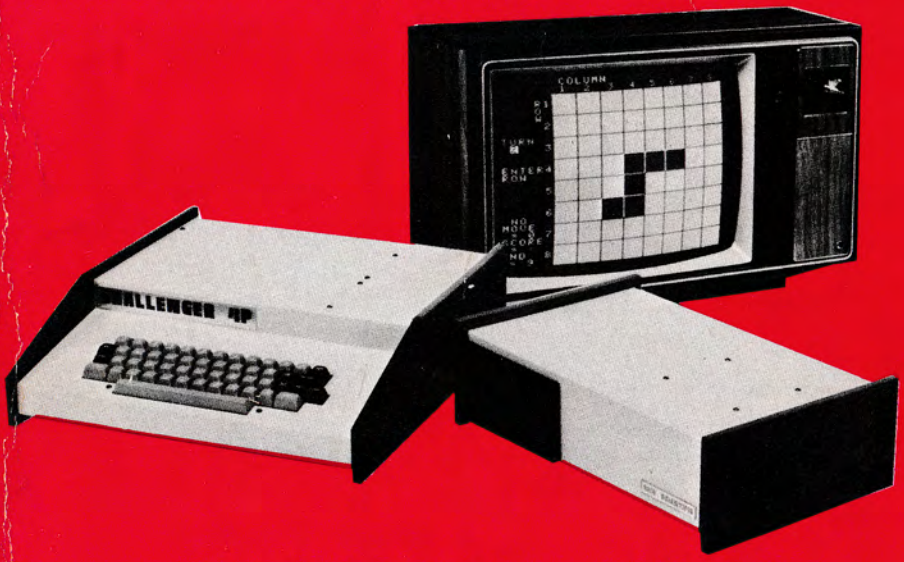
J. CLOTHIER & W. ADAMS

THE FIRST BOOK OF OHIO SCIENTIFIC



VOL. I

$ 7.95

ELCOMP PUBLISHING, Inc.
announces:

The First Book of Ohio Scientific, Vol. II

Introduction to OS-65D Version 2.0 Disk Operating System. Introduction to OS-65U Microcomputer Operating System, OS-65U String Variables, Moving machine code into OS-65U, OS-65U Editor.

The Ohio Scientific Wordprocessor WP-2, I/O Drivers, Memory Test Program, hints and tricks for the disk-user, business applications, Ohio Scientific Mini Floppy, Expansion Accessories, Creating'DATA FILES ONLY' diskettes and much much more.
Order No. 158 (May 1980) $ 7.95

The First Book of Ohio Scientific,Vol. III

Very important information for the Ohio Scientific System Experimenter, hard-to-get schematics, Interface Techniques, System Expansions, Accessories and much much more.
Order No. 159 (June 1980) $ 7.95

J. CLOTHIER & W. ADAMS

THE FIRST BOOK OF OHIO SCIENTIFIC VOL. I

ELCOMP Publishing, Inc. 3873-L Schaefer Avenue
Chino, CA 91710

Acknowledgement:

1. OHIO Scientific Technical Newsletters
   No. 1 - No. 23
   Copyright by
   OHIO SCIENTIFIC
   1333 S. Chillicothe Road
   Aurora, Ohio 44202
   (216) 562-5177

2. OHIO Scientific's Small Systems Journal
   Vol. I, No. 1 to No. 6 and
   Vol.II, No. 1 and No. 2

3. OHIO Scientific Expansion Information
   Conversion of C1P (Cassette) to 52x26 display. Detailed step-by-step instruction for doubling the C1P speed and display size!
   From Steven Chalfin

   Conversion of C1P (Minifloppy) to 52x26 display. Same as above but includes display driver for disk operating system.
   From Dave Wilkie

4. Aardvark Instruction Articles
   1. 600 Baud Baud-Rate-Modification
   2. Joystick Instructions
   3. Graphics with Challenger
   4. How to read a line of Microsoft BASIC
   5. File Handling with OHIO Challenger

   This information and much more, also software,you can order from
   Aardvark Technical Services
   1690 Bolton
   Walled Lake, MI 48088

5. Users Manual for Superboard II and 1P from OHIO Scientific

6. The Challenger Character Graphics Reference Manual by OHIO Scientific

7. COMPUTE. The Journal of Progressive Computing

COMPUTE is published by
Small System Services, Inc.,
900 - 902 Spring Garden Street
Greensboro, NC 27403

8. Expansion Handbook for 6502, edited by
Silver Spur
3873F Schaefer Avenue
Chino, CA 91710

published by ELCOMP Publishing, Inc.

FOREWORD

These volumes of "The First Book of Ohio Scientific" are our initial effort at getting Ohio Scientific information to their many deserving customers. Ohio Scientific is sincerely dedicated (in our opinion) to providing good computer hardware value. The equipment is fairly priced and well designed for ease of expansion. However, Ohio Scientific has lacked the sophistication required for writing and distributing a good set of technical manuals. At ELCOMP, we felt that we could no longer wait for the appearance of good documentation.

We have been astounded at the large number of fellow C1, C2, and C3 users who also have been unable to obtain copies of badly needed listings, schematics, software definitions, and operating instructions. "Lets get the Ohio Scientific bandwagon rolling", we say. And we ask others to help by sending contributionsand suggestions to this first edition. At the earliest possible date ELCOMP will finalize the manuals, incorporate new information, and add additional volumes, as necessary.

John Clothier
W. Adams
March, 1980

TABLE OF CONTENTS

Introduction to Personal Computing

Those who remember the first computer built in the 40's, ENIAC, may also remember that it was built with 18,000 vacuum tubes and devoured 130,000 watts of power. So it's not surprising that many people still think of computers as immovable, awesome and erratic creatures that only scientists could appreciate.
But over the years, computers have become considerably more approachable. Less than a quarter of a century after ENIAC, the technology that made digital watches and handheld calculators possible was applied to computer systems. The microprocessor, a computer on a single chip of silicon, made it possible to develop astonishingly powerful computer systems at astonishingly low cost.

Today, the computer's gigantic proportions have been scaled down to where the personal computer weighs only a few pounds, takes up perhaps 1 1/2 square feet of space, and uses about as much power as a small light bulb.

It's been estimated that by 1985, three out of four American homes will contain personal computers. But no one need wait until then to benefit from what they offer. For the age of the computer is here - now. The state-of-the-art has developed to a point that the personal computer is an affordable useful tool hardly more complicated to use than a calculator, and considerably more entertaining and useful in its diversity. In the next few pages, you'll find out how you can use it for learning, entertainment, small business and personal use, right at home. No, the personal computer isn't exotic. It's a valuable product of technology that young and old can use and profit by.

Small Computers are what Ohio Scientific is all about

Ohio Scientific has been manufacturing microprocessor computer systems since the introduction of the first microprocessor. Microcomputers are the only products Ohio Scientific makes.

