byte the monochip asserts DMA request (DMAREQ). This is synchronized by the arbiter which in time switches the memory address and data bus from the processor to OMNINET. The arbiter begins a memory cycle and asserts DMAGO. Following DMAGO, DMAREQ is negated. When the memory cycle is complete DMAGO is negated and the monochip sets up to accept the data into the ADLC (only on a read) and terminate the DMA cycle. The memory buses are then returned to the processor.

OMNINET produces 20 address bits. Address zero is converted into upper and lower device select before being sent to RAM. OMNINET ignores address bits 21 and up.

3.1.7 MEMORY ARBITRATION

The dynamic memory may be accessed from the MC 68000 or from the OMNINET DMA. A set of flip flops and logic arbitrate when both MC 68000 and OMNINET try to get access at the same time. The memory board requires RAMSEL before an access can start, and RAMSEL must be negated before a new access can start. NRAMACK signals that the data in an access has been processed.

3.1.8 MC68000 MEMORY ACCESS WITHOUT DMA CONFLICTS

When the MC 68000 produces an address in the dynamic memory range, 68K RAMSEL is asserted and is presented to the J input of a JK flipflop U204-11. After Address Select is asserted the Q of the JK flipflop U204-9 will become true following the next 16M clock. The Q (68KGO) is passed through an OR Gate to become RAMSEL U203-8. When the data has been processed by the memory, NRAMACK is asserted, goes through an OR gate to become NTACK U203-11, and through some gates to become NDTACK. After receiving NDTACK the MC 68000 disasserts Address Select, clearing the JK flipflop and preparing for the next memory access.

3.1.9 DMA WITHOUT MC68000 CONFLICT

OMNINET asserts DMAREQ (DMA request) which is applied to the J of the JK flipflop U104-3. At the next 16M clock the JK flipflop asserts DMAEN (DMA enable) U10. One 16M clock time later NDMAGO U204-6 is asserted which sets DMAGO2 U104-9. NDMAGO tells OMNINET that the requested DMA cycle has begun, DMAGO2 U104-9 switches the memory address and data busses to OMNINET.

DMAGO is passed through OR gate U203-10 to assert RAMACK. The assertion of DMAGO causes OMNINET to disassert DMAREQ. When NRAMACK is asserted it clears DMAEN and DMAGO, causing OMNINET to accept the data (if a read was in progress) and to complete the DMA cycle. DMAGO2 follows DMAGO one 16M clock later, switching the memory and data busses back to the MC 68000. The disassertion of DMAGO removes RAMACK, preparing for the next memory cycle.

3.1.10 COLLISIONS

If a MC 68000 RAMSELECT occurs when DMAEN or DMAGO is true, gate U205-6 will prevent 68KGO from occurring. When both DMAEN and DMAGO2 have been disasserted, the next 16M clock will cause 68KGO to be asserted and a memory cycle to begin as before.

If DMAEN is asserted while 68KGO is true, DMAGO will not be allowed to set. Moreover when NRAMACK is asserted it will clear DMAEN. After 68KGO is disasserted, the next 16M clock will allow DMAEN to set, beginning a DMA access now that the conflict has been removed.

3.1.11 DATA COMMUNICATIONS

Both serial data communications ports are able to communicate on RS232 data lines at baud rates from 110 to 19200, with all types of parity and with selectable word sizes. The receive and the transmit functions can be interrupt generating or not interrupt generating at will.

The UART Baud generator requires a 1.818 MHZ frequency which is obtained by dividing 16.364 MHz by 9 using a 74LS161 IC (U212). If 16.364 MHz is not available, a crystal oscillator may be insert- in the same location (U212).

Motorola's 1488 and 1489 devices (U213, U214, U314 and U415) are used to transmit and receive the + and - 12V RS 232 voltage levels. A 470 pF capacitor is used at each RS232 line for speed control. The keyboard interface is driven by a Schmitt trigger inverter (U115) which is also used to receive keyboard data.

Three Control lines on each port are received: Data Set Ready, Clear to Send and Data Carrier Detect which can be used for handshaking or with a modem. There are three Control lines output: Data Terminal Ready (DTR), Request To Send (RTS) and Rate Select (CH). DTR and RTS are functions of the UART and are controlled by setting the UART command and control registers. CH is a bit for each port on the VIA Port B and is usually used for selecting between high and low speed on dual rate modems.

3.1.12 INTERRUPTS

Although the processor can be run without interrupts, most of the I/O devices can cause interrupts so that an efficient interrupt driven operating system can be used. Because 6502 style I/O devices were used, which cannot produce vectors, the auto vector mode of interrupt is used. The highest priority interrupt, level 7 or Non Mashable Interrupt NMI is not used or connected except on a debug station when a software monitor is installed. The user does not have access to NMI. Interrupt Acknowledge is decoded in the following manner:

INTERRUPT LEVELS

PRIORITY LEVEL	SIGNAL NAME	DEVICE
7	NOT USED	
6	NKEYINT	KEYBOARD
5	NTIMINT	VIA TIMER
4	NSR0INT	RS232 PORT 0
3	NOMINT	OMNINET
2	NSR1INT	RS232 PORT 1
1	NIOCINT	DATACOMM CTRL/ 50 PIN SLOTS

Note that on priority level 1, the datacomm controls and 50 pin I/O slot interrupts share this interrupt. The interrupt priority level can be set to any of the seven levels. Interrupts that are below the current level will not be served until the interrupt level is dropped to that level or below. An interrupt raises the priority level to its own level. The return-from-interrupt sets the priority to what it was before the interrupt. If the priority is set to 7, no interrupt can occur, thus allowing critical code sections to complete without fear of interruption. For further details, refer to the Motorola MC68000 Users Manual.

The Keyboard and Data Communications devices are 6551 UARTS from Synertek, Rockwell or MOS Technology. The Timer is part of a 6522 Versatile Interface Adapter (VIA) from the same manufacturers. Additional information about these devices can be found in the Synertek Data Catalog.

1. KEYBOARD INTERRUPTS

   The Keyboard UART (U310) acts as a receive only UART except during some testing operations. Each time it receives a new character, it causes a level six interrupt. Most key strokes cause the keyboard software driver to sent a code to the current program. Key releases cancel automatic key repeat or remove qualifies such as a shift and control.

2. TIMER INTERRUPTS

   To aid in processing timing (including key repeat timing), one of the counters in the VIA (U313) is used. This counter interrupts every 50 microseconds with a priority level 5 interrupt. None of the other interrupt possibilities of the VIA are used.

3. DATA COMMUNICATION PORT 0

   UART U311 can be set up to interrupt on receiving or transmitting a character. It is intended for use with a terminal or modem, but can be configured for any RS232 function at a variety of Baud rates and parity selections Handshaking by hardware lines (Data Communication Control Lines) is covered in section 3.1.11. Each time a character is received or has been transmitted, a priority level 4 interrupt occurs.

4. DATA COMMUNICATION PORT 1

   The UART U312 is similar to that of the UART for Data Port 0, although it is primarily provided for driving serial printers. It generates a priority level 2 interrupt.

5. OMNINET

   Whenever OMNINET finishes an operation, it generates a priority level 3 interrupt. Unfortunately, Omninet cannot turn the interrupt off so NOMOFF must be sent at the end of the interrupt process to turn the interrupt off (U304). The interrupt setting operation takes several milliseconds and care must be taken not to respond to the same interrupt more than once.

3.1.13 CALENDAR

A battery backed up calendar is provided, to indicate time and date from tenths of seconds through months. Although leap year can be set to allow February to have 29 days, the calendar does not contain a years register. The battery may be an on board lithium or a tray mounted NICAD. The calendar is very similar to a watch circuit. To read or write to the calendar an address must be written to NCALM (calendar mode), followed by an address with a strobe written to NCALM. Next, the read or write is asserted by the signal NCALRW (calendar read/write). Finally, a zero address with no strobe is written to NCALM. All of this is necessary to satisfy the calendar circuit timing requirements.

3.1.14 BELL, TIMER, VIA

The bell is a transistor driven speaker, which is driven by the shift register in the Versatile Interface Adapter (VIA). The shift register rate is determined by one of the VIA timers, and the waveform by the data loaded into the shift register. The duration, pitch and timbre of the bell are determined by the bell driver and user program.

The other timer is used as a system resource. Its basic use is to perform the key wait and repeat timing. When a character key is pressed its code is used, and then after a wait of half a second the code is repeated at five times per second. If the FAST key is pressed together with any character key, that character code is immediately repeated at 15 times per second.

Three VIA inputs not so far mentioned are the two boot switches and the orientation switch. These are read on VIA port B bits 6, 7 and 3. The boot switches indicate whether OMNINET, a local hard disk, or floppy drive is to be used as the boot device. The orientaion switch is needed to indicate whether the screen is to be used horizontally or vertically. Three outputs from the VIA are VIDOFF, VA17, VA18. VIDOFF must be zero to turn the display on. VA17 and VA18 are normally zero for display. If they are not zeroed then other than normal areas of memory are displayed.

3.1.15 I/O SLOTS

Compatability with existing hardware was a major design goal. OMNINET and two RS232 ports are built in. Four fifty-pin board edge connector slots are provided to handle other local I/O, such as Corvus floppy drives or local disk systems. The 50-pin slots are very similar to the APPLE II (tm) I/O slots. The I/O slots do not support DMA or read 6502 code. The Figure below shows the slot pin descriptions.

I/O SLOT PIN DESCRIPTIONS

	CONCEPT	APPLE II tm		CONCEPT	APPLE II tm
PIN	SIGNAL NAME	EQUIVALENT	PIN	SIGNAL NAME	EQUIVALENT
1	NIOX	I/O SEL	26	GND	GND
2	A1	A0	27	---	DMAIN
3	A2	A1	28	---	INTIN
4	A3	A2	29	NNMI-X	NMI
5	A4	A3	30	NIRQ-X	NIRQ
6	A5	A4	31	NRESET	RES
7	A6	A5	32	---	NINH
8	A7	A6	33	-12V	-12V
9	A8	A7	34	-5V	-5V
10	A9	A8	35	---	---
11	A10	A9	36	---	7MHz
12	A11	A10	37	Q3	Q3
13	A12	A11	38	NIM	PHI 0
14	A13	A12	39	---	USER1
15	A14	A13	40	1M	PHI 1
16	A15	A14	41	NDEVX	NDEV SLCT
17	A16	A15	42	IOD7	D7
18	NWRITE	R/W	43	IOD6	D6
19	---	---	44	IOD5	D5

20	NIOSTB	I/O STROBE	45	IOD4	D4
21	---	NRDY	46	IOD3	D3
22	---	NDMA	47	IOD2	D2
23	---	INTOUT	48	IOD1	D1
24	---	DMAOUT	49	IOD0	D0
25	+5V	+5V	50	+12V	+12V

3.1.16 I/O Control and Slot Interrupts

The 6551 does not handle the data communications control lines correctly and so these are checked externally to the UART. Although these lines were originally for use with MODEMS, they are often used for handshaking of slow serial devices such as printers. Because these lines are expected to be changing infrequently the following 'trick' was played to minimize hardware. The Data Carrier Detect (DCD), Clear to Send (CTS), and Data Set Ready (DSR) lines from both serial ports were fed into a parity generator U414. When any one line changes the parity output will change, causing NIOCINT to change, causing an interrupt. The control lines can then be read from the VIA port A to determine which line has changed.

To clear the interrupt IOX (VIA Port A bit 7) is toggled. Note that this scheme does not detect two lines changing at the same time. There are two other lines feeding into the parity generator. One is RAMINT which is for future expansion for interrupts from the RAM board e.g. possibly RAM parity interrupt. The other input is NSLOTINT U603, U703 which signifies that an I/O slot NMI or IRQ has interrupted. To determine which line has interrupted it is necessary to read NSLTSTAT (I/O Slot interrupt status) U702. The usual way to clear this interrupt is to read the I/O Slot causing the interrupt, depending on the I/O card in the slot. If the I/O card is likely to assert its interrupt for a long time, IOX could be used to prevent redundant interrupts from NIOCINT. NIOCINT cuases a priority level 1 interrupt. It is possible that two inputs to the exclusive or could change simultaneously, thereby not producing an IOCINT. To overcome this, IOX can be toggled occasionally. The interrupt routine should then inspect all inputs to see if any change has occurred, and then toggle IOX, in just the same way as in a normal IOCINT interrupt.

3.2 MEMORY BOARD

The components of the Corvus Concept have been selected with reliability and speed in mind. The Memory board of the Concept (256K or 512K) also provides a sophisticated timing scheme which provides both Memory timing and Video timing.

3.2.1 OSCILLATOR

The oscillator is a self contained crystal oscillator at a frequency of 16.364MHz. This frequency is used as the base frequency and is divided down to fit the system needs.

3.2.2 HORIZONTAL COUNTER

The horizontal counter divides by 472 (8 x 59) to produce a horizontal sync pulse at 34.669 KHz. At this frequency it also produces a horizontal blanking pulse, which when combined with vertical blanking pulse turns off the video during retrace. The first stages of the counter produce signals at approximately 8MHz, 4MHz, 2MHz, and 1MHz which are used to produce the timing signals VIDTIME, LOADVID, 68K. Memory accesses occur synchronously with the edges of video time (VIDTIME). A video access always occurs following the assertion of VIDTIME, and a MC 68000 memory access occurs (when required) following the negation of VIDTIME. VIDTIME is a 1.02275 MHz signal, asserted and negated for 488.87 ns. This allows lax timing for a RAM with a minimum cycle time of 450 ns.

Towards the end of the video time a load video data (LOADVID) pulse occurs, which loads data from the RAM into the video shift registers. After this a pulse occurs to enable the start of a RAM access from the MC 68000.
Video times occur only during display time,
not during horizontal retrace. If four banks of RAM are installed instead of two, alternate video times are removed and replaced by 68K enables. In this case the video shift registers are loaded with 64 bits of data instead of 32 bits. The counting elements are 74163s, the first a (U202) Schottky the other two low power Schottky (U203 and U204). The clear line is not used so 74161s could be substituted.

The outputs of the 163s are decoded by an 82S153s (U201) logic array. A 14L6 PAL could be substituted. The delay through the array can be 40ns, and so to provide timing integrity the signals are resynchronized by a 74LS174 (U401). At count 470 signal Not Horizontal Terminal Count Previous (NHTCP) is asserted. One count later NHTC is asserted and applied to the synchronous load pins of the 74163s. At the next count the 74163s load to zero. NHTC is also an input to the logic array where it and NHTCP suppress N68KP at the terminal count. Not horizontal blank (NHBLANK) is an input to the logic array where it maintains NHBLANKP until disasserted by an internal decode. NHSYNC is buffered by a Schottky inverter (U105) to the monitor cable.

3.2.2 VERTICAL COUNTER

The vertical counter circuitry is very similar to the horizontal section. The 16MHz oscillator output is applied to three counters. The output of these counters are used as input to a logic array labled (VCTR) at location U301. This circuitry provides vertical synchronization (VSYNC), and the vertical blanking pulse (VBLANK). The logic array is clocked by the signal (NHBLANK) and resets itself at a rate of 50Hz or 60Hz. The reset of 50Hz is selected when a jumper is grounded. The signal (NCOUNT) is output to the video address counter. This clears the video address counter so that every screen becomes refreshed at the same point in time.

3.2.3 RAM TIMING

The row address select (RAS), column address select (CAS), addressing multiplexing signal (MUX) are produced by a 74S195 shift register (U405). Input logic allows either video time or the MC 68000 to initiate a memory cycle. As soon as RAS time is asserted (NRASTIME is low) feedback keeps the input low until the CASTIME signal (twice delayed through a 74LS174 U505) puts the 74S195 into a load condition. Logic high signals are loaded for three clock times assurring that the RAM precharge times are fulfilled. A pair of 74LS74 flip flops control 68000 memory acesses. The first flip flop sets following a MC 68000 memory request and the 68K signal. A memory cycle occurs until NCAS goes high, setting the second flip flop and asserting data acknowledge (NRAMACK). Eventually MC 68000 will disassert RAMACK, resetting these two flip flops. Only when the first flip flop is set and the second not set is a read or write requested by the MC 68000 allowed to happen.