Since its inception, the company has utilized the technology to produce computers that are considered highly advanced, by independent experts. Its small business computer, the C3-B for example, was named an outstanding microcomputer application for a small business at the 1978 WESCON show, which is the leading show of the electronic engineering community. A 1977 comparison of the execution speeds of microcomputer systems, conducted by KILOBAUD Magazine, ranked Ohio Scientific computers first, third and fourth in a field of about 30 small computer systems. The competition included computers from IBM, Wang, and Digital Equipment Corp., among others. And the Ohio Scientific units' retail prices were nearly the lowest of all.

In this book, you'll find the Challenger line of computers described in detail. Each has the advantage of being modular, which means that should you begin with a basic model, you can build greater sophistication into it as your requirements increase. In addition, when you purchase a personal computer from Ohio Scientific, you can take satisfaction in knowing that the entire line is backed by nationwide dealer service and trained experts, who can advise you in your choice and provide back-up service should you ever require it.

Q & A: Some of the most commonly asked questions about personal computers:

What is a computer, exactly?

It's really a system for using, storing and updating information. It can solve complex mathematical problems quickly and communicate in words, pictures and numbers, It maintains records, controls equipment and does other tasks that require extensive information storage. It is not an independent thinker. It gives back only the information that has already been fed into it.

What are its basic mechanical parts?

The computer is made of four essential components: the CPU (Central Processing Unit), which is the main part of the computer; an Input Device, which is a keyboard that looks like a typewriter and allows you to enter instructions into the computer; an Output Device that typically consists of a TV-like display terminal called a CRT; and a floppy disk or cassette player that provides information storage.

Extra add-on devices are called peripherals. When you add a peripheral, you "interface" it with your CPU. When you need to know something, you "access" the CPU's memory.

If a person doesn't know how to program the computer, how can he use it?

You needn't know how to program a personal computer in order to use it. Ohio Scientific has already created hundreds of programs on cassette or floppy disk, ready to put you "on line" in minutes. There's even a series of programs to help you learn a basic computer language called BASIC, which is the language

that enables you to communicate with the computer through English-like words and eventually create your own programs, if you so choose.

With such a wide range of programs already available for learning, business use, personal use and entertainment, some people don't feel a need to learn to program. But many, intrigued with the almost limitless possibility of the computer, do go on to create their own programs.

What about cost?

A personal computer ranges in price from several hundred dollars (less than the price of a medium-sized color TV set) to several thousand, depending on your requirements, i.e., how much you want it to do.

THE CHALLENGER PERSONAL COMPUTER AS TEACHER. Until their cost and size decreased, computer were not a practical addition to the classroom or the home. But today, thousands of personal computers are being used in schools and homes across the country.

With society so dependent upon the computer in every walk of life, just knowing how to interact with one is an aid to children. Beyond that, the computer teaches a child at his own pace and level with infinite patience The keyboard teaches manual dexterity and the computer keeps learning interest at a high level, for the simple reason that it's fun to use.

Ohio Scientific's library of programs is one of the largest offered by any computer manufacturer, and a professional staff of programmers at OSI ensures new programs on a continuing basis. At the present time, Chal-

lenger owners can choose from a variety of courses, ranging from foreign language drills in French, German or Spanish, to courses in physics, math (from basic addition to integrals and trig), English spelling and usage, on into such esoteric subjects as nuclear chemistry.

Some learning programs take the form of games. Others are drills. In effect, every transaction with the computer is a form of problem-solving and learning experience enjoyable for children or adults of virtually any age.

THE CHALLENGER PERSONAL COMPUTER AS A PERSONAL AID.

A computer can perform tasks that nobody likes to do on his own. It can balance your checkbook, calculate savings and interest projections, make annuity and loan analyses, aid in preparing tax returns and provide family budgeting and planning for the future.

In the area of home control, the Challenger can be programmed to control a wide range of AC appliances and lights remotely, without wiring, and as an interface with home security systems which monitor fire, intrusion, car theft, water levels and freezer temperature, all without messy wiring.

On a more personal level, the computer will chart your biorhythms, perpare a calorie guide, assist you along a schedule of exercise and muscle toning, catalogue your stamp and coin collections and even help prepare menus.

And for sheer enjoyment, the personal computer offers a delightful selection of games to play, from Star Wars to Black Jack, Tic Tac Toe to Dodgem, Cryptography, Concentration,

Bowling, Breakout, and many more. There are games the whole family can play against the computer or "solitaire" games that pit one person against the computer's expertise. And once again, the computer teaches problem-solving, even while providing fun and games for everyone in the family.

The Challenger Personal Computer as Business Tool

Not surprisingly, over half of the personal computers sold are bought by small business. In view of the computer's ability to save time, drudgery and money, it makes good sense to consider its business capability.

Business applications for personal computers generally fall into five categories: Data Processing (bookkeeping tasks such as Accounts Receivalbe, Accounts Payable, General Ledger and Payroll); Word Processing, the method of entering texts, lists and other manuscripts into the computer so that they can be corrected, edited, or rearranged; Inventory and Order Processing; Financial Consultation; and finally, Personal Services, which include such aids as providing a personal calendar, a computerized phone directory, and even automatic dialing and control.

Ohio Scientific offers a full range of software that complements your choice of a compute and helps it to realize its full potential as a business aid. Moreover thanks to their modular construction, these personal computers can be equipped with peripheral equipment that enlarges their applications boundaries and increases their utility.

Computer Glossary...

BASIC (Beginners All-Purpose Symbolic Instruction Code) A popular computer language ideally suited for use with Ohio Scientific computers. One of the simplest languages to learn, it can be used for a wide variety of applications.

BAUD A measure of the speed with which information can be communicated between two devices. E.g., if the information is in the form of alphabetic characters, then 300 baud usually corresponds to about 30 characters per second.

BIT The smallest amount of information that can be known. (One or zero.) Eight bits equal one byte.

BUS The means used to transfer information from one part of the computer to another.

BYTE A unit of information composed of 8 bits, which is treated by the computer as a single unit. A byte is usually used to represent an alphanumeric character or a number in the range of 0 to 255.

CASSETTE A medium for the electronic storage of data. Similar to magnetic tape. Most personal computers use ordinary audio-cassette tape recorders and tape.

CENTRAL PROCESSING UNIT (CPU) The part of computer hardware responsible for interpreting data and executing instructions.

COMPUTER An electronic device which is programmable and which processes, operates on and outputs information according to its stored program upon receipt of signals through an I/O device.

COMPUTER LANGUAGE A language that is used for programming a computer, e.g., BASIC.

DAC (Digital-to-Analog Converter) A device that changes digital signals into one continuous analog signal. (voltage output)

DATA The information, or set of signals, that is processed by a computer.

DIGITAL Word used to describe information that can be represented by a collection of bits. Modern computers store information in digital form.

DISK A circular piece of rigid material that resembles a record, which has a magnetic coating similar to that found on ordinary recording tape. Digital information can be stored magnetically on a disk.

DISK DRIVE A peripheral which can store information on, and retrieve information from, a disk. A "floppy disk drive" can store and retrieve information from a floppy disk.

FLOPPY DISK A thin, pliable 8" or 5¼" plastic square for storing data. 8" disks store 3, or more, times as much information as 5¼" floppies and access the information much faster.

FOREGROUND/BACKGROUND Operation term used to describe the ability of a computer to function with normal programs at the same time it monitors external devices, e.g. home appliances, security, etc.

HARD COPY Information printed on paper or any durable surface, as opposed to information temporarily presented on the CRT screen.

HARDWARE The physical equipment that makes up the computer system.

INPUT Signals given to a computer for processing.

INTERFACE The connection between two systems. A printer interface, for example, connects the printer to the computer.

I/O (INPUT/OUTPUT) DEVICE Hardware used for communication with a computer. For example, a keyboard, floppy disk drive and a printer are all I/O devices.

JOYSTICK Peripheral, accessory equipment that permits the user to move the figures on the monitor. For example, when you and another person play a joystick computer game, you operate joysticks to perform the functions of the game.

K The initial "K" stands for "kilo," meaning 1,000. In computer language, K means 1,024 bytes of information that can be stored in a computer system. A computer with 16K memory, for example, means that the computer has 16 times 1,024, which is 16,384 bytes of memory.

MEMORY The area in the computer for storage of data and instructions.

MICROCOMPUTER A computer based on a microprocessor.

MICROPROCESSOR The "brains" or CPU of a modern personal computer. All Ohio Scientific personal computers use the 6502 microprocessor, generally recognized as the fastest microprocessor available.

MINI-FLOPPY DISK A small 5¼" floppy disk that stores about ¼ the information of an 8" floppy disk.

MODEM Word derived from MOdulator-DEModulator. A device that allows the computer to communicate over telephone lines and other communications media by changing digital information into audio tones (modulating) and from audio tones into digital information (demodulating).

MONITOR A CRT or television screen. You can purchase an Ohio Scientific monitor to hook up to your computer or else simply use an ordinary TV set and attach it with an RF converter.

OS Operating system.

PERIPHERAL Any device that can send information to and/or receive information from a computer, e.g., printer, modem, etc.
PRINTER A peripheral device which makes hard copy of letters and numerals.
PROGRAM A set of instructions, arranged in a specific sequence, for directing the execution of a specific task, or the solution of a problem, by a computer.
RAM (Random Access Memory) A storage device and main memory of any computer, which can be read from and written into. Information and programs are stored in RAM, and they can be retrieved or changed by a program.
ROM (Read-Only Memory) A memory storage device in which the information is stored once, usually by the manufacturer, and cannot be changed.
SOFTWARE Programs and operating systems used by the computer; may be on cassette or on disk and in ROM.

Challenger 1P: The Perfect Starter Computer

The popular Challenger 1P just may be the best bargain in personal computers available anywhere. This cassette-based computer has a 53-key keyboard, and a black and white video display of 30 rows by 30 columns of alphabetical characters. It has 8K BASIC-in-ROM. This means that your computer comes equipped with approximately 8000 characters in its built in read-only-memory (ROM). Read-only-memory is available immediately when you turn on your computer. The C1P comes with 8K RAM. RAM is random access memory that can be read from and written into. In other words, with a C1P you have available to you approximately 8000 characters of memory or program storage.

The 1P also incorporates features normally associated with far more expensive models: upper and lower case on the keyboard, graphics, extremely fast execution, and effective screen resolution of 256x256 points in the graphics mode for detailed pictures.

The Challenger 1P needs no lengthy preparation for use. You just attach it to your TV (with an RF converter) or plug in an optional TV monitor.
Connect an ordinary cassette player, drop in an Ohio Scientific cassette program and you're on line in minutes. And, when you're ready to expand your system, you can add single or dual floppies for faster program execution and great versatility of function.

These features are all integrated into a compact portable package making the C1P particularly useful in the classroom as well as at home. And it effectively competes with, and often surpasses computers priced several hundred dollars higher.

Challenger 1P MF: Greater Speed, More Versatility

Although they share many of the same basic features, the C1P MF differs from the C1P in two major respects: First, this 8K BASIC-in-ROM, 12K RAM computer is a "mini-floppy", disk-based computer, instead of cassette-based. With "mini'floppies", you get program and data retrieval in seconds, compared to the more conservative cassettes which need several minutes to load and store information.

Second, you can enlarge the Challenger 1P MF 12K memory up to 32K RAM. Once the system is expanded to 20K or more, you can use Ohio Scientific's more powerful OS-65D V3.0 operating system. This system will support sequential, as well as random access data files, directly in BASIC, which allows data storage in advanced applications such as those in small businesses.

The C1P MF can support a printer for hard copy print-outs, a modem, which allows your computer to communicate with other computers over telephone lines, an AC Remote Control System that allows your computer to operate lights and appliances, and a real-time-clock, which gives you the ability to have your computer exercise a time control over computerized functions.

A real breakthrough in terms of both price and performance, the Challenger 1P MF is the first portable "mini-floppy", personal computer offered in the industry for less than a thousand dollars.

Personal Computer Breakthrough

The Ohio Scientific Challenger 1 is a Dramatic Demonstration of Price and Performance in a Single Package

Challenger 1 is the microcomputer that scored a dramatic breakthrough in price and performance at a terrific system price.

PRICE/PERFORMANCE LEADER

The development of the Ohio Scientific Challenger 1P marked a price revolution for microcomputers. Consider the following features:

(1) Microsoft 8K Basic-in-ROM
(2) 4K RAM-Expandable to 8K on board
(3) Full 53 Key keyboard with upper and lower case
(4) Elaborate graphic display capability
(5) Uses a standard cassette for input
(6) Uses a standard television with an RF modulator for video output.

All of these features makes the Challenger 1P best buy for a beginner or hobbyist on a limited budget wishing to get involved with microcomputer, such as students + their educators, and various other professionals.

ALREADY LOOKING AHEAD??

Of course, we are not beginners forever (hopefully!!). What happens to the C1P then? Is it obsolete? That answer is a definite no!!

Consider the specification facts on the expandability of the Challenger 1P:

(1) A fully expanded Challenger 1P can support
   (a) Basic-in-ROM
   (b) 32K of RAM memory
   (c) Dual Mini-floppies
   (d) cassette

(e) printer
(f) modem
(g) full BUS expansion capability via the OSI 48 line BUS through which over 40 accessories can be added (A/D, D/A, voice output, I/O re memory, etc.)

As you can see, the Challenger 1P leaves more than adequate room for expansion. Thus, the system is not one you will have to 'get rid' of as you progress to more sophisticated computer functions.

CHALLENGER 1P with Disk Capabilities
If you are familiar enough with computers to realize the importance of disk storage, then the Challenger 1P Mini Floppy is a prime consideration.

It is the first microcomputer system to offer disk storage capability under $1000, another breakthrough. The C1P MF offers the advantage of using either Basic-in-ROM or Pico-Dos, a compact DOS. The unit comes standard with 12K of RAM.