3.2.4 VIDEO ADDRESS COUNTER

This counter sequentially addresses all locations of memory displayed on the monitor, including some overlap during horizontal and vertical retrace. The counter increments during horizontal display but does not increment for most of horizontal blanking. The memory accesses two banks at a time, and so the counter must increment by two each video time. If four banks of memory are installed it is optional to load four banks of data into the shift registers at once. Jumper TJ may then be set to increment the counter by four.

3.2.5 MEMORY ACCESS CONTROLLER

Flipflop U403 and gates U402 control whether th MC 68000 has access to the memory. If "68k" and ram access request RAMSEL are both present and datastrobe (DS) is active, the first flipflop U403-5 sets. This begins a RAS/CAS cycle the end of which the data buffer latches the data, irrespective of whether the operation is a read or a write At this time the flipflop U403-9 assserts RAMACC which becomes DTACK(data acknowledge) for the MC 68000. No new MC 68000 RAM accesses can occur until RAMSEL has been disasserted, although video accesses will continue. The disasertion of RAMSEL resets the flipflops U403 ready for a new access.

3.2.6 ADDRESS MULTIPLEXER

The MC 68000 (or OMNINET) address is fed to the board through connectors J4. Video addresses are provided by the video address counter. Both sets of addresses are multiplexed by the MUX signal to produce a row address (strobed by RAS), and a column address (strobed by CAS). The multiplexers are 74ls257 U701, U702, U801, U802 and 74ls151 U601. This last has jumper TX which is pulled high when two banks of memory are installed, and grounded when four banks are present. The RAS addresses are fast moving and are all exercised within 2 us . When four banks are installed the address bit 2 is used for bank switching and not as part of RAM address line RA1.

3.2.7 DATA BUFFER

The data buffer comprises 74ls373s Uxxx..Uxxx, for parallel data buffering. The data is held from the trailing edge of CAS until the leading edge of the next CAS. It can be read at any time by the MC 68000 or Omninet device. The MC 68000 waits until after the DTACK has been received before it reads the data on the bus, by which time the memory board may already be performing a video cycle. The data between the memory and processor uses flat cable connector J5.

3.2.8 VIDEO SHIFT REGISTERS

The video shift registers comprise 74S299 U106, U206 and 74ls299 U306..U806 for video data. the data is shifted at 16 MHz in two pairs of registers. The output of one register is fed directly into a 74S157 multiplexer U104 which is switched betwwen two inputs at a 16 MHz rate. The other register feeds a 74S112 flipflop U103 which delays the data by 30 ns and then feeds it to the multiplexer. This makes a data rate of 32 MHz.

3.2.9 BANK SWITCHING

The 74ls139 decoders U605 and U804 select which bank is to be written to or read from. If there are two banks, jumper TB is grounded and address RA1 switches between banks. If there are four banks, jumper TB is attached to RA2 and switches between bank pairs. For a read two halves of gate U604 use UDS and LDS to decide whether upper, lower or both bytes should drive the data bus. U804 decides which bank shall be driven by RAS. U805 allows a bank select or video time to let RAS go to the banks. U705 combines a RAS enable with a RASTIME to drive the appropriate RAS lines.

3.3 PROCESSOR SIGNAL DESCRIPTION

After isolating the failure down to the processor board, further troubleshooting may be desired. With the aid of flow charts, timing diagrams, and test points a technician may troubleshoot down to a defective component. NOTE: The processor board consists of multi-layers. Damage wiil occur to the board if proper soldering skills are not followed. The processor board contains 24 test points of various signals throughout the board (see figure x.x).

Subject. Brief Corvus CONCEPT Hardware Description

Rev Lvl.	01	07-01-82	M. Cook (from Hardware Manual)
	02	08-25-82	L. Franklin
	03	09-21-82	L. Franklin (update I/O slot addresses)
	04	11-08-82	L. Franklin (update VIA control registers)

This Technical Note defines the basic hardware I/O mapping for the Corvus CONCEPT.

7.1 Memory and I/O Mapping

The following is a memory and I/O map of the Corvus CONCEPT workstation. The map is partially decoded so that in some cases several addresses access the same location. The CONCEPT uses memory from 0 to $0FFFFF. I/O is in the range $030000 to $03FFFF. In the boxes are recommended addresses. "x" means "don't care". All addresses are hexadecimal even if no $ is shown. Actual addresses are shown, with possible addresses in parentheses.

```
+------------+----------+----------+----------+ +---------------+
:            :          :          :          : :               :
: Static RAM :   ROM 0  :   ROM 1  :    I/O   : :  Dynamic RAM  :
:            :          :          :          : :               :
:    000008- : 000000-  : (020000- :  030000- : : 080000-0FFFFF :
:    000FFF  : 000007   :  02FFFF) :  03FFFF  : :               :
:            : 010000-  :          :          : :               :
:            : 011FFF   :          :          : :               :
:   (000008- : (010000- :          :          : :               :
:    00FFFF) : 01FFFF)  :          :          : :               :
+------------+----------+----------+---------V+ +---------------+
                                             :
+--------------------------------------------+
:
:
V
```

```
|
V
+---------------------------------------------------------------+
| I/O - address bits 12,13,14,15 = x                            |
|     - addresses must be odd                                   |
+--------+------+------+------+------+------+--------+----------+
|        |  IO1 |  IO2 |  IO3 |  IO4 | SLOT | CALENDR|   I/O    |
|   DEVS |  ROM |  ROM |  ROM |  ROM | STAT |   R/W  |   PORTS  |
+--------+------+------+------+------+------+--------+----------+
|    I/O |      |      |      |      |      |        |  RS-232, |
|   slot |      |      |      |      |      |        |    keys, |
|   regs |      |      |      |      |      |        |     etc. |
+--------+------+------+------+------+------+--------+----------+
|   300xx| 302xx| 304xx| 306xx| 308xx| 30Axx|   30Cxx|     30Exx|
|   301xx| 303xx| 305xx| 307xx| 309xx| 30Bxx|   30Dxx|     30Fxx|
+-------V+-----V+-----V+-----V+-----V+-----V+--------+---------V+
        |      |      |      |      |      |                   |
+-------+      |      |      |      |      |                   |
|+-------------+------+------+------+      |                   |
||+----------------------------------------+                   |
|||  +---------------------------------------------------------+
|||  |
|||  V
|||  +----------------------------------------------------------+
|||  | I/O Ports                                                |
|||  +------+------+------+------+-------+-------+-------+------+
|||  |      |      |      |      |Calendr|       |Omninet|      |
|||  |  Key-| Dcomm| Dcomm|      | ALTMAP|Omninet| I'rupt|      |
|||  | board|    1 |    2 |   VIA| Volume| Strobe|    Off|IOSTRB|
|||  +------+------+------+------+-------+-------+-------+------+
|||  |      |      |      | 30F6x|       |       |       |      |
|||  | 30F0x| 30F2x| 30F4x| 30F7x|  30F8x|  30FAx|  30FCx| 39FFF|
|||  +------+------+------+------+-------+-------+-------+------+
|||
|||  +----------------------------------------------------------+
||+->| Slot Status                                              |
||   +----------+-----+-----+-----+-----+-----+-----+-----+-----+
||   |      Bit |   0 |   1 |   2 |   3 |   4 |   5 |   6 |   7 |
||   +----------+-----+-----+-----+-----+-----+-----+-----+-----+
||   |     Type | NNMI| NNMI| NNMI| NNMI| NIRQ| NIRQ| NIRQ| NIRQ|
||   +----------+-----+-----+-----+-----+-----+-----+-----+-----+
||   |     Slot |   1 |   2 |   3 |   4 |   1 |   2 |   3 |   4 |
||   +----------+-----+-----+-----+-----+-----+-----+-----+-----+
||   | When an I/O slot NMI or IRQ interrupt occurs:            |
||   |   1.  SlotINT causes an interrupt (level 1)              |
||   |   2.  Slotstat is read to find out which slot            |
||   |       interrupted and whether it was an NMI or IRQ.      |
||   | NOTE: I/O SLOT DMA is not supported.                     |
||   +----------------------------------------------------------+
||
VV
```

```
VV
::
::     +-----------------------------------------------------------+
:+-->: ROM Mapping of I/O Slots                                   :
:      +--------+-----------+-------+-------+-------+-----+-------+
:      :  Slot  :           : Byte 0: Byte 1: Byte 2:  ...: Byte n:
:      +--------+-----------+-------+-------+-------+-----+-------+
:      :    1   : Apple II  :  C100 :  C101 :  C102 :  ...:  C1FF :
:      :        : CONCEPT   : 30201 : 30203 : 30205 :  ...: 303FF :
:      +--------+-----------+-------+-------+-------+-----+-------+
:      :    2   : Apple II  :  C200 :  C201 :  C202 :  ...:  C2FF :
:      :        : CONCEPT   : 30401 : 30403 : 30405 :  ...: 305FF :
:      +--------+-----------+-------+-------+-------+-----+-------+
:      :    3   : Apple II  :  C300 :  C301 :  C302 :  ...:  C3FF :
:      :        : CONCEPT   : 30601 : 30603 : 30605 :  ...: 307FF :
:      +--------+-----------+-------+-------+-------+-----+-------+
:      :    4   : Apple II  :  C400 :  C401 :  C402 :  ...:  C4FF :
:      :        : CONCEPT   : 30801 : 30803 : 30805 :  ...: 309FF :
:      +--------+-----------+-------+-------+-------+-----+-------+
:      : IOSTRB - only available in 16 locations, not 2k          :
:      +--------+-----------+-------+-------+-------+-----+-------+
:      :        : Apple II  :  CFF0 :  CFF1 :  CFF2 :  ...:  CFFF :
:      :        : CONCEPT   : 39FE1 : 39FE3 : 39FE5 :  ...: 39FFF :
:      +--------+-----------+-------+-------+-------+-----+-------+
:      : The initial $Cxxx of APPLE is replaced by a $30xxx.      :
:      : The lower three nibbles are shifted left and 1 added.    :
:      +-----------------------------------------------------------+
:      : Note that the 6502 code will not execute on a 68000.     :
:      : However, the ROM ID and various tables may be used.      :
:      : Drivers must be written in 68000 code and included in    :
:      : the OS or attached using the ASSIGN command.             :
:      +-----------------------------------------------------------+
:
:
V
```

```
V
:
:    +-----------------------------------------------------------+
+--->: I/O Slot Registers                                        :
     : - address bit 8 = x                                       :
     +-------------+----------+----------+----------+----------+
     : I/O Register :  Slot 1 :  Slot 2  :  Slot 3  :  Slot 4  :
     +-------------+----------+----------+----------+----------+
     :      0      :  030021  :  030041  :  030061  :  030081  :
     :      1      :  030023  :  030043  :  030063  :  030083  :
     :      2      :  030025  :  030045  :  030065  :  030085  :
     :      3      :  030027  :  030047  :  030067  :  030087  :
     :      4      :  030029  :  030049  :  030069  :  030089  :
     :      5      :  03002B  :  03004B  :  03006B  :  03008B  :
     :      6      :  03002D  :  03004D  :  03006D  :  03008D  :
     :      7      :  03002F  :  03004F  :  03006F  :  03008F  :
     :      8      :  030031  :  030051  :  030071  :  030091  :
     :      9      :  030033  :  030053  :  030073  :  030093  :
     :     10      :  030035  :  030055  :  030075  :  030095  :
     :     11      :  030037  :  030057  :  030077  :  030097  :
     :     12      :  030039  :  030059  :  030079  :  030099  :
     :     13      :  03003B  :  03005B  :  03007B  :  03009B  :
     :     14      :  03003D  :  03005D  :  03007D  :  03009D  :
     :     15      :  03003F  :  03005F  :  03007F  :  03009F  :
     +-------------+----------+----------+----------+----------+
```

7.2 VIA -- General purpose I/O port (SYNERTEK 6522)

30F61		ORB, IRB Output register B, Input register B	
	0	Video off	Output
	1	Video address 17	Output
	2	Video address 18	Output
	3	Horizontal/vertical switch	Input
	4	CH Rate select DC0	Output
	5	CH Rate select DC1	Output
	6	Bcot switch 0	Input
	7	Boot switch 1	Input
30F63		ORA, IRA Output register A, Input register A handshake	
30F65		DDRB (set to 37 in boot PROM) Data direction register B (0 = in, 1 = out)	
30F67		DDRA (set to 80 in boot PROM) Data direction register A (0 = in, 1 = out)	
30F69		T1L-L,T1C-L Timer 1 latch low byte, write latch, read counter	
30F6B		T1L-H Timer 1 latch high byte	
30F6D		T1L-L	
30F6F		T1L-H	
30F71		T2L-L,T2C-L	
30F73		T2C-H	

30F75	SR Shift Register
30F77	ACR Auxiliary Control Register
30F79	PCR Peripheral Control Register
30F7B	IFR Interrupt Control Register
30F7D	IER Interrupt Enable Register
30F7F	ORA Output register A, Input register A, no handshake

0	Ready (Omninet)		Input
1	Clear to send	DC0	Input
2	Clear to 'send	DC1	Input
3	Dataset ready	DC0	Input
4	Dataset ready	DC1	Input
5	Data carrier detect	DC0	Input
6	Data carrier detect	DC1	Input
7	IOX exclusive OR of above signals for interrupt		Output

Port A -- DDRA (30F67) preset to 80
ORA (30F7F) to read/write

Port B -- DDRB (30F65) preset to 37
ORB (30F61) to read/write

Note: The inputs are used in non-latching mode. The bell speaker is controlled by the timer 2 and shift register. The interrupt timer is timer 1. For more detail see the Synertek 6522 application notes and handbook.

7.3 Omninet

$30FA1-$30FBF	Transporter register
$30FC1-$30FDF	Reset OMNINET interrupt
For Omninet operations refer to the Omninet users guide.	

7.4 Clock/Calendar/ALTMAP/Volume

$30F81	Clock/Calendar/ALTMAP/Volume
The clock/calendar is a CMOS device with 'awkward' timing. The registers of the clock/calendar are addressed by data at $30F81. Data bits 0-3 are the register address. Bit 4 is the chip enable. Bits 5, 6 and 7 must be 0. Firstly the address must be written with the chip enable = 1. Then the address must be rewritten with the chip enable = 0. Thirdly the register read or write is performed at address $30C01. Finally $10 is written to $30F81 to deselect the chip. The data to or from the chip is on bits 0 to 3.	
Bit 5 is the volume control for the bell. A zero must be written to bit whenever this register is written to, except when a quiet bell is sound. When the quiet bell has stopped sounding, a zero must again be written to this bit.	
Bit 6 selects between the CONCEPT memory mapping and an alternate memory mapping. Whenever this register is written to, this bit must be set to zero.	

7.5 Data Communication and Keyboard Registers

The keyboard and datacomm use a 6551 UART from Synertek or Rockwell.