Additional memory can be added anytime by simply plugging in additional RAM into the socketed memory board. Upon adding 8K additional RAM to get to 20K, you may run OS65DV3.1, a full-fledged DOS allowing named program and data files as well as random and sequential access capabilities to these files.

Superboard II: A Computer for the Budget-Minded

Superboard II is the ideal choice for the beginner who wants a full-fledged personal computer for the lowest possible price. It in-

cludes a complete computer system on a board, with full keyboard, video display, audio cassette interface, 8K BASIC-in-ROM and 4K RAM. All it requires for operation is 5V at 3 amp power and your own (optional) enclosure. The system can be easily expanded to include floppy disk drive, which means when you're ready to go on to more complex computer functions, you needn't discard the existing equipment - you simply add to it to get the system that will do what you want it to do.

Challenger 4P: Color, Sound, Exceptional Display

Like the C1P, the Challenger 4P is cassette-based, with 8K BASIC-in-ROM and 8K RAM. But, there the resemblance ends. For, the C4P features a highly sophisticated video display with 16 colors in both alphabetics and graphics 32 rows by 64 columns of upper and lowe case, graphics and gaming elements, plus an effective screen resolution of 256 x 512 points. The result: over three times the display capability of the C1P as well as visual quality and readability equal to that of the finest color television set.

For voice and music generation and output, th C4P has a 200-20KHz programmable tone generator and a DAC (companding digital-to-analog converter) as well as, 2-10 keypad interfaces printer interface (not wired to connector) an AC Remote Control interface. It utilizes a modular four-slot BUS architecture generally associated with more expensive computers, which allows easy expansion up to 32K RAM and two mini-floppy drives.

The C4P has a fully RF-shielded aluminium cas with 2-step baked on enamel finish. It is trimmed with solid, oiled walnut and die-cast, chromed dress panels, Compare this

functional and stylized construction to the standard plastic cases on other personal computers.

Challenger 4P MF: The ultimate portable Personal Computer

This mini-floppy-based, premium portable is ultra-fast. In fact it is two-to-three times as fast as many competitive models. And remember, the faster your computer can execute instructions, the more elaborate the I/O can be. The C4P MF has more I/O built-in and ready to run than any other personal computer in its class. That is, if you can find one in its class. Regardless of price you'll find none that even come close.

As if the C4P MF's speed isn't already super-fast, with the addition of Ohio Scientific's GT option, utilizing a 6502C microprocessor in conjunction with ultra-fast static memories, it nearly doubles its speed again. The GT option allows your computer to handle 1.2 million instructions per second, average, as well as offering memory to accumulator ADD time of 600NS and JUMP extended of 900NS.

The C4P MF has all the great features of the C4P plus a real-time clock and count-down timer, modem interface, printer interface, 16 parallel line for additional control interfaces, an accessory BUS for an external 48 line I/O board, PROM blaster, analog data module, or education board, and a Home Security interface for fire and intrusion detection and monitoring.

The C4P MF comes with 24K RAM and a single mini-floppy, and can be directly expanded to 48K RAM and two mini-floppies.

Another exciting feature of the C4P MF is its

Foreground/Background operation capability. This means that your computer can be simultaneously monitoring home security and handling appliance functions at the very same time that it is being used for regular basic programming.

Ohio Scientific offers a comprehensive library of both applications software and system software for the C4P MF. Diskettes include games, personal, business, educational and home control application programs as well as a real-time operating system, word processor and a Data Base Management system.

The C4P MF is truly a remarkable, premium computer - years ahead of the market. There probably won't be anything like it for a very long time.

Challenger 8P: Ohio Scientific's Mainframe Class Personal Computer

The Challenger 8P is an 8-slot, Mainframe Class Computer with 5 open slots. This means that it allows over three times the expansio capabilities of the Challenger 4P for advanced home, experimental and small business applications. The cassette-based Challenger 8P comes with 8K BASIC-in-ROM and 8K static RAM. The 8P can be expanded to 22K RAM and you have the option of adding a dual, 8-inch floppy disk drive.

Not only does the Challenger 8P have all the exceptional features of the Challenger 4P, b in addition, its modular construction allows you to easily expand your Challenger 8P to a Challenger 8P DF system which represents the highest state-of-the-art in personal compute systems.

Challenger 8P DF: The Top of Line in Personal Computers

Not only is the Challenger 8P DF the top of Ohio's line, but it is the top of any line of personal computers on the market today. Incorporating the most advanced technology now available, the Challenger 8P DF offers full capabilities as a personal computer, a small business computer, a home monitoring security system and an advanced process controller.

The 8P DF operates 2- to 3-times faster than other personal computers (except the C4P MF). And with the addition of Ohio Scientific's QI option, your Challenger 8P DF can nearly double its speed again.

The 8P DF comes with 32K RAM (expandable to 48K). It features dual, 8" floppy disk drives, which means you can store about 8 times the information of a single mini-floppy and access it many times faster. All the exceptional features of the Challenger 8P are available on the 8P DF. They include: color graphics in 16 colors, effective screen resolution of 256 x 512 points, programmable tone generator, D/A converter, keypad interfaces, joystick interfaces, AC Remote Control interface. Plus: a real-time clock, modem interface, 16 parallel lines, excessory BUS, Foreground/Background operating system and a truly fantastic home control and home security system.

Ohio Scientific's Challenger 8P DF Home Security system allows your computer to monitor fire, intrusion, car theft, water levels and freezer temperature, all without messy wiring. When equipped with an optional Universal Telephone Interface system (UTI) the Challenger 8P DF has the capability to dial any telephone number (either rotary dial or touch tone). Itcan respond to touch tone or

modem signals and can route voice to tape recorders. When equipped with Votrax ® module or used in conjunction with an Ohio Scientific CA-14A voice I/O your Challenger 8P DF can answer by touch tone, modem, stored message or Votrax ® voice output. Equipped with UTI voice output AC Remote, and the home security system option, your Challenger 8P DF is truly "the home computer of the future" with an almost human-like capability to communicate via phone lines and operate and monitor typical functions in your home.

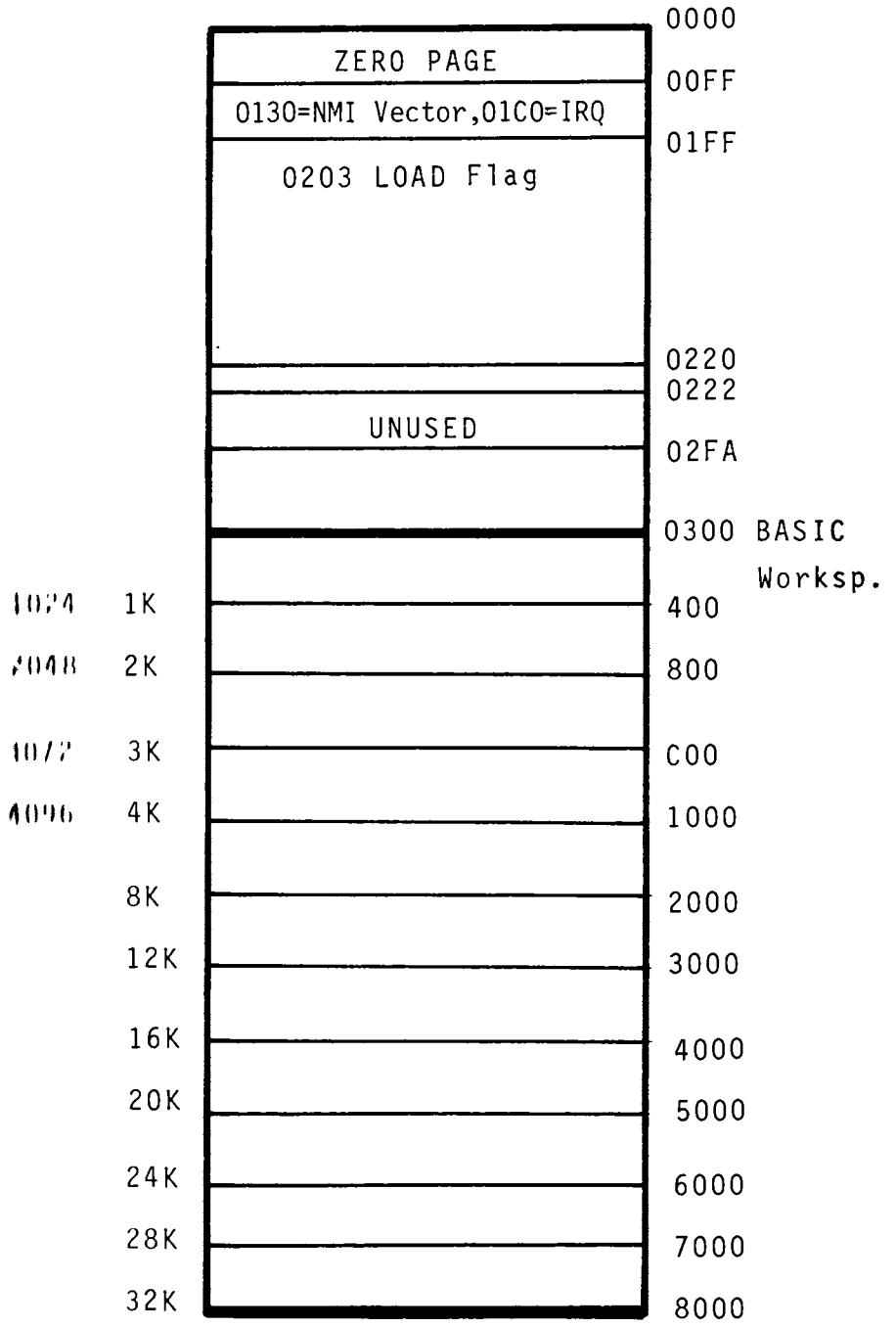
Also note that the Challenger 8P DF has a universal accessory BUS connector accessible at the back of the computer or you can plug in additional 48 lines of parallel I/O and/or a complete analog signal I/O board with A/D, D/A and multiplexers.

Furthermore, the Challenger 8P DF with its full size 8" floppy disk is compatible with Ohio Scientific's advanced small business operating system, OS-65U and two types of information management systems, OS-MDMS and OS-DMS. Since it comes standard with a high speed printer interface, a modem interface, 53 key ASCII keyboard, and a 2048 character display with upper and lower case, it is ideally suited for business use and word processing applications.

Hundreds of personal applications, games, educational and business software packages are currently available for use with the Challenger 8P DF.

The Challenger 8P DF is far and away the finest personal computer available today.

C1P Challenger Memory Map



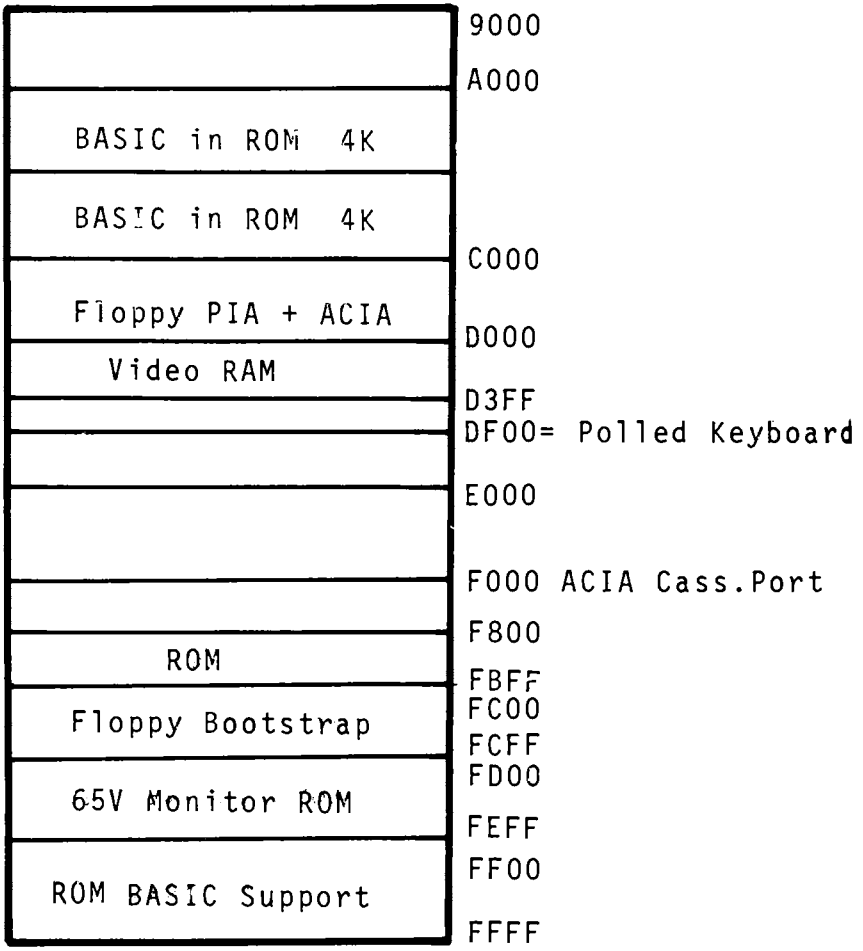

9000
A000
BASIC in ROM 4K
BASIC in ROM 4K
C000
Floppy PIA + ACIA
D000
Video RAM
D3FF
DF00= Polled Keyboard
E000
F000 ACIA Cass.Port
F800
ROM
FBFF
FC00
Floppy Bootstrap
FCFF
FD00
65V Monitor ROM
FEFF
FF00
ROM BASIC Support
FFFF