Register	Keyboard	Data comm 1	Data comm 2
Data	30F01	30F21	30F41
Status	30F03	30F23	30F43
Command	30F05	30F25	30F45
Control	30F07	30F27	30F47

7.5.1 Status Register

Status Register Bits			
Bit	Value	Description	Comment
0	0	No parity error	self clearing
	1	Parity error	
1	0	No Framing error	self clearing
	1	Framing error	
2	0	No overrun	self clearing
	1	Overrun	
3	0	Receive register empty	no data received
	1	Receive register full	cleared by read data
4	0	Transmit register full	not ready for new data
	1	Transmit register empty	cleared by write data
5	0	DCD low	hard-wired low
	1	DCD high	
6	0	DSR low	hard-wired low
	1	DSR high	
7	0	No interrupt request	
	1	Interrupt request	
Note: Writing to the status register resets status bit 2			

7.5.2 Command Register

Command Register Bits (7 6 5 4)

7	6	5	4	Description
-	-	0	-	No parity transmitted or received, no check
0	0	1	-	Odd parity transmitted and received
0	1	1	-	Even parity transmitted and received
1	0	1	-	Mark parity transmitted, no parity check
1	1	1	-	Space parity transmitted, no parity check
-	-	-	0	Normal transmit, receive (no echo)
-	-	-	1	Echo mode (received data is retransmitted)

Command Register Bits (3 2 1 0)

3	2	1	0	Description
0	0	-	-	Tx INT disabled, RTS off, Transmitter off
0	1	-	-	Tx INT enabled, RTS on, Transmitter on
1	0	-	-	Tx INT disabled, RTS on, Transmitter on
1	1	-	-	Tx INT disabled, RTS on, Transmit BREAK
-	-	0	-	IRQ interrupt enabled from bit 3 of status
-	-	1	-	IRQ interrupt disabled
-	-	-	0	Disable receiver, disable all interrupts DTR off
-	-	-	1	Enable receiver, enable all interrupts DTR on

7.5.3 Control Register

Control Register Bits (7 6 5 4)

7	6	5	4	Description
0	-	-	1	One stop bit
1	-	-	1	2 stop bits on 7 bit words, 1 stop bit on 8 bit words
-	0	0	1	8 bit word length
-	0	1	1	7 bit word length
-	1	0	1	6 bit word length
-	1	1	1	5 bit word length

Control Register Bits (3 2 1 0)

0	16x external clock	8	1200 baud
1	50 baud	9	1800 baud
2	75 baud	A	2400 baud
3	109.92 baud	B	3600 baud
4	134.58 baud	C	4800 baud
5	150 baud	D	7200 baud
6	300 baud	E	9600 baud
7	600 baud	F	19200 baud

7.6 Interrupt Priority

Interrupt priority	
7	Non-maskable interrupt - not installed
6	Keyboard
5	Timer
4	Datacomm 0
3	Omninet
2	Datacomm 1
1	Datacomm control/Apple slots
0	Normal (no interrupt)

When an interrupt occurs the priority is automatically raised to the level of the interrupt, preventing further interrupts of that priority and below. A return-from-interrupt restores the priority at the time of the interrupt. Most interrupts are self clearing, that is, the reading of the status of the resource clears the interrupt. OMNINET interrupts must be cleared by accessing $030FC1 (NOMOFF). Data comm control line interrupts must be cleared by complementing bit 7 of VIA port A.

Subject: Corvus CONCEPT Memory Map

Rev Lvl: 01 07-21-82 L. Franklin
02 11-08-82 L. Franklin

This Technical Note discusses the Corvus CONCEPT memory map. The initial system stack pointer is defined and the procedure to change the initial system stack pointer is described.

Memory Map

The following is a memory map of the Corvus CONCEPT workstation. Primary address locations are for 256k systems. (....) address locations are for 512k systems.

```
000000 --+----------------------------------------------------+
         !                                                    !
         ! Static RAM (except 0-8)                            !
         !                                                    !
002000 --+----------------------------------------------------+

010000 --+----------------------------------------------------+
         !                                                    !
         ! Corvus CONCEPT boot PROM                           !
         !    workstation initialization                      !
         !    keyboard driver                                 !
         !    display driver                                  !
         !    local disk driver                               !
         !    floppy disk driver                              !
         !    OMNINET disk driver                             !
         !                                                    !
012000 --+----------------------------------------------------+

020000 --+----------------------------------------------------+
         !                                                    !
         ! MACSBUG (optional)                                 !
         !    uses boot PROM for I/O                          !
         !                                                    !
022000 --+----------------------------------------------------+

030000 --+----------------------------------------------------+
         !                                                    !
         ! I/O                                                !
         !                                                    !
040000 --+----------------------------------------------------+
080000 --+--------------------------------------------------+
         :                                                  :
         : Display screen buffer                            :
         :                                                  :
08E000 --+--------------------------------------------------+
         :                                                  :
         : Heap (expands upward)                            :
         :                                                  :
       --+--------------------------------------------------+
         :                                                  :
         : Unused data space ....                           :
         : .... for use by heap and stack                   :
         :                                                  :
       --+--------------------------------------------------+
         :                                                  :
         : Stack (expands downward)                         :
         :                                                  :
09E000 --+ (0AE000) -------------- initial system stack pointer +
         :                                                  :
         : Static OS code and data (expands upward)         :
         :    Loadable drivers                              :
         :    Character sets                                :
         :                                                  :
       --+--------------------------------------------------+
         :                                                  :
         : Unused code space ....                           :
         : .... for use by static OS code and programs      :
         :                                                  :
       --+--------------------------------------------------+
         :                                                  :
         : Programs (expands downward)                      :
         :                                                  :
       --+--------------------------------------------------+
         :                                                  :
         : Corvus CONCEPT Operating System                  :
         :                                                  :
0C0000 --+ (100000) ----------------------------------------+
```

After the operating system and required drivers are loaded, memory available to the user is:

memory size	code	data
256k	83k	57k
512k	275k	121k

The line dividing code space and data space is known as the initial system stack pointer. The initial system stack pointer may be adjusted to accomodate software requiring more code space or more data space. Adjusting the initial system stack pointer reinitializes (reboots) the system. Drivers are reloaded at the new initial system stack pointer, and volumes are mounted again.

Notes on Program Space Requirements

The Pascal compiler requires 82k-83k of code space. Currently, Pascal can compile on the default 256k system.

The FORTRAN compiler requires 86k of code space. Therefore, the initial system stack pointer must be readjusted in order to use the FORTRAN compiler on the default 256k system. The command "SP 9D000" adjusts the system stack pointer to allocate sufficient code space to run the FORTRAN compiler (at the expense of data space).

With very large programs, the linker may require more data space. Adjust the initial system stack pointer upward with the SP command.

Corvus LogiCalc uses data space to store model information. The table on the next page explains the relationship between memory size and Corvus LogiCalc model size.

Corvus LogiCalc Data Space Requirements

memory	SP	model size	comment
256k	9E000	997	system default
256k	A4000	1,509	practical limit
512k	AE000	2,363	system default
512k	DE000	6,459	practical limit

Display or Set the Initial System Stack Pointer Command

The command SP stands for initial system "Stack Pointer", and is used to determine where the stack pointer is to be located in system memory. There are two forms of the SP command:

The SP command without arguments displays the current setting of the stack pointer:

```
Select function: SP
sp = 0009E000
```

The SP command with a parameter sets the initial stack pointer to that value if the parameter is valid. The parameter is interpreted as a hexadecimal number.

```
Select function: SP A0000
Restarting ....
```

After the SP command sets the system stack pointer, it restarts the system. A message is issued (Restarting ....) and the operating system reinitializes by loading drivers, mounting volumes, etc.

The SP command can only be issued in the top level dispatcher. Any attempt to change the value of the initial stack pointer from a nested program results in an error message.

An attempt to set the stack pointer to an invalid value (such as an odd address or overlaying code and stack) results in an error message. In this case, the initial stack pointer value is not changed.

CORVUS DISK SYSTEM
TECHNICAL REFERENCE MANUAL

NOVEMBER 19, 1982

CORVUS DISK SYSTEM
TECHNICAL REFERENCE MANUAL

TABLE OF CONTENTS

1. DISK HARDWARE INTERFACE

1.1 General

All cable assignments are TTL.

1.2 Cable wire assignments

NAME	ORIGINATOR	FLAT CABLE WIRE
Data Bit 0	bi-directitonal	25
Data Bit 1	bi-directitonal	26
Data Bit 2	bi-directitonal	23
Data Bit 3	bi-directitonal	24
Data Bit 4	bi-directitonal	21
Data Bit 5	bi-directitonal	22
Data Bit 6	bi-directitonal	19
Data Bit 7	bi-directitonal	20
DIRC (bus dir)	drive	9
READY	drive	27
-STROBE	computer	29
-RESET	drive	31
+5 volts	drive	3,4,34
Ground drive		6,8,10,17,28,30,32
Unused	----	1,2,5,7,11-16,18,33

1.3 Cable timing

1.3.1 General case

Command initionation and computer to drive data transfer.

```
READY   ----------------+        +-------------------+        +-
                        |        |                   |        |
                        +--------+                   +--------+

-STROBE ----------+  +-------------------------+  +-------------
                  |  |                         |  |
                  +--+                         +--+

                /------------\               /------------\
DATA  ---------<              >-------------<              >----
                \------------/               \------------/

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

DIRC
```

The drive indicates its readiness to accept a command by raising the READY line. The computer then puts a command byte to the data lines and pulses -STROBE (the command byte is to be latched by the drive on the rising edge of -STROBE). Upon seeing the -STROBE pulse, the drive drops the READY line as an acknowledgement to the computer. When ready for the next command byte the drive again raises the READY line.

At the end of the command sequence, the drive will keep the READY line low until the desired operation has been performed. Upon completion of the operation, the drive will lower the DIRC line, raise the READY line and then allow the computer to read data and status information. Note that all commands consist of a write phase (during which command and data information is sent to the drive), followed by a read phase (during which status and data information is received from the drive).

Drive to computer data transfer.

```
            +--------+           +--------+                    +-
            |        |           |        |                    |
READY   ----+        +-----------+        +------------//------+

-STROBE ------+  +-----------------+  +----------------//------
              |  |                 |  |
              +--+                 +--+

            /------------\       /------------\
DATA  -----<              >-----<              >-------//----
            \------------/       \------------/

DIRC ----+                                                 +----
         |                                                 |
         +--------------------------------------------//---+
```

The drive starts a computer read sequence by lowering the DIRC line. The drive then puts a byte to the data lines and raises the ready line. The computer then pulses the -STROBE line, capturing the data on the rising edge. The drive then lowers the READY line until the next data byte is ready to send. After the last byte is transferred, the drive raises the DIRC line prior to raising the READY line.

1.3.2 Special conditions

There are two special conditions which deviate from the general cable timing information presented and must be accounted for by the computer/disk controller or by the computer/disk handler.

Case 1 -- READY line glitch after the last byte of command.

After the last command byte is received by the drive, the READY line will go high (for 20 uSEC. or less). Since this occurs prior to the completion of the command operation, it must be ignored. Since the glitch occurs while the DIRC line is high, it is easy to detect either in hardware (by gating) or in software (by the procedure shown below in Pascal pseudo-code).

```
        REPEAT UNTIL (DIRC = LOW) AND (READY = HIGH );
```

Case 2 -- DIRC line glitches after last byte of Mirror command.

After the last command byte of a Mirror command is received, the DIRC line will repeatedly alternate between high and low (while the drive talks to the Mirror). Since these changes occur while the READY line is low, they are easy to detect either in hardware (by gating) or in software (by the procedure shown below in Pascal pseudo-code).

```
        REPEAT UNTILL (READY = HIGH) AND ( DIRC = LOW);
```

Note that the two glitch cases are resolved with a single fix.

1.4 Cable connector description

17 x 2 female connector on cable, red stripe on cable is pin 1.

```
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
        | 1| 3| 5| 7| 9|11|13|15|17|19|21|23|25|27|29|31|33|
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
        | 2| 4| 6| 8|10|12|14|16|18|20|22|24|26|28|30|32|34|
        +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Pin 1 is normally designated by a square pin on the cicuit side of the interface card.

2. Disk Controller

2.1 System area

The first 2 cylinders on all drives are allocated as a system area, the second cylinder being a backup copy of the first. There are no spare tracks allowed in this region; all blocks must be good. The usage for the blocks within a cylinder are shown below.

Block 0 =	Boot Block.
Block 1 =	Disk parameter block. Spare track table (see 2.5.4) Interleave information. Step time Virtual drive track offset table (see 2.5.5).
Block 2 =	Diagnostic block.
Block 3 =	Constellation parameter block (see 2.5.3).
Blocks 4 through 5 =	Dispatcher code.
Blocks 6 through 7 =	Pipes and semaphores (see 2.5.3).
Blocks 8 through 17 =	Mirror controller code.
Blocks 18 and 19 =	LSI-11 controller code.
Blocks 20 and 21 =	Pipes controller code.
Blocks 22 through 39 =	Reserved for future use.
Blocks 40 through 59 =	Reserved for boot command.
Blocks 60 through remainder of cylinder =	Unused.

The paragraphs that follow provide brief descriptions of the content of each of the system area regions.

Boot block -- Contains Z-80 code.

Disk parameter block -- Contains disk related information as shown below:

```
+----------------------+
|  spare track table   |
|  (see 2.5.4)         |
+----------------------+
|  interleave factor   |
|  (default = 9)       |
+----------------------+
|  unused              |
+----------------------+
|  VDO table           |
|  (see 2.5.5)         |
+----------------------+
|  LSI-11 VDO table    |
|  (see 2.5.5)         |
+----------------------+
|  unused              |
+----------------------+
```

Diagnostic block -- This area contains code used by the Z-80 (in the controller) during diagnostic mode commands (format, verify, etc).

Constellation parameter block -- Contains multiplexer polling parameters and the pipe area definition, as shown below:

```
+----------------------+
|  multiplexer poll    |
|  parameters          |
+----------------------+
|  pipe area define    |
|  (see 2.5.3)         |
+----------------------+
|  unused              |
+----------------------+
```

Dispatcher code -- This area contains code used by the Z-80 (in the controller) during normal mode commands.

Pipes and semaphores -- This block contains code for the dispatcher and support utilities for pipes and semaphores, and also contains the semaphore table.