Table I

A Partial List of OSI BASIC in ROM Scratch Pad Memory

IND(01)	Initially, address of cold start ($BD11). Replaced by warm start ($A274).
IND(06)	USR INVAR address.
IND(08)	USR OUTVAR address.
IND(0B)	USR program address.
M(0D) 13	Number of NULL'S selected (for slow carriage return)
M(0E) 14	Terminal character count
M(0F) 15	BASIC terminal width
M(11-12)	Arguments of statement such as PEEK, POKE, GOTO, GOSUB, line numbers, etc.
M(13-5A)	Input buffer.
IND(71)	Scratch pad address for garbage collection, line insertion, etc.
IND(79)	Address of beginning of BASIC code. ($0301)
IND(7B)	Address of beginning of Variable Table.
IND(7D)	Address of first array entry in Variable Table. If no arrays, end of Variable Table.
IND(7F)	Address of end of Variable Table
IND(81)	Lowest string address.
IND(83)	Scratch pad string address.
IND(85)	Address, plus one, of highest allocated memory.
M(87-88)	Present BASIC line number.
M(89-8A)	Line number at BREAK.
IND(8B)	Pointer to BASIC code for CONT.
M(8D-8E)	Line number for present DATA statement.
IND(8F)	Address of next DATA statement
IND(91)	Address of next value after comma in present DATA statement.
M(93-94)	ASCII code for present variable

M(BC-D3) Subroutine: Points through code one byte at a time, RTS with code value in A and carry clear if ASC(0-9); otherwise, carry set. Return A = 0 if end of line. Ignores spaces.

OMD)C3) Code location pointer for above sub routine.

M(AF-B0) USR input variable storage.

M(FB) MONITOR keyboard control flag. (= 0 for keyboard).

M(100-107)Storage of conversion of floating point number to ASCII.

M(1FF) Top of BASIC stack.

M(200-20E)Temporary storage for CR simulator subroutine ($BF2D).

M(212) CTRL C flag.
(=$01 if CRTL C off).

M(213-216)Temporary storage, keyboard pollin program (#FD00).

TABLE 2

BASIC Routines Needed for BASIC Renumbering

[illegible]	Print an error message from the message table. Enter with X containing the location of the message relative to $A164. Message terminator is ASCII having bit 7 on.
[illegible]	BASIC line insertion routine. Enter with line assembled in the line buffer $0013-$005A with 00 as line terminator. Also, character count must be in $005D and the line number(hex) at $0011/12.
[illegible]	Evaluate an expression whose beginning address is in $00C3/C4. Use this subroutine to convert from ASCII to binary, with the result appearing in the floating accumulator: $00AC/AD/AE/AF.
[illegible]	Convert fixed number in $00AD/AE to floating number. Enter with the result appearing in the floating accumulator: $00AC/AD/AE/AF.
[illegible]	Convert binary value, such as line number, in floating accumulator to two-byte fixed number and place in $0011/12.
[illegible]	Convert floating number at $00AC/AD/AE/AF to ASCII and place in string starting at $0101, preceded by a space or minus sign at $0100 and terminated by 00.
[illegible]	BASIC warm start. Prints "OK".
[illegible]	Prints message. Enter with ADH in Y, ADL in A. Message is ASCII string ending with 00.
[illegible]	Print the decimal integer whose hex value is in registers A and X, for example, a line number.

Table of Locations (I/O Parameters and BASIC Pointers)

This table lists memory locations which contain I/O parameters and BASIC pointers.

LOCATION	LABEL	FUNCTION
FFE0	HOME	Defines home position of cursor - normally set to 64(hex) for 440 video or 40(hex) for 540 video screen.
FFE1	LEN	Defines line length (in hexadecimal) - set to one less than the desired line length.
FFE2	SIZE	Defines size of CRT screen - set to 00 for 440 screen, set to 01 for 540 screen.
FFE3	SOURCL	(LO) pointer to workspace lower boundary
FFE4	SOURCL	(HI) pointer to workspace lower boundary
FFE5	SOURCL	(LO) pointer to workspace upper boundary
FFE6	SOURCH	(HI) pointer to workspace upper boundary
FFE7	TEMPWL	(LO) pointer to temporary workspace lower boundary
FFE8	TEMPWL	(HI) pointer to temporary workspace lower boundary
FFE9	TEMPWH	(LO) pointer to temporary workspace upper boundary
FFEA	TEMPWH	(HI) pointer to temporary workspace upper boundary
FFFA	NMI	(LO) non-maskable interrupt vector
FFFB	NMI	(HI) non-maskable interrupt vector
FFFC	RST	(LO) reset vector
FFFD	RST	(HI) reset vector

In addition, BASIC branches out to the following locations for I/O:

BASIC STATEMENT	LOCATION	CONTENTS (HEX)	EXPLANATION
INPUT	FFEB	4C	JUMP to location (WXYZ) for start of "IN-
	FFEC	(YZ)	PUT" routine (1)
	FFED	(WX)	
OUTPUT	FFEE	4C	Jump to location (WXYZ) for start of "OUT-
	FFEF	(YZ)	PUT" subroutine (1)
	FFF0	(WX)	
CONTROL-C	FFF1	4C	Jump to location (WXYZ) for start of "CON-
	FFF2	(YZ)	TROL-C" subroutine (2)
	FFF3	(WX)	
LOAD	FFF4	4C	Jump to location (WXYZ) for start of
	FFF5	(YZ)	"LOAD" routine. (1)
	FFF6	(WX)	
SAVE	FFF7	4C	Jump to location (WXYZ) for start of
	FFF8	(YZ)	"SAVE" routine. (1)
	FFF9	(WX)	

(1) Las+ executed instruction in these subroutines must be "Return From

Subroutine" (RTS).
(2) The return point for this statement depends on what action is required of BASIC. If the external routine does not detect control-C at the input terminal, or if it is desired to bypass the control-C routine in BASIC, execute an "RTS" instruction. If the external routine detects an input character but does not test to determine if it is control-C, do a jump absolute (JMP) to A636, leaving the character in the accumulator. Jump absolute (JMP) to A633 will cause BASIC to input a character and test it entirely without outside support.

Two other important locations are:

0206 - Controls baud rate of CRT simulator 0 sets fast baud rate, 1 sets slow baud rate. This location may be changed under software control via POKE command. It is also automatically changed during LOAD/SAVE operations.

BD11 - BASIC cold start location.

2. The following source listing of the I/O driver package contains the following useful subroutines:

STARTING ADDRESS	DESCRIPTION
BEE4	UART INPUT routine
BEF3	UART OUTPUT routine
BEFE	UART initialization routine (1)
BF07	ACIA INPUT routine
BF22	ACIA initialization routine (2)
BF15	ACIA OUTPUT routine
BF2D	CRT simulator

(1) UART is set for 8 bits, 2 bits, no parity bit.
(2) ACIA is set for 8 bits plus 2 stop bits. Clock is set for 16 mode.

Useful Subroutine Entry Points

A274 warm start for BASIC
BD11 cold start for BASIC
BF2D CRT simulator - prints char in A register to screen offset by cursor (200 HEX)
FD00 input char from keyboard, result in
FCB1 output 1 byte from A to cassette
FE00 entry to monitor, clears screen, res ACIA
FE0C entry to monitor, bypasses stack ini tialization.
FE43 entry to address mode of monitor
FE80 input ASCII char from cassette resul in A, 7 bit cleared
FE93 convert ASCII hex to binary, result A, -80 if bad
FF69 BASIC output to cassette routine, ou one char to cassette, displays on sc outputs 10 nuls of carriage return c racter

HANDY LOCATIONS

PAGE 0 USAGE

0000 JMP to warm start in BASIC (4C/74/A2
00FB cassette/keyboard flag for monitor
00FC data temporary hold for monitor
00FE-00FF address temporary hold for monit

PAGE 1

0100-0141 stack
0130 NMI vector-NMI interrupt causes jump to this point
01C0 IRQ vector

PAGE 2

0200 cursor position
0203 load flag

```
[illegible]    save flag
[illegible]    CRT simulator baud rate - varies
               from 0 fast to FF slow
[illegible]    Control-C flag
[illegible]    input vector FFBA
[illegible]    output vector FF69
[illegible]    Control C check vector FF9B
[illegible]    load vector FF8B
[illegible]    save vector FF96
[illegible] - 02FA  unused

[illegible]  and up to end of RAM is BASIC workspace

[illegible] - BFFF  BASIC in ROM
[illegible] - D3FF  Video refresh memory
[illegible]         polled keyboard
[illegible] - E001  cassette port 6850
[illegible]   FFFF  Monitor EPROM
[illegible]         Floppy bootstrap
[illegible]         Keyboard input routine
[illegible]         Monitor
[illegible]         BASIC I/O suport
```

SEMI-FAST SCREEN CLEAR (without the USR FUNCTION)

[illegible] to be bothered with the USR screen clear I can't remember it off hand and hate to look it up. Besides, it takes too much memory. This one is fast, clears the whole screen in 2.16 seconds, and easy to use.

```
[illegible] FORX 1TO29:?:NEXT
```

```
    (C2)                          (C1)
[illegible] FORX 55168TO55295:   110 FORX54147TO54275:
    POKEX,32: NEXT                   (54307 on some mo-
                                     nitors) POKEX, 32:
                                     NEXT
```

MORE USEFUL SUBROUTINE ENTRY POINTS

```
[illegible]    Reset entry point
[illegible]    Load flag routine
```

FF96	Save flag routine
FF9B	Control-C routine
FFBA	BASIC input routine

PRINTER POKES

There are two handy locations you should know if you are trying to use a fast cassette interface or a printer with a slow carriage return. In either case it is sometimes handy to put more nulls than the ten that BASIC allows. The null number is stored in 13. By POKEing the number of nulls you want into 13 you can get up to 255 nulls.

Location 518 is the baud rate simulator. It's end effect is to put an interval between printed characters. It can help if your printer has handshake problems or if you just want to slow down the print for emphasis. POKEing a number into 518 gives you the interval.

A PARTICULARLY HANDY SUBROUTINE

BF2D is a real time saver if you are writing machine code. It is a subroutine that prints a character from the accumulator to the screen offset by the value stored in 200 (hex). It also increments the cursor and can process a carriage return so a lot of your work is done for you.

PRINT AT STATEMENT

OSI has a great BASIC but the lack of a PRINT AT command makes it difficult to print scores and names and similar items where you want them on the screen. You usually end up with a long series of POKE statements and you have to decide the score up into individual digits to do even that. There is a simple solution. Add the subroutine to your program -

```
[illegible]ORY=1TOLEN(D$):POKED+Y,ASC(MID$(D$,Y,1))
    NEXT:RETURN
```

To POKE up any name, word, or even sentence on the screen simply set the name equal to D$ and make D=equal the starting address on the screen, i.e.

```
1000D$ "WINNER IS":D=54040:GOSUB5000
```

Scores should be done just a little differently. You start at the second digit because the BASIC thinks the sign is the first digit in the string and can set you over one space from where you planned. You may also want to blank the digit after the string to allow for the possibility that the score may decrease (say from three to two digits). To use it you set the score equal to D$ and the final product looks like this -

```
1000D$=STR$(SCORE):D=54040:GOSUB5000
[illegible]ORY=2TOLEN(D$):POKED+Y,ASC(MID$(D$,
    Y,1)):NEXT
[illegible]POKEY,32:RETURN
```

SOME POKES YOU SHOULD KNOW

If old in reading you may want to set the line length down to 32 on a C2 or to 23 on a C1. Unfortunately, if you set them down when you start up the system you will be unable to load tapes. Fortunately, the line length is stored in location 15. You can reset line length by executing 100POKE15,32 (or any other number down to as little as one) and then reset with 200POKE15,72 to record the program.

If you find it annoying to reserve space for your programs when you fire up the system (I always forget to do it when I am using the rapid screen clear) you can set the memory space by POKEing the high order digit (in HEX) into location 134 and the low order digit

into 133. For instance, the line 100POKE 134,14 will reserve space for the screen clear without resetting the system.

You can even make self starting BASIC pro grams if you are willing to do a few addi tional moments work when you make the tap The flag for LOAD is in location 515. A 1 POKEd into that location turns off the lo mode. Therefore, to make a self startint as soon as the program finishes reading o the tape and while the system is still in SAVE mode, type in POKE 515,1:RUN.
That command will record on the tape and the program automatically when it finishe loading.
SAVE can be turned off in a similar manne POKEing a 0 into location 517.

EASY KEY DETECTION

If you are doing a one player game, you c tect the control keys without either POKE the keyboard or turning off the CONTROL C The values for the shifts, rept, control, esc keys are recorded continuously in loc tion 57100, i. e. if you push the right s a 3 always appears in 57100. To see how i works try this program

```
10PRINTPEEK(57100):GOTO10
```

Then push the control keys one at a time. is simple, fast, and allows you to keep t CONTROL C function to break the program.

[illegible]

[illegible] generation. [illegible] s the new generation up so that you can see the change.

```
100 DIMA(705):FORX=1TO8:READM(X):NEXT:C=53504
110 DATA63,64,65,1,-1,-63,-64,-65
200 REMFORC1 & SUPERBOARD DATA 31,32,33,1,-1,-31,-32,-33
310 INPUTA$:IFA$<>"DONE"THEN310
400 FORX=1TO704:IFPEEK(X+C)=42THENGOSUB2000
410 NEXT:FORX=1TO704:IFA(X)=3THENPOKEC+X,42
420 IFA(X)<2ORA(X)>3THENPOKE(X+C),32
430 A(X)=0:NEXT:GOTO400
2000 FORY=1TO4:IFX<639THENA(X+M(Y))=A(X+M(Y))+1
2010 NEXT:FORY=5TO8:IFX>65THENA(X+M(Y))=A(X+M(Y))+1
2050 NEXT:RETURN
OK
```

This program could be changed to fit the 600 board (C1 and Superboard) by changing the value 1408 to 1000 everywhere it appears, changing 639 in line 2000 to 935, and POKEing G into 54157 and 54158. However, the 4K version runs well on the C1 so that I doubt that it would be worth the effort to type in the longer version.

A LITTLE MORE LIFE?