```
+----------------------+--+
| dispatcher code      |  |
| for pipes and        |  +-block 6
| semaphores           |  |
+----------------------+--+

+----------------------+--+
| semaphore table      |  |
| (see 2.5.2)          |  |
+----------------------+--+-block 7
| pipe and semaphore   |  |
| utilities code       |  |
+----------------------+--+
```

Mirror controller code -- xx.

LSI-11 controller code -- xx.

Pipes controller code -- xx.

Reserved area -- xx.

Boot extension -- Blocks 40 through 43 are currently used to support the Apple.

2.2 User area

The user area always starts at the third cylinder. The user area can be viewed as logical or physical sectors.

Logical sector numbers range from 0 to the size of the drive. The sizes are:

```
        11220 for the 6 Mbyte drive.
        21220 for the 10 Mbyte drive.
        38460 for the 20 Mbyte drive.
```

Physical sector numbers are given as head, cylinder, sector #.

The algorithm for converting logical sector numbers to physical sector numbers would be as shown below, if it were not for the system area, virtual devices and spare tracks (the real algorithm will be explained immediately following the simplified form):

```
        disk sector # = block # modulo track size.
        disk track # = block # div track size.
        disk head # = disk track # modulo surfaces.
        disk cylinder # = disk track # div surfaces.
```

Note that the disk track # is a temporary result and is not a directly addressable entity in the drive; a given block is addressed physically by sector #, head # and cylinder #.

The real algorithm for converting logical sector numbers to physical sector numbers is shown below:

```
        disk sector # = block # modulo track size.
        relative track # = block # div track size.
        physical' track # = relative track # plus system area
                            offset plus virtual drive offset.
        physical track # = physical' track # plus one for every
                           spare track preceding.'
        disk head # = physical track # modulo surfaces.
        disk cylinder # = physical track # div surfaces.
```

Where the following sizes apply:

SIZE	Model 6 Mb	Model 11 Mb	Model 20 Mb
Sectors/track	20	20	20
Surfaces (heads)	4	3	5
Cylinders	144	358	388
Total tracks per drive	576	1074	1940
Usable tracks per drive	561	1061	1923

2.3 Controller commands (numerical order)

2.3.1 Controller command notation

All of the controller commands are discribed in this section. The notation for each command is as follows: COMMAND NAME followed by (xxh : xxd), where xxh is the hex value of the command code, and where xxd is the equivalent decimal value of the same command code.

In some instances, a command code will consist of a primary code along with an additional command modifier. For these cases the notation is as follows: COMMAND NAME followed by (xxh,yyh : xxd,yyd), where xxh,yyh is the command code and the command modifier, respectively, and where and where xxd,yyd is the equivalent decimal value of the same command and command modifier.

2.3.2 Normal mode commands

2.3.2.1 Read sector (02h : 2d)

This command reads a 256 byte sector from the disk.

Send 4 bytes:

byte 1 = 02h (command).
byte 2 = logical drive #.
byte 3 = sector # (lsb).
byte 4 = sector # (msb).

Receive 257 bytes:

byte 1 = disk status.
byte 2-257 = sector data.

2.3.2.2 Write sector (03h : 3d)

This command writes a 256 byte sector to the disk.

Send 260 bytes:

byte 1 = 03h (command).
byte 2 = logical drive #.
byte 3 = sector # (lsb).
byte 4 = sector # (msb).
byte 5-260 = sector data.

Receive 1 byte:

byte 1 = disk status.

2.3.2.3 Get drive parameters (10h : 16d)

This command returns certain drive parameters.

Send 2 bytes:

byte 1 = 10h (command).
byte 2 = logical drive #.

Receive 129 bytes:

byte 1 =	status.
byte 2-32 =	ASCII text (31 bytes).
byte 33 =	firmware version.
byte 34 =	ROM version.

byte 35 =	sectors/track.
byte 36 =	tracks/cylinder.
byte 37 =	cylinders/drive (lsb).
byte 38 =	" (msb).
byte 39 =	capacity of physical drive in 512 byte blocks (lsb).
byte 40 =	capacity of physical drive in 512 byte blocks.
byte 41 =	capacity of physical drive in 512 byte blocks (msb).
byte 42-57 =	spare track list (see 2.5.4 for format).
byte 58 =	interleave factor.
byte 59-70 =	Constellation parameters.
byte 71-76 =	pipe parameters (see 2.5.3 for format).
byte 77-90 =	VDO table (see 2.5.5 for format).
byte 91-98 =	LSI-11 VDO table (see 2.5.5 for format).
byte 99-106 =	LSI-11 spare track list.
byte 107 =	physical drive number.
byte 108 =	capacity of logical drive in 512 byte blocks (lsb).
byte 109 =	capacity of logical drive in 512 byte blocks.
byte 110 =	capacity of logical drive in 512 byte blocks (msb).
byte 111-129 =	filler.

2.3.2.4 Diagnostic mode select (11h : 17d)

This command takes the drive out of normal mode and sets it to diagnostic mode.

Send 514 bytes:

byte 1 = 11h (command).
byte 2 = logical drive #.
byte 3-514 = executable Z-80 code (execution starts at first byte). This code is the monitor for diagnostic mode which interprets the rest of the diagnostic mode commands. Normally, this block is the same as block 2 of the firmware.

Receive 1 byte:

byte 1 = disk status.

2.3.2.5 Read chunk (12h or 22h or 32h : 18d or 34d or 50d)

This command reads a 128, 256 or 512 byte "chunk" from the disk. The three read chunk command formats are shown below:

Send 4 bytes:

byte 1 = 12h (command).
byte 2 = logical drive #.
byte 3 = chunk # (lsb).
byte 4 = chunk # (msb).

Receive 129 bytes:

byte 1 = disk status.
byte 2-129 = data (128 bytes).

Send 4 bytes:

byte 1 = 22h (command).
byte 2 = logical drive #.
byte 3 = chunk # (lsb).
byte 4 = chunk # (msb).

Receive 257 bytes:

byte 1 = disk status.
byte 2-257 = data (256 bytes).

Send 4 bytes:

byte 1 = 32h (command).
byte 2 = logical drive #.
byte 3 = chunk # (lsb).
byte 4 = chunk # (msb).

Receive 513 bytes:

byte 1 = disk status.
byte 2-513 = data (512 bytes).

2.3.2.6 Write chunk (13h or 23h or 33h : 19d or 35d or 51d)

This command writes a 128, 256 or 512 byte "chunk" to the disk. The three write chunk command formats are shown below:

Send 132 bytes:

byte 1 = 13h (command).
byte 2 = logical drive #.
byte 3 = chunk # (lsb).
byte 4 = chunk # (msb).
byte 5-132 = data (128 bytes).

Receive 1 byte:

```
        byte 1 = disk surface.
```

Send 260 bytes:

```
        byte 1 = 23h (command).
        byte 2 = logical drive #.
        byte 3 = chunk # (lsb).
        byte 4 = chunk # (msb).
        byte 5-260 = data (256 bytes).
```

Receive 1 byte:

```
        byte 1 = disk status.
```

Send 516 bytes:

```
        byte 1 = 33h (command).
        byte 2 = logical drive #.
        byte 3 = chunk # (lsb).
        byte 4 = chunk # (msb).
        byte 5-516 = data (512 bytes).
```

Receive 1 byte:

```
        byte 1 = disk status.
```

2.3.1.7 Boot (14h : 20d)

This command returns the contents of the specified sector of firmware on track #2.

Send 2 bytes:

```
        byte 1 = 14h (command).
        byte 2 = sector # (0-19).
```

Receive 513 bytes:

```
        byte 1 = disk status.
        byte 2-513 = boot data (512 bytes).
```

2.3.3 Diagnostic mode commands

2.3.2.1 Reset drive (00h : 0d)

This command takes the drive out of diagnostic mode and sets it in normal mode.

Send byte :

```
        Byte 1 = 00h (command).
```

Receive 1 byte:

```
        Byte 1 = 0
```

2.3.3.2 Format drive (01h : 1d)

This command formats a drive if the FORMAT switch is ON, else returns an error status.

Send 513 bytes:

```
        byte 1 = 01h (command).
        byte 2-513 = format pattern data (512 bytes).
```

Receive 1 byte:

```
        byte 1 = disk status.
```

2.3.3.3 Verify (07h : 7d)

This command performs a CRC check of every sector on the disk.

Send 1 byte.

```
        byte 1 = 07h   (command).
```

Receive n*4+2 bytes (n = errors):

```
        byte 1 = status.
        byte 2 = number of bad sectors * 4.
        byte 3 = head number of 1st bad sector.
        byte 4 = cylinder of 1st bad sector (lsb).
        byte 5 = cylinder of 1st bad sector (msb).
        byte 6 = sector number of 1st bad sector.
         .
         .
        byte n*4-1 = head number of nth bad sector.
        byte n*4+0 = cylinder of nth bad sector (lsb).
        byte n*4+1 = cylinder of nth bad sector (msb).
        byte n*4+2 = sector number of nth bad sector.
```

2.3.3.4 Read Corvus firmware (32h : 50d)

This command reads a block of data from the system area.

Send 2 bytes:

byte 1 = 32h (command)
byte 2 + head (3 bits), sector (5 bits).

Receive 513 bytes:
byte 1 = disk status.
byte 2-513 = contents of block (512 bytes).

2.3.3.5 Write Corvus firmware (33h : 51d)

This command writes a block of data to the system area.

Send 514 bytes:

byte 1 = 33h (command).
byte 2 = head (3 bits), sector (5 bits).
byte 3-514 = data (512 bytes).

Receive 1 byte:

byte 1 = disk status.

2.3.4 Semaphore Commands

The principal reason for using semaphores is to avoid a situation where two or more users are simultaneously accessing the same volume.

There is no problem if two users are merely reading from the same volume. However, if one user is writing to a volume, another user simultaneously accessing that volume may cause inconsistant data to be read. A more serious problem occurs if multiple users are writing to a file or volume at the same time.

This problem arises because the operating system in each processor has a copy of the directory for each active disk volume. The directory is usually updated on the disk only when the local operating system thinks it is necessary. Since each user can be adding, deleting, or changing files, the directory may be different in two or more processor's memory. This leads to two users writing out their files or directories and only the last user to write actually updating the directory on the disk.

To avoid this problem, there are several alternatives useful in specific instances. Read-only access to system utilities or data bases avoids the problem on shared disks. Read-write access to shared volumes can be made safe if all writes are made to existing pre-allocated files and the file is locked while any program has write access to it.

Semaphores can be used to keep two or more programs from writing to the same file or section of a file at the same time. User application programs that need shared read-write access to a data base can be configured to test the status of a semaphore before allowing access to a file. The semaphore is used to indicate that a particular file is being written to.

Each processor may, at any time, request to lock a semaphore. The request is granted if no other processor has already locked that particular semaphore. The label for the semaphore can be any eight character name that is agreed upon by the programs that wish to share access.

The Lock and Unlock commands send an eight byte name, called the semaphore, that is either placed into or removed from the semaphore table managed by the Corvus disk controller. If the semaphore table is full or if a semaphore has already been entered, a locked semaphore status is returned. The application program using the semaphores can continue to poll the semaphore table until a space is available or the desired semaphore is no longer locked. The status of the semaphore prior to each operation is also returned to provide for a full test-set or test-clear operation.

The semaphore table can be initialized by any processor, but this should only be performed on system-wide initialization or for recovery from error conditions.

2.3.4.1 Semaphore Initialize (1Ah,10h : 26d,16d)

For command explanation see the table above.

Send 5 bytes:

byte 1 = 1Ah (command).
byte 2 = 10h (command modifier).
byte 3-5 = filler.

Receive 1 byte:

byte 1 = disk status.

2.3.4.2 Semaphore lock (0Bh,01h : 11d,1d)

For command explination see the table above.

Send 10 bytes:

byte 1 = 0Bh (command).
byte 2 = 01h (command modifier).
byte 3-10 = semiphore key (8 byte name).

Receive 2 bytes:

byte 1 = disk status.
byte 2 = semaphore status.

2.3.4.3 Semaphore unlock (0Bh,11h: 11d,17d)

Send 10 bytes:

byte 1 = 0Bh (command).
byte 2 = 11h (command modifier).
byte 3-10 = semaphore key (8 byte name).

Receive 2 bytes:

byte 1 = disk status.
byte 2 = semaphore status.

2.3.4.4 Semaphore status (1Ah,41h : 26d,65d)

Send 5 bytes

byte 1 = 1Ah (command).
byte 2 = 41h (command modifier).
byte 3 = 03h (command modifier).
byte 4-5 = filler (0's).

Receive 257 bytes:

byte 1 disk status.
byte 2-257 = semaphore table (256 bytes).

See section 2.5.2 for the format of the semaphore table.

2.3.5 Pipe commands

The Corvus disk controller provides a method, called Pipes, by which different computers or programs can send data to each other. A Pipe is a FIFO (first-in-first-out) buffer that is written by a sender and is read by a receiver. Pipe commands control writing data to and reading data from the FIFO buffer. Senders and receivers may be different programs on different computers (with the Constellation network) running at different times. The only restriction on the sender/receiver combination is that the sender must send all data before the data is available to the receiver.

Before a Pipe can be utilized, it must be opened for write. The program that is sending data issues an Open Write command which creates, names, and gives a number to a Pipe.

After the Pipe is successfully opened for writing, the Pipe is ready to receive data. Pipe Write commands are used to write data to the Pipe. The Pipe Write command contains the Pipe number returned by the Open Write command. A maximum of 512 bytes may be written with one Pipe Write command.

After all the desired data has been written to a Pipe, a Close Write command is issued. The Close Write command closes a Pipe for writing and makes the Pipe available for reading.

A Pipe cannot be read until it has been written in the sequence described above. To read a Pipe, an Open Read command is issued which opens the specified Pipe for reading.

After the Pipe is successfully opened for reading, the Pipe is ready to transmit data. Pipe Read commands are used to read data from the Pipe. The Pipe Read command contains the Pipe number returned by the Open Read command. A maximum of 512 bytes may be read with one Pipe Read command.

After all the data from the Pipe has been read, a Close Read command is issued. The Close Read command closes a Pipe for reading. If all the data from a Pipe has been read when it is closed for read, the resources allocated for that Pipe are released and may be used by other Pipes.

The Pipe Initialization command initializes a Pipes area on the disk. It contains the starting disk block number and the number of disk blocks to allocate for Pipe processing.

The Purge Pipe command is used to purge unwanted Pipes by Pipe number.

The Pipe Status command returns two data blocks (512 bytes each). The first data block contains a name table of active Pipes. The second block is the pointer table, which contains state information and pointers for both ends of each active Pipe.

In a Corvus network, Pipes provide a general communications mechanism that can be used to build more sophisticated network applications. Pipes can serve as a utility that enables different computers connected to the same Corvus disk system to communicate with each other or share common peripheral equipment.

2.3.5.1 Pipe read (1Ah,20h : 26d,32d)

Send 5 bytes

byte 1 = 1Ah (command).
byte 2 = 20h (command modifier).
byte 3 = pipe number from open command (1-62).
byte 4 = 0.
byte 5 = 2.

Receive 516 bytes

byte 1 = disk status
byte 2 = pipe status
byte 3 = length of data returned (lsb).
byte 4 = length of data returned (msb).
byte 5-516 = data (512 bytes)

2.3.5.2 Pipe write (1Ah,21h : 16d,33d)

Send 5 + data length bytes

byte 1 = 1Ah (command).
byte 2 = 21h (command modifier).
byte 3 = pipe number from the open command (1-62).
byte 4 = length of data actually written (lsb).
byte 5 = length of data actually written (msb).
byte 6-n = data.

Receive 12 bytes.

byte 1 = disk status.
byte 2 = pipe status.
byte 3 = length of data actually written (lsb).
byte 4 = length of data actually written (msb).
byte 5-12 = filler.

2.3.5.3 Pipe close (1Ah,40h : 26d,64d)

Send 5 bytes:

byte 1 = 1Ah (command).
byte 2 = 40h (command modifier).

byte 3 = pipe number from the open command (1-62).
byte 4 = action code.
byte 5 = filler.

Receive 12 bytes.

byte 1 = disk status.
byte 2 = pipe status.
byte 3-12 = filler.

2.3.4.4 Pipe status (1Ah,41h : 26d,65d)

Send 5 bytes:

```
byte 1 = 1Ah (command).
byte 2 = 41h (command modifier).
byte 3 = 1 for name table status (read 512 bytes).
         2 for pipe pointer table (read 512 bytes).
         0 for both of above (read 1024 bytes).
byte 4-5 = filler (0's).
```

Receive 513 or 1025 bytes:

byte 1 = disk status.
byte 2-513 = name table status or pipe pointer table.
byte 514-1025 = pipe pointer table, if specified.

See section 2.5.3 for the formats for the pipe tables.

2.3.5.5 Pipe open write (1Bh,80h : 27d,128d)

Send 10 bytes:

byte 1 = 1Bh (command).
byte 2 = 80h (command modifier).
byte 3-10 = pipe name (8 bytes).

Receive 12 bytes:

byte 1 = disk status.
byte 2 = pipe status.
byte 3 = pipe number assigned (1-62).
byte 4 = pipe state.
byte 5-12 = filler.

2.3.5.6 Pipe area initialize (1Bh,A0h : 27d,160d)

Send 10 bytes:

byte 1 = 1Bh (command).
byte 2 = A0h (command modifier).
byte 3 = pipe area disk block number (lsb).
byte 4 = pipe area disk block number (msb).
byte 5 = pipe area size -- number of blocks (lsb).
byte 6 = pipe area size -- number of blocks (msb).
byte 7-10 = filler.

Receive 12 bytes:

byte 1 = disk status.
byte 2 = pipe status.
byte 3-12 = filler.

2.3.5.7 Pipe open read (1Bh,C0h : 27d,192d)

Send 10 bytes:

byte 1 = 1Bh (command).
byte 2 = C0h (command).
byte 3-10 = pipe name (8 bytes).

Receive 12 bytes.

byte 1 = disk status.
byte 2 = pipe status.
byte 3 = pipe number assigned (1-62).
byte 4 = pipe state.
byte 5-12 = filler.

2.4 Controler status codes

2.4.1 Normal mode command status codes

Error codes returned by the Corvus disk controller contain the type of error and error severity. Error severity is coded as follows:

Bit 7 set = Fatal error
Bit 6 set = Verify error
Bit 5 set = Recoverable error

Disk Status Codes

Non-fatal				Fatal (>= 128)						
Recoverable Error		Verify Error				Recoverable Error		Verify Error		
dc	hx	dc	hx	dec	hx	dec	hx	dec	hx	
32	20	64	40	128	80	160	A0	192	C0	Header fault
33	21	65	41	129	81	161	A1	193	C1	Seek timeout
34	22	66	42	130	82	162	A2	194	C2	Seek fault
35	23	67	43	131	83	163	A3	195	C3	Seek error
36	24	66	44	132	84	164	A4	196	C4	Header CRC error
37	25	67	45	133	85	165	A5	197	C5	Rezero fault
38	26	68	46	134	86	166	A6	198	C6	Rezero timeout
39	27	69	47	135	87	167	A7	199	C7	Drive not online
40	28	70	48	136	88	168	A8	200	C8	Write fault
41	29	71	49	137	89	169	A9	201	C9	--
42	2A	72	4A	138	8A	170	AA	202	CA	Read data fault
43	2B	73	4B	139	8B	171	AB	203	CB	Data CRC error
44	2C	74	4C	140	8C	172	AC	204	CC	Sector locate error
45	2D	75	4D	141	8D	173	AD	205	CD	Write protected
46	2E	76	4E	142	8E	174	AE	206	CE	Illegal sector address
47	2F	77	4F	143	8F	175	AF	207	CF	Illegal command op code
48	30	78	50	144	90	176	B0	208	D0	Drive not acknowledged
49	31	79	51	145	91	177	B1	209	D1	Acknowledge stuck active
50	32	80	52	146	92	178	B2	210	D2	Timeout
51	33	81	53	147	93	179	B3	211	D3	Fault
52	34	82	54	148	94	180	B4	212	D4	CRC
53	35	83	55	149	95	181	B5	213	D5	Seek
54	36	84	56	150	96	182	B6	214	D6	Verification
55	37	85	57	151	97	183	B7	215	D7	Drive speed error
56	38	86	58	152	98	184	B8	216	D8	Drive illegal address error
57	39	87	59	153	99	185	B9	217	D9	Drive r/w fault error
58	3A	88	5A	154	9A	186	BA	218	DA	Drive servo error
59	3B	89	5B	155	9B	187	BB	219	DB	Drive guard band
60	3C	90	5C	156	9C	188	BC	220	DC	Drive PLO error
61	3D	91	5D	157	9D	189	BD	221	DD	Drive r/w unsafe

2.4.2 Diagnostic mode disk status codes

2.4.3 Semaphore command status codes

SEMAPHORE STATUS CODES

DECIMAL	HEX	MEANING
0	00	Prior semaphore state = not set.
128	80	Prior semaphore state = set.
253	FD	Semaphore table full.
254	FE	Disk error.

2.4.4 Pipe command status codes

PIPE STATUS CODES

DECIMAL	HEX	MEANING
0	00	Successful pipe request.
8	08	Tried to read an empty pipe.
9	09	Pipe was not open for read or write.
10	0A	Tried to write to a full pipe.
11	0B	Tried to open an open pipe.
12	0C	Pipe does not exist.
13	0D	No room for new pipe.
14	0E	Illegal command.
15	0F	Pipe area not initialized.

PIPE STATE CODES

DECIMAL	HEX	MEANING
1	01	Open for write, file empty.
2	02	Open for read, file empty.
128	80	Full, not open.
129	81	Full, open for write.
130	82	Full, open for read.

2.5 Controller theory of operation

2.5.1 Disk operations

2.5.1.1 CRC operations

On a data read, if the first try produces no CRC error the data is returned to the computer and no further action is taken. However, if the first try produces a CRC error then one of two things will happen: 1) if the data is read successfully within 10 tries then the data is rewritten to the disk and a soft error is reported or 2) if the data cannot be read successfully within 10 tries then the data read on the last try is rewritten to the disk (along with a new CRC) and a hard error is reported.

2.5.1.2 Format operation

2.5.2 Semaphores

Semaphores provide a method for communicating between independent programs and/or systems. The disk controller provides for up to 32 named semaphores, each key (name) being from 1 to 8 characters in length.

The semaphores are implemented using a lookup table containing an 8 byte entry for each of the 32 possible semaphore keys. The presence of a key indicates that the semaphore is locked, and the absence of a key indicates that the semaphore is unlocked. Unused table entries (and unlocked semaphores) are represented by 8 bytes of blank ASCII code (20h).

The format of the semaphore table on disk (block 7) is shown below:

```
+-----------+  byte 1
|  key #1   |
+-----------+
|  key #2   |
+-----------+
|           |
=           =
|           |
+-----------+
|  key #31  |
+-----------+
|  key #32  |
+-----------+  byte 256
```

Each of the key entries has the form shown below:

```
        +--------------+
        |   1st byte   |  relative byte 1
        +-            -+
        |   2nd byte   |
        +-            -+
        |   3rd byte   |
        +-            -+
        |   4th byte   |
        +-            -+
        |   5th byte   |
        +-            -+
        |   6th byte   |
        +-            -+
        |   7th byte   |
        +-            -+
        |   8th byte   |  relative byte 8
        +--------------+
```

2.5.3 Pipes

There is a 6 byte region in the Constellation parameter block (see section 2.1) which provides pipe parameters, specifically a pipe area definition. The format for the pipe parameters is shown below:

```
        +--------------------+
        | block # of (lsb)   |
        +-  pipe names      -+
        |   table      (msb) |
        +--------------------+
        | block # of (lsb)   |
        +-  pipe pointer    -+
        |   table      (msb) |
        +--------------------+
        | number of (lsb)    |
        +-  blocks in the   -+
        |   pipes area (msb) |
        +--------------------+
```

The three pipe parameters are intially set to 1111h, 2222h and 3333h,, which indicates an uninitialized pipe area. The pipe area may be defined by the user using the Pipe Initialize command (section 2.3.5.6).

The format of the pipe area is shown below:

```
+-----------------+
|  pipe names     |  1 block
|  table          |
+-----------------+
|  pipe pointer   |  1 block
|  table          |
+-----------------+
|  pipe data      |
|  area           |
=                 =  n blocks
|                 |
|                 |
+-----------------+
```

The pipe names table contains 64 entries of 8 bytes each. The first and last names in the table are reserved for system use. The first name is "WOOFWOOF" and the last name is "FOOWFOOW".

The pipe pointer table also contains 64 entries of 8 bytes each, each entry being formatted as shown below:

```
byte 1 = pipe number.
byte 2 = starting byte address (lsb).
byte 3 = starting byte address.
byte 4 = starting byte address (msb).
byte 5 = ending byte address (lsb).
byte 6 = ending byte address.
byte 7 = ending byte address (msb).
byte 8 = pipe status (see 2.4.4).
```

Individual pipe disk space allocation

Definitions:

Active hole -- a contiguous aea of unused disk space bounded on the low address end by an open for writing pipe.

```
+-----------------+
|   open for      |
|   writing       |
|   pipe          |
+-----------------+
|   active        |  the open pipe in front of the hole
|   hole          |  can grow into this region.
+-----------------+
|   pipe          |
+-----------------+
```

Inactive hole -- a contiguous area of unused disk space bounded on the low address end by the pipe area limit, the end of a closed pipe or the end of an open for reading pipe.

```
+-----------------+
|   open for      |
|   reading or    |
|   closed pipe   |
+-----------------+
|   inactive      |  the pipe in front of the hole
|   hole          |  cannot grow into this region.
+-----------------+
|   pipe          |
+-----------------+
```

New pipe allocations are made by first examining all of the holes in the pipe area. The allocator looks for the larger of: 1) the largest inactive hole or 2) 1/2 the size of the largest active hole. A new pipe starts at the beginning of an inactive hole or at the midpoint of an active hole. All pipes grow in the same direction, by increasing address.

2.5.4 Spare tracks

There is a 16 byte region in the disk parameter block (see section 2.2.3.2) which provides for the sparing of up to 7 tracks. The format for the spare track list is shown below:

```
+--------------------+
| track number (lsb) |
+-  of 1st          -+
| spare track (msb)  |
+--------------------+
| track number (lsb) |
+-  of 2nd          -+
| spare track (msb)  |
+--------------------+
|                    |
+-                  -+
|                    |
+--------------------+
| track number (lsb) |
+-  of 7th          -+
| spare track (msb)  |
+--------------------+
| FFh          end   |
+-             of   -+
| FFh         list   |
+--------------------+
```

The first entry with a track number equal to FFFFh will indicate the logical end of the list.

2.5.5 Virtual drives

There is a 14 byte region in the disk parameter block (see section 2.1) which provides for the definition of up to 7 virtual (logical) drives. The format for the virtual drive list is shown below:

```
        +--------------------+
        | track offset (lsb) |
        +-  of 1st virtual  -+
        | drive        (msb) |
        +--------------------+
        | track offset (lsb) |
        +-  of 2nd virtual  -+
        | drive        (msb) |
        +--------------------+
        | track offset (lsb) |
        +-  of 2nd virtual  -+
        | drive        (msb) |
        +--------------------+
        | track offset (lsb) |
        +-  of 2nd virtual  -+
        | drive        (msb) |
        +--------------------+
```

An entry with a track offest equal to FFFh will indicate the absence of the corresponding virtual drive.

3.1 General

The Corvus Systems MIRROR is an inexpensive interface that adds the capability to provide backup and archival storage for the Corvus disk system. This data formatting interface converts data from a digital signal on the disk to a video signal that can be recorded on a standard video cassette recorder (VCR) at the Standard Play (SP) speed. The MIRROR is compatible with all present hardware and software -- all programs and peripherals that work with the Corvus disk system will work with the MIRROR installed.

The MIRROR allows over 100 megabytes of storage on an inexpensive, removable, and transportable media, a video cassette tape.

Redundancy and CRC error detection assure the ability to recover data. Because of redundancy and built in error checking, it is possible to recover data reliably even when errors are encountered that could not be recovered on conventional tape storage media. The result is reliable backup of mass storage. This method generally produces a few soft errors during the backup process. An error may occur in one block of a set of multiple blocks, however, by having multiple copies of each block a single good block can normally be restored.

When data is being restored to the disk, the MIRROR uses the redundant blocks to reconstruct a good block of data.

With the MIRROR, the user can make a video tape copy of an entire Corvus disk, a virtual device, or a single file (contiguous area on the disk). In approximately fifteen minutes, the contents of an entire ten million byte disk can be transferred to a standard video cassette.

The normal format creates four images of each block being backed up. Since there are four images of each block, the possibility of unrecoverable errors is minimal.

3.2 Mirror functional description

backup
restore
redundant recording
error checking
high speed search

3.3 Mirror commands (numerical order)

3.3.1 Mirror command notation

All of the Mirror commands are discribed in this section. The notation for each command is as follows: COMMAND NAME followed by (xxh : xxd), where xxh is the hex value of the command code, and where xxd is the equivalent decimal value of the same command code.

In some instances, a command code will consist of a primary code along with an additional command modifier. For these cases the notation is as follows: COMMAND NAME followed by (xxh,yyh : xxd,yyd), where xxh,yyh is the command code and the command modifier, respectively, and where and where xxd,yyd is the equivalent decimal value of the same command and command modifier.

3.3.2 Backup (08h : 8d)

Send 520 bytes:

byte 1 = 08h (command).
byte 2 = logical drive number.
byte 3 = image I.D.
byte 4 = number of 512 byte blocks to backup (lsb).
byte 5 = number of 512 byte blocks to backup (msb).
byte 6 = number of first block to backup (lsb).
byte 7 = number of first block to backup (msb).
byte 8 = format type: 0 = fast, 1 = normal, 2 = compatible (for 6 MB drive).
byte 9-520 = user defined header (512 bytes).

Receive 2 bytes:

byte 1 = disk status.
byte 2 = number of disk read errors, if byte 1 < 80h;
Mirror status, if byte 1 = FFh;

3.3.3 Restore (09h : 9d)

Send 8 bytes:

byte 1 = 09h (command).
byte 2 = logical drive number.
byte 3 = image I.D.
byte 4 = number of 512 byte blocks to restore (lsb).
byte 5 = number of 512 byte blocks to restore (msb).
byte 6 = number of first block to restore (lsb).
byte 7 = number of first block to restore (msb).
byte 8 = filler.

Receive 2 bytes:

byte 1 = disk status.

byte 2 = number of disk write errors, if byte 1 < 80h; Mirror status, if byte 1 = FFh;

3.3.4 Identify (0Ah,00h : 10d,0d)

Send 4 bytes:

byte 1 = 0Ah (command).

byte 2 = 00h (comand modifier).

byte 3 = image I.D. to read: 0 = next header, else as specified.

byte 4 = 0.

Receive 516 bytes

byte 1 = disk status.

byte 2 = image I.D., if byte 1 = 0; unused, if byte <> 0.

byte 3 = number of blocks for image (lsb).

byte 4 = number of blocks for image (msb).

byte 5-516 = image header (512 bytes).

3.3.5 Verify (0Ah,01h : 10d,1d)

Send 4 bytes:

byte 1 = 0Ah (command).

byte 2 = 01h (comand modifier).

byte 3 = image I.D. to verify.

byte 4 = 0

Receive 2 bytes

byte 1 = disk status.

byte 2 = number of disk read errors, if byte 1 < 80h; Mirror status, if byte 1 = FFh;

3.3.6 Verify error report (0Ah,02h : 10d,2d)

Send 4 bytes:

byte 1 = 0Ah (command).

byte 2 = 02h (command modifier).

byte 3 = 0
byte 4 = 0

Receive 5 + 2 * hard errors bytes:

byte 1 = number of soft errors (lsb).
byte 2 = number of soft errors (msb).
byte 3 = number of CRC failures.
byte 4 = number of disk verify errors.
byte 5 = number of hard errors.
byte 6-n = hard error block numbers (lsb,msb).

3.3.7 Remote operation select (0Ah,04h : 10d,4d)

Send 4 bytes:

byte 1 = 0Ah (command).
byte 2 = 04h (command modifier).
byte 3 = operation code (see table below).
byte 4 = 0.

Receive 1 byte:

byte 1 = command status.

Operations codes:

0 = J3 pin 2 (PLAY) pulsed low.
1 = J3 pin 3 (FAST FORWARD) pulsed low.
2 = J3 pin 4 (REWIND) pulsed low.
3 = J3 pin 5 (STOP) pulsed low.
14 = J3 pin 1 (RECORD) is set high.
15 = J3 pin 1 (RECORD) is set low.

3.3.7.1 Remote status (0Ah,05h : 10d,5d)

Send 4 bytes:

byte 1 = 0Ah (command).
byte 2 = 05h (command modifier).
byte 3 = 0.
byte 4 = 0.

Receive one byte:

byte 1 = status (see table below).

Status bits (0 is lsb, 7 is msb):

bit 0 = CRC generator status; 0 = no error, 1 = error.
bit 1 = unused.
bit 2 = unused.
bit 3 = unused.
bit 4 = J3 pin 14 (REWIND status); 1 = tape rewinding.
bit 5 = unused.
bit 6 = J3 pin 13 (FRAME SYNC); 1 pulse per every 2 frames.
bit 7 = J3 pin 11 (START OF TAPE); 0 = start of tape.

3.3.7.2 Verify retry (0Ah,06h : 10d,6d)

Send 4 bytes:

byte 1 = 0Ah (command).
byte 2 = 06h (command modifier).
byte 3 = image I.D. to verify.
byte 4 = 0.

Receive 2 bytes:

byte 1 = disk status.
byte 2 = number of tape read errors, if byte 1 < 80h;
Mirror status, if byte 1 = FFh;

3.3.7.3 Jump forward (0Ah,07h : 10d,7d)

Requires a model NV8200 Panasonic VCR and remote option.

Send 4 bytes:

byte 1 = 0Ah (command).
byte 2 = 07h (command modifier).
byte 3 = number of blocks to jump / 256 (lsb).
byte 4 = number of blocks to jump / 256 (msb).

Receive 1 byte:

byte 1 = 0.

3.3.7.4 Jump reverse (0Ah,08h : 10d,8d)

Requires a model NV8200 Panasonic VCR and remote option.

Send 4 bytes:

byte 1 = 0Ah (command).
byte 2 = 08h (command modifier).

byte 3 = number of blocks to jump / 256 (lsb).
byte 4 = number of blocks to jump / 256 (msb).

Receive 1 byte:

byte 1 = 0.

3.3.7.5 Find present location (0Ah,09h : 10d,9d)

Send 4 bytes:

byte 1 = 0Ah (command).
byte 2 = 09h (command modifier).
byte 3 = 0.
byte 4 = operation code (see table with Remote operation command, section 3.36).

Receive 8 bytes:

byte 1 = disk status.
byte 2 = image I.D.
byte 3 = image format (0-2).
byte 4 = block type found: F8h = image header, F1 = image trailer, F6h,06h = data block.
byte 5 = block number (lsb).
byte 6 = block number (msb).
byte 7 = image size in blocks (lsb).
byte 8 = image size in blocks (msb).

3.3.7.6 Find image trailer (0Ah,0Ah : 10d,10d)

Send 4 bytes:

byte 1 = 0Ah (command).
byte 2 = 0Ah (command modifier).
byte 3 = 0.
byte 4 = 0.

Receive 2 bytes:

byte 1 = disk status.
byte 2 = image I.D.

3.3.8 Restore retry (0Ch,00h : 12d,0d)

Send 4 bytes:

```
        byte 1 = 0Ch (command).
        byte 2 = logical drive number.
        byte 3 = 00h (command modifier).
        byte 4 = filler.
```

Receive 2 bytes:

```
        byte 1 = disk status.
        byte 2 = number of disk read errors, if byte 1 = 00h;
                 Mirror status, if byte 1 = FFh.
```

3.3.9 Error report for backup, restore, verify, retry (0Ch,01h : 12d,1d)

Send 4 bytes:

```
        byte 1 = 0Ch (command).
        byte 2 = logical drive number.
        byte 3 = 01h (command modifier).
        byte 4 = filler.
```

Receive 5 + 2 * hard errors bytes:

```
        byte 1 = number of soft errors (lsb).
                 (recovered errors / rebuild attempts)
        byte 2 = number of soft errors (msb).
                 (recovered errors / search misses)
        byte 3 = number of CRC failures.
                 (tape read errors / rebuild failures)
        byte 4 = number of disk verify errors.
                 (disk write errors)
        byte 5 = number of hard errors.
                 (disk read errors / bad blocks)
        byte 6-n = hard error block numbers (lsb,msb).
```

3.3.10 Partial restore (0Dh : 13d)

Send 10 bytes:

```
        byte 1 = 0Dh (command).
        byte 2 = logical drive number.
        byte 3 = image I.D.
        byte 4 = number of 512 byte blocks to restore (lsb).
        byte 5 = number of 512 byte blocks to restore (msb).
        byte 6 = destination of first block to restore (lsb).
        byte 7 = destination of first block to restore (msb).
        byte 8 = offset within image (lsb).
        byte 9 = offset within image (msb).
        byte 10 = filler.
```

Receive 2 bytes:

```
        byte 1 = disk status.
        byte 2 = number of disk read errors, if byte 1 = 80h;
                 Mirror status, if byte 1 = FFh.
```

3.4 Mirror status code

MIRROR STATUS CODES

DECIMAL	HEX	MEANING
0	00	Successful Mirror request.
1	01	Image I.D. mismatch.
2	02	Illegal restore command.
3	03	Illegal retry command (retry not enabled).
4	04	Image size mismatch.
5	05	Illegal opcode.
7	07	Start of image not found (30 second timeout).
8	08	Position error.
134	86	Tape dropout duirng playback operation (5 second timeout).

3.5 Mirror theory of operation

```
        +-----------+    +-----------+    +-----------+
        | Computer  |    |           |    |   Disk    |
        |    or     +----+  Mirror   +----+ Controler |
        |   Mux     |    |           |    |           |
        +-----------+    +-----+-----+    +-----------+
                               |
                         +-----+-----+
                         |           |
                         |    Vcr    |
                         |           |
                         +-----------+
```

VCR cable (j3) description

The control cable that connects the Mirror to the VCR has the command and status lines below:

```
        pin 1 = RECORD low (pulse).
        pin 2 = PLAY low (pulse).
        pin 3 = FAST FORWARD low (pulse).
```

```
pin 4 = REWIND low (pulse).
pin 5 = STOP low (pulse).
pin 11 = START OF TAPE status.
pin 13 = FRAME SYNC status.
pin 14 = REWIND status.
```

Format of a tape frame

Images on tape consist of a number of frames, each frame corresponding to one commmplete TV picture scan. Frames are recorded on tape at a rate of 60 per second, and have the general format shown below.

```
+----------------+
|     sync       |
+----------------+
|  image I.D.    |
+----------------+
|                |  F8h = image header,, F1h = image
|  image code    |  trailer, F6h or 06h = image
|                |  data.
+----------------+
|  image size    |  number of disk blocks.
+----------------+
|  rel block #   |  within image.
+----------------+
|      data      |
=                =  512 bytes.
|  portion       |
+----------------+
|      CRC       |
+----------------+
|  CRC reset     |
+----------------+
```

Format of a tape image

Each image on tape is comprised of groups of the three frame types (header, data and trailer) as shown below:

```
+-----------------------------------+
| image       | image      | image      |
| header      | data       | trailer    |
| frames      | frames     | frames     |
+-----------------------------------+
```

Image header

The image header consists of approximately 4 seconds of header frames (240 frames), with the data portion of each frame containing the 512 byte user I.D.

Image data

The image data is recorded in one of two formats: slow (4 copies of each disk block) or fast (2 copies of each disk block). For both formats, a given data frame contains two copies each of three disk blocks, as shown below:

```
+----------------+
|   block m      |
|   block m      |
|   block m+1    |
|   block m+1    |
|   block m+2    |
|   block m+2    |
+----------------+
```

Slow format data frames are grouped as shown below:

```
        +---------------+
        | blocks m      |  2 copies of each block.
        | thru m+2      |
        +---------------+
        | blocks m      |  2 copies of each block (4 total).
        | thru m+2      |
        +---------------+
        | blank         |  3 to n of these frames, as necessitated
        | blocks        |  by disk timing.
        +---------------+
        | blocks m+3    |
        | thru m+5      |
        +---------------+
        | blocks m+3    |
        | thru m+5      |
        +---------------+
        | blank         |
        | blocks        |
        +---------------+
        =               =
        +---------------+
        | blocks n-2    |
        | thru n        |
        +---------------+
        | blocks n-2    |
        | thru n        |
        +---------------+
        | blank         |
        | blocks        |
        +---------------+
```

Fast format data frames are grouped as shown below:

```
    +---------------+
    | blocks m      |  2 copies of each block.
    | thru m+2      |
    +---------------+
    | blocks m+3    |  2 copies of each block
    | thru m+5      |
    +---------------+
    | blank         |  3 to n of these frames, as necessitated
    | blocks        |  by disk timing.
    +---------------+
    | blocks m+6    |
    | thru m+8      |
    +---------------+
    | blocks m+9    |
    | thru m+10     |
    +---------------+
    | blank         |
    | blocks        |
    +---------------+
    =               =
    +---------------+
    | blocks n-5    |
    | thru n-3      |
    +---------------+
    | blocks n-2    |
    | thru n        |
    +---------------+
    | blank         |
    | blocks        |
    +---------------+
```

Image trailer

The image trailer consists of approximately 2 seconds of trailer frames (120 frames).

Format of images on tape (archival Mirror)

Images (each of which consists of many frames) are stored on tape sequentially, as shown below.

```
+-------------------------------------||----------------------+
| directory  | image   | image   |         | image  | image   |
|  (image    |  #1     |  #2     |         | #n-1   | #n      |
|   #0)      |         |         |         |        |         |
+-------------------------------------||----------------------+
```

The directory is maintained by external software and the directory data is read and written using the same commands as any other image. The directory contains 16 bytes of information for each of up to 32 images, as shown below:

```
+-----------+
|   date    |  2 bytes.
+-----------+
|   size    |  2 bytes.
+-----------+
|   name    |  12 bytes.
+-----------+
```

Appendix A

DISK COMMAND SUMMARY

Command	Code:Modifier	Number of Data Bytes Sent	 Received
Normal Mode Commands:			
Read Sector	02	4	257
Write Sector	03	260	1
Get Drive Parameters	10	2	129
Diagnostic Mode Select	11	514	1
Read Chunk (128 Bytes)	12	4	129
Read Chunk (256 Bytes)	22	4	257
Read Chunk (512 Bytes)	32	4	513
Write Chunk (128 Bytes)	13	132	1
Write Chunk (256 Bytes)	23	260	1
Write Chunk (512 Bytes)	33	516	1
Boot	14	2	513
Diagnostic Mode Commands:			
Reset Drive	00	1	1
Format Drive	01	513	1
Verify	07	1	4n+2
Read Corvus Firmware	32	2	513
Write Corvus Firmware	33	514	1
Semaphore Commands:			
Semaphore Initialize	1A:10	5	1
Semaphore Lock	0B:01	10	2
Semaphore Unlock	0B:11	10	2
Semaphore Status	1A:41	5	257
Semaphore Initialize (Rev A)	10:0A	5	12
Pipe Commands:			
Pipe Read	1A:20	5	516
Pipe Write	1A:21	x+5	12
Pipe Close	1A:40	5	12
Pipe Status 1	1A:41	5	513
Pipe Status 2	1A:41	5	513
Pipe Status 0	1A:41	5	1025
Pipe Open Write	1B:80	10	12
Pipe Area Initialize	1B:A0	10	12
Pipe Open Read	1B:C0	10	12

Mirror Commands:

Backup	08	520	2
Restore	09	8	2
Read Image Header	0A:00	4	516
Verify Image	0A:01	4	2
Report Errors After Verify	0A:02	4	n
Remote Operation	0A:04	4	1
Remote Status	0A:05	4	1
Verify Retry	0A:06	4	2
Jump Forward	0A:07	4	1
Jump Reverse	0A:08	4	1
Find Present Location	0A:09	4	8
Find Image Trailer	0A:0A	4	2
Restore Retry	0C:00	4	2
Restore Error Report	0C:01	4	1
Partial Restore	0D	10	2

n = number of errors   x = number of data length bytes

Appendix B

STATUS CODE SUMMARY

Error codes returned by the Corvus disk controller contain the type of error and error severity. Error severity is coded as follows:

```
Bit 7 set = Fatal error
Bit 6 set = Verify error
Bit 5 set = Recoverable error
```

Normal mode command status codes

Disk Status Codes

Non-fatal				Fatal (>= 128)						
Recoverable Error		Verify Error				Recoverable Error		Verify Error		
dc	hx	dc	hx	dec	hx	dec	hx	dec	hx	
32	20	64	40	128	80	160	A0	192	C0	Header fault
33	21	65	41	129	81	161	A1	193	C1	Seek timeout
34	22	66	42	130	82	162	A2	194	C2	Seek fault
35	23	67	43	131	83	163	A3	195	C3	Seek error
36	24	66	44	132	84	164	A4	196	C4	Header CRC error
37	25	67	45	133	85	165	A5	197	C5	Rezero fault
38	26	68	46	134	86	166	A6	198	C6	Rezero timeout
39	27	69	47	135	87	167	A7	199	C7	Drive not online
40	28	70	48	136	88	168	A8	200	C8	Write fault
41	29	71	49	137	89	169	A9	201	C9	--
42	2A	72	4A	138	8A	170	AA	202	CA	Read data fault
43	2B	73	4B	139	8B	171	AB	203	CB	Data CRC error
44	2C	74	4C	140	8C	172	AC	204	CC	Sector locate error
45	2D	75	4D	141	8D	173	AD	205	CD	Write protected
46	2E	76	4E	142	8E	174	AE	206	CE	Illegal sector address
47	2F	77	4F	143	8F	175	AF	207	CF	Illegal command op code
48	30	78	50	144	90	176	B0	208	D0	Drive not acknowledged
49	31	79	51	145	91	177	B1	209	D1	Acknowledge stuck active
50	32	80	52	146	92	178	B2	210	D2	Timeout
51	33	81	53	147	93	179	B3	211	D3	Fault
52	34	82	54	148	94	180	B4	212	D4	CRC
53	35	83	55	149	95	181	B5	213	D5	Seek
54	36	84	56	150	96	182	B6	214	D6	Verification
55	37	85	57	151	97	183	B7	215	D7	Drive speed error
56	38	86	58	152	98	184	B8	216	D8	Drive illegal address error
57	39	87	59	153	99	185	B9	217	D9	Drive r/w fault error
58	3A	88	5A	154	9A	186	BA	218	DA	Drive servo error
59	3B	89	5B	155	9B	187	BB	219	DB	Drive guard band
60	3C	90	5C	156	9C	188	BC	220	DC	Drive PLO error
61	3D	91	5D	157	9D	189	BD	221	DD	Drive r/w unsafe

SEMAPHORE STATUS CODES

DECIMAL	HEX	MEANING
0	00	Prior semaphore state = not set.
128	80	Prior semaphore state = set.
253	FD	Semaphore table full.
254	FE	Disk error.

PIPE STATUS CODES

DECIMAL	HEX	MEANING
0	00	Successful pipe request.
8	08	Tried to read an empty pipe.
9	09	Pipe was not open for read or write.
10	0A	Tried to write to a full pipe.
11	0B	Tried to open an open pipe.
12	0C	Pipe does not exist.
13	0D	No room for new pipe.
14	0E	Illegal command.
15	0F	Pipe area not initialized.

PIPE STATE CODES

DECIMAL	HEX	MEANING
1	01	Open for write, file empty.
2	02	Open for read, file empty.
128	80	Full, not open.
129	81	Full, open for write.
130	82	Full, open for read.

MIRROR STATUS CODES

DECIMAL	HEX	MEANING
1	01	File ID mismatch.
2	02	Illegal restore command (usually checksum error).
3	03	Illegal retry command (retry not enabled, or checksum error).
4	04	File size mismatch.
5	05	Illegal opcode.
7	07	Start of image not found (30 sec. timeout).
8	08	Position error.
134	86	Tape dropout during playback operation (5 sec. timeout).

The Corvus Concept

Omninet Programmer's Guide

TABLE OF CONTENTS

CHAPTER 1

Omninet Local Network

INTRODUCTION

OMNINET is a local network which operates with many popular microcomputers, ranging from the Apple II to the LSI-11. It provides the microcomputer users with cost effective installation and growth capability. Network benifits include the ability to access a common data base, share use of printers and other peripherals, inter-computer communications, and multifunctional capabilities at each station. The OMNINET local network transfers data at 1 million bits per second over an RS-422 twisted pair wire cable up to 4000 feet in length.

Each device attached to the OMNINET local network has an interface controller called a transporter. The transporter utilizes a Motorola 6801 microprocessor, a custom gate array device to control high speed direct memory access (DMA) data transfers, and associated support components.

The transporter interfaces directly to both the RS-422 serial line and the host microcomputer's memory. In order to reduce the software burden placed on the host computer, the transporter performs many of the high level network tasks which are usually the responsibility of the host computer. The transporter automatically handles message acknowledgement, error detection and retransmission, and detection of duplicate messages. Basically, the transfer of data to or from the host microcomputer is performed without intervention by host software.

The OMNINET local network is a distributed control network; hence, no master controller is required. Network control is assumed by any transporter which has a message to send as soon as the network is available.

Since OMNINET is a shared access local network that allows any device to transmit at any time, a collision avoidance scheme is implemented in the transporters. Collision avoidance is performed using a combination of two methods. First, the popular Carrier-Sense Multiple-Access (CSMA) mechanism is utilized to determine when the network is available and second, the transporter computes a randomized transmit start time to minimize the probability of two devices trying to access an available network at the same time.

Unlike many other local networks, OMNINET does not require a collision detection mechanism. This allows OMNINET to be implemented on an RS-422 twisted pair wire and eliminates the cost associated with collision detection hardware.

Error detection and retransmission activities are performed by the transporters without software intervention. Thus, the host software burden is substantially reduced while overall system performance is improved. OMNINET utilizes a positive acknowledgment method to ensure that a message has reached its destination without error. If a positive acknowledgment is not received for a message, retransmission is automatically performed by the transporter until it is successful, or a user-specified number of retries have been performed. If transmission is unsuccessful, the sending host is informed as to the nature of the error.

Device addressing within the OMNINET local network allows a message to be sent to any device attached to the network or a message to be broadcast to all devices on the network. In addition to device addressing, OMNINET supports receive sockets. A receive socket provides additional addressing capability by allowing a message to be sent to a particular buffer within the selected host microcomputer. Four sockets can be defined for each device to aid in the separation of different types of messages within the host microcomputer.

The transporter accepts two major categories of commands from the host microcomputer to control the overall flow of messages within the local network. The two major categories are:

- Send Message
- Setup Receive

The send message command transmits a message up to 2047 bytes in length to any designated host socket. The send message command includes a command code, a result address, a destination host socket number, the message address and length, and optional user control information up to 255 bytes in length. The transporter utilizes DMA to transfer the message and optional user control information without additional host software involvement. When the message is delivered, or failed to be delivered after retries, the result status code is set to reflect the status of the send message command. Some of the possible results are:

- Message delivered successfully
- Message failed after 'n' retries
- Receiving socket not set up
- Message too long for receiving socket
- Invalid socket or host device number

Setup receive commands prepare host sockets to receive incoming messages. The setup receive command includes a command code, a result address, a socket number, a data buffer address and maximum message length. The optional user control information length is also specified.

When optional user control information is sent, the user control information can be placed into a different memory address than the message data. This feature allows device drivers to place the message data directly into the user's buffer while placing the driver control information into the driver's buffer, thus eliminating the need for the driver to move the message data to the user's buffer.

OMNINET is capable of supporting a variety of shared peripherals. Shared peripherals are attached to the network using a device called a server. The Corvus disk server can be used with OMNINET to supply 5 to 80 million bytes of shared Winchester disk storage. The disk server appears as just another device on the network to OMNINET, but performs disk sharing for the other microcomputers on the network. The disk server contains both an OMNINET network interface and a Corvus disk system interface. Additional shared peripherals will be added to the OMNINET local network in the future.

Various types of network protocols can be implemented using the OMNINET local network. Corvus Systems has implemented its field-proven CONSTELLATION software protocol on the OMNINET local network. The CONSTELLATION software allows multiple microcomputers to share the Corvus disk system.

PROGRAMMING OVERVIEW

The host computer communicates with the transporter by first formatting a command control block and then giving the address of the command control block to the transporter. The method of giving the command control block to the transporter varies with the type of microcomputer being used. Chapter Three , "Transporter card installation and programming guide", describes the transporter card installation procedure and how to send a command control block address to the transporter for the specific computer type. The command control block always contains the address of a result record and may contain additional command dependent information. The result record is used to indicate the status of the command.

The transporter transmits one message at a time, but may be activated to receive up to four messages using four different socket addresses. The socket defines the memory location and length of the receive buffer. Two of the sockets, $A0 and $B0, may also have an optional user control data buffer up to 255 bytes in length. The other two sockets, $80 and $90, are not allowed to receive user control data and hence do not have a user control data buffer. The user control data buffer is always located 4 bytes after the start of the result record.

All transporter addresses and lengths must be stored with the most significant byte first and the least significant byte last. Commands are initiated from the host by setting up the command control block and initializing the first byte of the result record, the return code, to hex $FF. Next the command control block address is given to the transporter in the form of 3 bytes with the most significant byte given first. After the address of the command has been given to the transporter, the command is processed and the result record is updated. The first byte of the result record, the return code, is set to a value of other than hex $FF to indicate the the command is completed, and to indicate the status of the command. On host microcomputers that support interrupts, an interrupt is generated after the return code is updated.

TRANSPORTER COMMANDS

The transporter supports the following commands:

- SEND MESSAGE
- SETUP RECEIVE
- END RECEIVE
- INITIALIZE TRANSPORTER
- WHO AM I
- ECHO
- PEEK/POKE

The command control block and result record for each command is shown below with a description of the command.

SEND MESSAGE

COMMAND CONTROL BLOCK

COMMAND = $40	
RESULT	msb
RECORD	
ADDRESS	lsb
DESTINATION SOCKET	
DATA	msb
ADDRESS	
	lsb
DATA	msb
LENGTH	lsb
CONTROL LENGTH	
DESTINATION HOST	

RESULT RECORD

RETURN CODE
(unused)
(unused)
(unused)
USER CONTROL DATA 0 to 255 bytes of user control info. to be transmittted with message.

The SEND MESSAGE command directs the transporter to send a message to the indicated DESTINATION HOST and DESTINATION SOCKET. The message consist of DATA LENGTH bytes of data from DATA ADDRESS and CONTROL LENGTH bytes of data from the USER CONTROL DATA area. Valid values for DATA LENGTH are between 0 and 2047. If the DESTINATION SOCKET is $80 or $90 then CONTROL LENGTH must be zero. If the DESTINATION SOCKET is $A0 or $B0, then CONTROL LENGTH can be any value between 0 and 255 however, it must exactly match the value setup by the receive socket.

For a broadcast command, DESTINATION HOST number is set to hex $FF.

The message will be retransmitted, if necessary, until sucessful or until the retry count has been exceeded. The retry count has a default value of 10, but may be altered by the PEEK/POKE command. The user must not modify the message buffers or attempt to send any command to the transporter until the command has finished as indicated by the RETURN CODE.

The RETURN CODE will be one of the following values upon completion:

$00 -- Message sent successfully without retries
$nn -- Message sent successfully after nn retries
$80 -- Message was not acknowledged (retry count exceeded)
$81 -- Message DATA LENGTH too long for recerver's buffer
$82 -- Message sent to uninitialized socket
$83 -- Message CONTROL LENGTH did not match receiver's CONTROL LENGTH
$84 -- Invalid DESTINATION SOCKET number in command control block
$85 -- Invalid DESTINATION HOST # in command control block

SETUP RECEIVE

COMMAND CONTROL BLOCK

COMMAND = $F0	
RESULT	msb
RECORD	
ADDRESS	lsb
SOCKET NUMBER	
DATA	msb
ADDRESS	
	lsb
DATA	msb
LENGTH	lsb
CONTROL LENGTH	

RESULT RECORD

RETURN CODE	
SOURCE HOST	
DATA	Msb
LENGTH	lsb
USER CONTROL DATA 0 to 255 bytes of user control info. receive with the message.	

The SETUP RECEIVE command prepares the socket SOCKET NUMBER for the receiving of a single message. The data will be received at DATA ADDRESS provided the transmitted message is not longer than DATA LENGTH bytes. The DATA LENGTH field of the RESULT RECORD will contain the actual length of the data received from the SOURCE HOST. If the SOCKET NUMBER is $A0 or $B0, USER CONTROL DATA can also be received as part of the message. The sender must send exactly CONTROL LENGTH bytes of USER CONTROL DATA. If the SOCKET NUMBER is $80 or $90, then CONTROL LENGTH must be zero for both the sender and receiver.

The SETUP RECEIVE command has two replies; The first reply indicates that the command has been processed and the receive socket is ready to receive a message or an error has occured. After the receive socket is setup, a second reply occurs when a message is actually received for the setup socket.

For the first reply the RETURN CODE will be one of the following:

$84 -- Invalid SOCKET NUMBER in command control block
$85 -- Receive SOCKET NUMBER in use.
$FE -- Socket SOCKET NUMBER successfully setup to receive

If the first reply has a REASON CODE of $FE, a second reply is received when a message arrives for the setup socket. The RETURN CODE is set to $00 to indicate a message was received successfully. Before the RETURN CODE is updated, the following fields of the RESULT RECORD are updated. The SOURCE HOST is set to the transporter device address of the message originator. The DATA LENGTH field is set to the number of bytes received at DATA ADDRESS. The USER CONTROL DATA field will contain CONTROL LENGTH bytes of user control data as sent by the sender.

If a message is received in which the data portion is longer than the DATA LENGTH of the setup socket, or the user control data is not exactly the same length as specified by the CONTROL LENGTH field in the receiver's COMMAND CONTROL BLOCK, the message is rejected. The sending host is informed of the error but the receiving host is not. The receive socket remains setup as if no message was received.

END RECEIVE

COMMAND CONTROL BLOCK

COMMAND = $10	
RESULT	msb
RECORD	
ADDRESS	lsb
SOCKET NUMBER	

RESULT RECORD

RETURN CODE

The END RECEIVE command tells the transporter to release the specified SOCKET NUMBER. This disables reception of any further messages for the socket until another SETUP RECEIVE command is issued for the socket.

One of the following RETURN CODES is returned upon completion:

$00 -- Operation complete
$84 -- Invalid SOCKET NUMBER

INITIALIZE TRANSPORTER

COMMAND CONTROL BLOCK

COMMAND = $20	
RESULT	msb
RECORD	
ADDRESS	lsb

RESULT RECORD

RETURN CODE

The INITIALIZE TRANSPORTER command initializes the transporter as in a hardware reset or a power-up. All parameters are set to their default values and event counters are reset to zero.

The RETURN CODE is set to the transporter's device address.

WHO AM I

COMMAND CONTROL BLOCK

COMMAND = $01	
RESULT	msb
RECORD	
ADDRESS	lsb

RESULT RECORD

RETURN CODE

The WHO AM I command causes the RETURN CODE to be set to the transporter's device address.

ECHO

COMMAND CONTROL BLOCK

COMMAND = $02	
RESULT	msb
RECORD	
ADDRESS	lsb
DESTINATION HOST	

RESULT RECORD

RETURN CODE

The ECHO command request the transporter to send an echo packet to the DESTINATION HOST. The ECHO command is a means by which one network device can verify the presence of another network device without disturbing that device. The transporter with the DESTINATION HOST number receives the packet and acknowledges without informing the attached host computer. It is not necessary for the DESTINATION HOST to have any sockets setup for the command to operate.

One of the following RETURN CODES is returned upon completion:

$80 -- Packet was not acknowledged
$86 -- Invalid DESTINATION HOST number
$C0 -- Destination transporter acknowledged successfully

PEEK/POKE

COMMAND CONTROL BLOCK

COMMAND = $08	
RESULT	msb
RECORD	
ADDRESS	lsb
6801	msb
ADDRESS	lsb
$00=PEEK, $FF=POKE	
POKE DATA BYTE	

RESULT RECORD

RETURN CODE

The PEEK/POKE command allows the host computer to examine or alter data within the transporters internal memory.

Upon completion of the command, the RETURN CODE is $00 for a POKE request and contains the value of the byte at the 6801 ADDRESS for a PEEK command.

The hex addresses within the 6801 internal memory that may be of interest to the user are:

$00E1 -- Maximun # of retries on transmit message failure. The default value is $0A (10 decimal).

$00E2 -- Number of closing flags to send after each packet. The default value is 7. User supplied values must be greater than one for correct operation.

$00E3 -- Initial value to be used to scale the random delay in the retry logic. The default value is $07. User supplied values must be $01, $03, $07, $0F, $1F, $3F, $7F or $FF.

TRANSPORTER RETURN CODES

The RETURN CODES were described within the context of the various commands. Below is a list of all possible command RETURN CODES for ease of reference:

$00	--	Command successfully completed
$00-$3F	--	Transporter device address
$01-$7F	--	Transmit retry count
$80	--	Transmit failure (no acknowledgement after max retries)
$81	--	DATA LENGTH too long for receive buffer
$82	--	Message sent to uninitialized socket
$83	--	Transmit CONTROL LENGTH not equal to receive CONTROL LENGTH
$84	--	Invalid socket number (must be $80, $90, $A0, or $B0)
$85	--	Receive socket in use
$86	--	Invalid transporter device address (must be $00-7F, or $FF)
$C0	--	Received an acknowledgement for an ECHO command
$FE	--	Receive socket setup sucessfully

CHAPTER 2

Network Servers

DISK SERVER INSTALLATION AND PROGRAMMING GUIDE

INSTALLATION

The Corvus disk server contains a Dip switch with eight microswitches. Microswitches 1-6 are used to set the unique OMNINET device address. Normally, the disk server OMNINET device address is set to zero. The switch setting for each of the possible 64 device addresses are described in the OMNINET Installation Guide.

Microswitch number 7 is used to set a bias offset on the OMNINET trunk cable. It is recommended that the disk server be used as the network bias device. Therefore, switch number 7 should be on for the disk server and off for all other network devices. A network bias is used to reduce the effect of noise on the line when it is idle.

Microswitch number 8 is reserved for network termination. It should be off for all network devices because terminators are physically installed at both ends of the network as described in the OMNINET Installation Guide.

PROGRAMMING GUIDE

Chapter One of this OMNINET Programmer's Guide describes the commands that can be used by a computer with an OMNINET network interface card, called a transporter. These commands are used to send messages to, and receive messages from, the disk server. The disk server, in turn, interfaces with the Corvus Disk system which performs the actual disk operations.

Any computer on the OMNINET local network can communicate with the disk server by formatting a command control block and sending the message to the disk server. The Computer communicating with the disk server must first setup a receive socket to receive the response, and then send the message to the disk server. This insures that the receive socket is setup for the disk server's response. The disk server supports every disk command that the Corvus disk system supports. The content of the message sent to the disk server is basically a standard disk system command block.

For a complete list of Corvus disk commands and additional techinical information on the Corvus disk systems see the Corvus Disk Systems Technical Reference Manual. For a detailed description of OMNINET commands, control block formats, and return codes refer to the Chapter One. Refer to Chapter Three for the Transporter card installation and programming guide for the specific computer's transporter card.

Since the disk server has limited buffer space, disk commands that are longer than 4 bytes in length are performed in two transfers. The first transfer is used to send the first 4 bytes of the disk command to the disk server. The computer then waits until a "GO" response is received from the disk server. After the "GO" is received, the rest of the data for the disk command is sent to the disk server. This allows the disk server to control the use of its buffer space. The disk server is capable of queuing one request for each OMNINET network device and processing any disk command that the Corvus disk system supports.

The disk server uses socket address $B0 to receive all disk commands with a length of 4 bytes or less. For disk commands greater than 4 bytes in length, the socket address $B0 is used to receive the first 4 bytes of the disk command and the socket address $A0 is used to receive the second part of the disk command. The second part of the disk command is not sent until after a "GO" is received from the disk server. The disk server sends all of its responses to socket address $B0.

The disk server uses socket address $80 to receive broadcast messages. A send message command with a broadcast address ($FF) can be used to determine the transporter device address of the disk server.

BROADCAST COMMANDS

To broadcast a message to the disk server, the computer must perform two transporter commands as follows:

STEP 1

Use the setup receive command to setup receive socket $B0.

STEP 2

Use the send message command to broadcast a message to socket $80.

The command control block details for STEP 1, the setup receive command, are as follows:

COMMAND CODE	= $F0
RESULT RECORD ADDRESS	= (see below)
SOCKET NUMBER	= $B0
DATA ADDRESS	= Address of buffer for disk command response
DATA LENGTH	= Length of buffer at DATA ADDRESS
CONTROL LENGTH	= 3

The RESULT RECORD contains the following:

RETURN CODE	= Transporter status code
SOURCE HOST #	= Disk server transporter device address
DATA LENGTH	= Length of data received at DATA ADDRESS
USER CONTROL DATA	= Information from disk server (3 bytes)

Length of disk command response including status byte (2 bytes)
Disk command status(1 byte)

The command control block details for STEP 2, the broadcast message command, are as follows:

COMMAND CODE	=	$40
RESULT RECORD ADDRESS	=	Address for transporter RETURN CODE
DESTINATION SOCKET	=	$80
DATA ADDRESS	=	(see below)
DATA LENGTH	=	Length of data to send at DATA ADDRESS
CONTROL LENGTH	=	0
DESTINATION HOST #	=	$FF (broadcast)

The DATA contains the following:

(3 bytes)	Special code of hex 01FE01
(2 bytes)	Send length of disk command (4 bytes max.)
(2 bytes)	Receive length of disk command excluding status byte
(4 bytes max.)	Disk command (i.e., Read boot block)

When data is received in the receive socket $B0, the most significant bit of the first byte of the USER CONTROL DATA must be checked. If this bit is on, the disk has been reset and the operation should be restarted from STEP 1 above. The RESULT RECORD contains a field called SOURCE HOST #. This field is used to determine the disk server's transporter number.

SHORT DISK COMMANDS

To send any disk command to the disk server that is 4 bytes or less in length, the computer must perform the following two transporter commands:

STEP 1

Use the setup receive command to setup receive socket $B0.

STEP 2

Use the send message command to send a message to socket $B0.

The command control block details for STEP 1, the setup receive command, are as follows:

COMMAND CODE	=	$F0
RESULT RECORD ADDRESS	=	(see below)
SOCKET NUMBER	=	$B0
DATA ADDRESS	=	Address of buffer for disk command response
DATA LENGTH	=	Length of buffer at DATA ADDRESS
CONTROL LENGTH	=	3

The RESULT RECORD contains the following:

RETURN CODE	=	Transporter status code
SOURCE HOST #	=	Disk server transporter device address
DATA LENGTH	=	Length of data received at DATA ADDRESS
USER CONTROL DATA	=	Information from disk server (3 bytes)

Length of disk command response including status byte (2 bytes)
Disk command status (1 byte)

The command control block details for STEP 2, the send message command, are as follows:

COMMAND CODE = $40
RESULT RECORD ADDRESS = (see below)
DESTINATION SOCKET = $B0
DATA ADDRESS = Address of disk command
DATA LENGTH = Length of disk command (4 bytes max.)
CONTROL LENGTH = 4
DESTINATION HOST # = Disk server transporter number

The RESULT RECORD contains the following:

RETURN CODE = Transporter status code
RESERVED = (3 bytes unused)
USER CONTROL DATA = Information for disk server (4 bytes)
Send length of disk command (2 bytes)
Receive length of disk command excluding status byte (2 bytes)

When data is received in receive socket $B0, the RESULT RECORD contains a field called SOURCE HOST #. This field should be checked to insure the request is from the disk server. Additionally, the most significant bit of the first byte of the USER CONTROL DATA must be checked. If this bit is on, the disk has been reset and the operation should be restarted from STEP 1 above.

LONG DISK COMMANDS

To send any disk command to the disk server that is 5 bytes or more in length, the computer must perform the following four transporter commands:

STEP 1

Use the setup receive command to setup receive socket $B0 for the "GO" response.

STEP 2

Use the send message command to send the first 4 bytes of the disk command to the disk server's socket address $B0.

STEP 3

Use the setup receive command to setup receive socket $B0 to receive the response from the disk.

STEP 4

Use the send message command to send the remaining bytes of the disk command, starting with the fifth byte, to the disk server's socket address $A0.

The command control block details for STEP 1, the setup receive command, are as follows:

COMMAND CODE = $F0
RESULT RECORD ADDRESS = (see below)
SOCKET NUMBER = $B0
DATA ADDRESS = Address of buffer to receive two byte "GO".
DATA LENGTH = Length of buffer at DATA ADDRESS (2 byte min.)
CONTROL LENGTH = 0

The RESULT RECORD contains the following:

RETURN CODE = Transporter status code
SOURCE HOST # = Disk server transporter device address
DATA LENGTH = Length of data received at DATA ADDRESS (2 bytes)
USER CONTROL DATA = None

The command control block details for STEP 2, the send message command for the first 4 bytes of the disk command, are as follows:

COMMAND CODE = $40
RESULT RECORD ADDRESS = (see below)
DESTINATION SOCKET = $B0
DATA ADDRESS = Address of disk command (first 4 bytes)
DATA LENGTH = 4
CONTROL LENGTH = 4
DESTINATION HOST # = Disk server transporter number

The RESULT RECORD contains the following:

RETURN CODE = Transporter status code
RESERVED = (3 bytes unused)
USER CONTROL DATA = Information for disk server (4 bytes)
Send length of disk command including current 4 bytes (2 bytes)
Receive length of disk command excluding status byte (2 bytes)

When data is received in receive socket $B0, the RESULT RECORD contains a field called SOURCE HOST #. This field should be checked to insure the request is from the disk server. Additionally, the most significant bit of the first byte of the DATA must be checked. If this bit is on, the disk has been reset and the operation should be restarted from STEP 1 above. The first two bytes of the DATA should contain the upper case ASCII characters "GO". After a valid "GO" is received steps three and four should be performed.

The command control block details for STEP 3, the second setup receive command, are as follows:

COMMAND CODE = $F0
RESULT RECORD ADDRESS = (see below)
SOCKET NUMBER = $B0
DATA ADDRESS = Address of buffer for disk command response
DATA LENGTH = Length of buffer at DATA ADDRESS
CONTROL LENGTH = 3

The RESULT RECORD contains the following:

RETURN CODE = Transporter status code
SOURCE HOST # = Disk server transporter device address

```
   DATA LENGTH           = Length of data received at DATA ADDRESS
   USER CONTROL DATA     = Information from disk server (3 bytes)
      Length of disk command response including status byte (2 bytes)
      Disk command status byte (1 byte)
```

The command control block details for STEP 4, the send message command for the remaining bytes of the disk command, are as follows:

```
   COMMAND CODE          = $40
   RESULT RECORD ADDRESS = Address for transporter status code only
   DESTINATION SOCKET    = $A0
   DATA ADDRESS          = Address of disk command (bytes 5-N)
   DATA LENGTH           = Length of remaining bytes of disk command
   CONTROL LENGTH        = 0
   DESTINATION HOST #    = Disk server transporter number
```

When data is received in receive socket $B0, the RESULT RECORD contains a field called SOURCE HOST #. This field should be checked to insure the request is from the disk server. Additionally, the most significant bit of the first byte of the USER CONTROL DATA must be checked. If this bit is on, the disk has been reset and the operation should be restarted from STEP 1 above.

CHAPTER 3

Transporter Card Programming Guide

CORVUS CONCEPT OMNINET PROGRAMMING GUIDE

Chapter One of this OMNINET Programmer's Guide describes the commands that can be used with the transporter. The Corvus Concept communicates with the built-in Omninet transporter by first formatting a command control block and then sending the command control block address to the transporter through the use of two I/O addresses. When the command is completed, a return code is placed in the result record address as specified in the command control block. An interrupt is generated when the operation is completed. For a detailed description of commands, control block formats, and return codes see Chapter One.

TRANSPORTER READY STATUS FLAG

The Omninet transporter ready (RDY) flag is bit 0 of VIA port A which is at address $30F7F. This status bit is read-only.

When set, this bit indicates the transporter is ready to receive the next byte of the three byte command control block address. This bit is cleared when a byte is moved into the Command Address Register (CAR).

CAR - COMMAND ADDRESS REGISTER

The Command Address Register (CAR) is and 8-bit write-only register with a standard address of $30FA1.

To issue a command to the transporter, the three byte address of the command control block is given to the transporter one byte at a time. Every time an address byte is placed into the CAR, the RDY bit in VIA Port A goes low and the next byte must not be sent until the RDY bit returns high again. The three byte address is sent with the most significant byte first.

SOFTWARE NOTES

While the transporter is receiving a packet from the network,it will not process a byte moved into the CAR so the RDY bit in VIA Port A remains low until the transporter can process the next byte. This leads to a situation where a software I/O driver may have to wait to up to several milliseconds before the RDY bit goes high again.

Since the transporter processes one command at a time, the computer should not place any additional data into the CAR after it has issued a command, until the command has completed as indicated by the command return code.

OPERATION COMPLETE RETURN CYCLE

An operation complete return code is generated after the completion of each command issued to the transporter. The return code is placed in the address specified as the result record address in the command control block. Two return codes are generated for a valid setup receive command. The first return code indicates the command was accepted and the socket is setup to receive a message. The second return code occurs when a message is received. The program should initialize the return code byte result record to hex $FF before the command code block is sent to transporter.

INTERRUPTS

Transporter generates an interrupt upon completion of each command issued to the transporter after the return code has been updated. The only exception to this is for the setup receive command. For this command no interrupt is generated for the first status change, receive socket setup. An interrupt is generated when a message is received in the setup socket.

Omninet interrupts are generated on level 3. The interrupt status must be cleared by strobing the interrupt address $30FC1.

A potential problem exists when multiple operations are pending concurrently on Omninet. If a second Omninet operation completes before the interrupt status at interrupt address $30FC1 has been reset, then the second interrupt may be missed. The situation can normally be avoided by checking all outstanding Omninet request return codes before returning from the Omninet interrupt routine. In general, when an Omninet interrupt occurs, the routine must check the return code value of each active transporter command to determine which operations have completed.

BYTE ORDER

All Omninet addresses and lengths must be specified with the most significant byte first and the least significant byte last. Additionally, some addresses and lengths are not on word boundaries.

SAMPLE PROGRAM

The following sample code shows how the command control block address is sent to the transporter. Also shown is the code to perform an Omninet interrupt reset.

```
;
; CLRINT - Clear OmniNet interrupt
;
CLRINT     TST.B        $30FC1.L                ;Reset the OmniNet interrupt
           RTS
;
;
;
; STROBCMD - Strobe in CommandAddress into Transporter
;          Entry : (D1) = CommandAddress
;          Exit  : (EQ) = Transporter was ready and did strobe.
;                  (NE) = Transporter not ready, D7 has error code.
; clobbers registers D0,D3,A6
;
STROBCMD   LEA          $30FA1.L,A6             ;transporter strobe address
           MOVE.L       D1,D0
           SWAP         D0                      ;Do High first
           BSR.S        WAITRDY
           BEQ.S        STBERR                  ;transporter not ready
           MOVE.B       D0,(A6)                 ;strobe in hi byte
;
           MOVE.L       D1,D0                   ;Do Medium byte of address
           LSR.W        #8,D0
           BSR.S        WAITRDY
           BEQ.S        STBERR                  ;transporter not ready
           MOVE.B       D0,(A6)                 ;strobe in hi byte
;
           MOVE.L       D1,D0                   ;Do Low byte
           BSR.S        WAITRDY
           BEQ.S        STBERR                  ;transporter not ready
           MOVE.B       D0,(A6)                 ;strobe in hi byte
;
           CLR.L        D0                      ;Show no error
           RTS
;
; Error Exit Transporter not ready
;
STBERR     MOVEQ        #TRNPNR,D7              ;Error code
           RTS
;
; WAITRDY - Wait for Transporter ready or until timed out
;
WAITRDY    MOVE.W       #LONGTIME,D6
WAITLP     BTST         #RDYBIT, $30F7F.L       ;repeat
           DBNE         D6,WAITLP               ;until count done or set
           RTS                                  ;DBcc innstruction doesn't
           END                                  ; effect the cc's
```

APPLE II TRANSPORTER INSTALLATION AND PROGRAMMING GUIDE

INSTALLATION

The Apple II OMNINET interface card, called a transporter, contains a Dip switch with eight microswitches. Microswitches 1-6 are used to set the unique OMNINET device address. The switch setting for each of the possible 64 device addresses are described in the OMNINET Installation Guide.

Microswitch number 7 is reserved as a network bias offset switch and is disabled on Apple II transporter cards. It is recommended that the disk server be used as the network bias device and hence switch number 7 should be on for the disk server and off for all transporter cards. In Apple networks without a disk server, a network junction can be used to set the network bias for the segment. A network bias is used to reduce the effect of noise on the line when it is idle.

Microswitch number 8 is reserved for network termination and is disabled on the Apple II transporter cards. Switch 8 should be off for all transporter cards and terminators must be physically installed at both ends of the network as described in the OMNINET Installation Guide.

The transporter card may be installed in any Apple II slot except slot 0. When used with Corvus CONSTELLATION software, the transporter card is normally installed in slot 6 of the Apple II.

PROGRAMMING GUIDE

Chapter One of this OMNINET Programmer's Guide describes the commands that can be used with the transporter. The Apple II communicates with the transporter by first formatting a command control block and then sending the command control block address to the transporter through the use of one control register. This control register is referred to as the Status and Command Address Register (SCAR). When the command is completed, a return code is placed in the result record address as specified in the command control block. For a detailed description of commands, control block formats, and return codes see Chapter One.

The Status and Command Address Register (SCAR) is an 8-bit register. The SCAR address is determined by the slot the transporter is installed in as shown in the chart that follows:

#	Hex	Decimal	Decimal
1	$C090	49296	-16240
2	$C0A0	49312	-16224
3	$C0B0	49328	-16208
4	$C0C0	49344	-16192
5	$C0D0	49360	-16176
6	$C0E0	49376	-16160
7	$C0F0	49392	-16144

The sign bit of the SCAR is the Transporter ready bit (RDY). When set, this bit indicates the transporter is ready to receive the next address byte of the three byte command control block address. To issue a command to the transporter, the three byte address of the command control block must be given to the transporter one byte at a time. Every time an address byte is placed into the SCAR, the RDY bit of the SCAR goes low and the next byte cannot be sent until the RDY bit returns high again. The three byte address is sent with the most significant byte first. For the Apple II the first byte is always zero since the Apple II address space only requires two bytes.

SOFTWARE NOTES

While the transporter is receiving a packet from the network, it cannot process a byte moved into the SCAR, so the RDY bit of the SCAR remains low until the transporter can process the next byte. This leads to a situation where a software I/O driver may have to wait up to several milliseconds before the RDY goes high again.

A command return code is generated after the completion of each command issued to the transporter. The return code is placed into the address specified by the result record address field of the command control block. The program should initialize the command return code to a hex value of $FF before the command control block address is sent to the transporter. After the command control block address is sent to the transporter, the program can periodically check the return code for a value of other than hex $FF to determine when the operation has completed. Two return codes are generated for a valid setup receive command. The first return code indicates the command was accepted and the socket is setup to receive a message. The second return code occurs when a message is received.

Since the transporter processes one command at a time, the computer should not place any additional data into the SCAR after it has issued a command, until the command has completed as indicated by the command return code.