If you have 8K and want to fill the entire screen on a C2, you can do so and add a few frill in the process.
You will need line 110, 420, 430, 2000, 2010, and 2050 from the 4K program.

```
 100 DIMA(1408):FORX=1TO8:READM(X):NEXT:C=53504:X=1
 300 INPUTA$(X):IFA$(X)<>"D"THENX=X+1:GOTO300
 310 FORY=1TO15:PRINT:NEXT:FORY=1TOX-1:PRINTA$(Y):NEXT
 320 FORY=1TO15-(X/2):PRINT:NEXT
 340 FORX=1TO192:POKEX+C+1407,161:POKEC-X,161:NEXT
 350 FORX=65TO1407:IFPEEK(C+X)=42THENL=X:GOTO370
 360 NEXT:IFL<THENL=1
 370 FORX=1442TO1STEP-1:IFPEEK(C+X)=42THENU=X:GOTO390
 380 NEXT:IFU>1407THENU=1407
 390 X=INT(G/10):POKE55005,X+48:POKE55006,G-(10*X)+48:G=G+1
 400 FORX=1TOU:IFPEEK(C+X)=42THENGOSUB2000
 410 NEXT:FORX=LTOU:IFA(X)=3THENPOKEX+C,42
2020 IFX>U-65THENU=X+65:IFU>1408THENU=1408
2030 IFX<L+65THENL=X-65:IFL<1THENL=1
OK
```

Challenger Superboard Introduction

How to switch it on and try it out

After you have hooked up your C1P to a +5Volt power supply and a modified television set (or monitor) you are ready to start your own programming. You must also install the cassette recorder for saving and loading of your programs.(See manual).

When the powersupply is switched on, the monitor screen fills up with random characters. Now it is ready for you to hit the BREAK key. On the bottom line of the screen you will see the letters D/C/W/M.
If you have only a cassette system, you now have the choice between C = BASIC coldstart
M = Enter System Monitor

The letter W is for warmstart in BASIC and D = used for loading the disk operating system from disk drive A.

Let us assume that we want to start programming in machine language. The superboard monitor (resident in ROM) allows you to do the following:

- Look at and input RAM Address (4 digits) in hex XXXX
- Look at and input RAM DATA (2 digits) in hex XX
- Change from address mode to data mode (/)
- Change from Data to Address mode (.)
- Start a machine language program (G)
- Load machine language program from tape (L)

As an example let us type in the following machine language display program:

Mnemonic Form			Machine Form		
Address	Inst.	Operand	Addr.	Opcode	Oper
0222	JSR	FD00	0222	20	00FD
0225	JSR	BF2D	0225	20	2DBF
0228	JMP	0222	0228	C4	2202

We will put the little program in unused RAM space in memory page 2 (0222 - 02FA). The starting address is 0222 Hex. We type M for Monitor which leads us immediately into the address mode or we can type a period (. when we are still in the DATA Mode.

When the address mode is entered, you will type in the address followed by the data to be loaded:

Address	Data	
0222	20	RETURN
0223	00	RETURN
0224	FD	RETURN
0225	20	RETURN
0226	2D	RETURN
0227	BF	RETURN
0228	C4	RETURN
0229	22	RETURN
022A	02	RETURN

You will recognize that the bytes in the ma chine version of our programs are interchan For further explanation refer to your 6502 programing manual or 6502 instruction refer card.

Our program uses two useful subroutines fro the C1P operating system.

FD00 = Input character from keyboar put result in accumulator (A

BF2D = CRT simulator. Prints char in accumulator to CRT screen, offset by cursor (200 hex)

The internal subroutines should be used as often as possible. Therefore the machine language programmer should first prepare a list of all useful memory locations and subroutines. Why should you write an ASCII output subroutine by yourself, when you can call an even better one from the system?

To start our program we have to type in the starting address 0222 Hex. Therefore we go into the address mode and type in 0222 followed by a point (.) Now we stay in the address mode and we can start the program with G.

Our program allows us to type in letters and display them at the screen. To stop the program you have to hit the BREAK key. Restart with M and 0222 G.

You now can modify this program to escape the loop by typing A and starting the program by hitting the S-key.

For the modification you can use the CMP command (compare with accumulator).

After writing and testing a program you want to save on cassette, you may suddenly realize that there is no machine language write routine in MOM. There is a read routine, which you can start in the monitor. You need only enter the monitor with an M keystroke and follow it with an L keystroke to read tape in OSI's standard format. There are,of course, both read and write tape routines for BASIC in MOM.

The Kansas City tape standard used by OSI

is very reliable, but 300 Baud is not a very fast rate.
Another slowing factor is that the OSI tape format puts three bytes on tape to record one byte of data.
This brings us to an effective rate of 100 Baud, when recording machine language programs.

Now something in general about the audio cassette portition of the Ohio CIP system. The Kansas City interface is really a modem (FSK modulator/demodulator) and this may be used to couple the computer:

1) To a cassette recorder
2) To a printer
3) To a phone line
4) To a 2-way radio

The cassette interface consists of a 6850 ACIA which is programmed to write and read the Kansas City format. Refer to the 6850 ACIA data sheet for a background discription of ACIA programming parameters.

The OSI tape format mimics the keyboard as inputs to the monitor program. That is, each byte is entered as two hex digits in ASCII-code.

As described before, the monitor has two modes. The (.) period symbol puts it into a "read address" mode. Then the next two bytes are placed in memory at $FE and treated as an address. The (/) = slash symbol puts the monitor into the "accept text" mode. Then each byte entered will be stored in temporary memory. When RETURN is received, the byte is moved to memory at the address stored in $FE. Then this address is incremented by one so that the load process ratches along in memory space. After all text is loaded,

" symbol puts the monitor back into the address mode. Inputting a new address with a CR keystroke followed, starts the program at this address.

The following program allows you to write machine language tapes in OSI format.

```
N=221
M=1*N+15
Q=64512:R=Q+1
REM ACIA ADDRESS Q=61440 FOR 600 SERIES.
INPUT "START TAPE AND WAIT FOR LEADER, TH
INPUT G";A$
DATA 46,48,50,50,50,47:REM.0222/
DATA 46,70,69,48,48,71:REM.FE00G
FORI=1TO6:READC:WAITQ,2:POKER,C:PRINTCHR
$(C);:NEXT
S=546:E=S+N
REM FORI=$0222 TO $02FF
FOR I=S TO E
C=PEEK(I):H=C AND 240:L=C AND 15
H=H/16+48:IF H>57 THEN H=H+7
L=L+48:IF L>57 THEN L=L+7
WAIT Q,2:POKE R,H
WAIT Q,2:POKE R,L
WAITQ,2:POKE R,13
PRINT CHR$(H);CHR$(L);" ";
NEXT I
FOR I=1 TO6:READ C:WAIT Q,2:POKER,C:PRIN
T(C);:NEXT
REM FORMAT FOR TAPES IS:
REM.HLHL/HLRHLR...HLR.HLHLG
REM WHERE THE HLHL AT THE START IS THE S
TART ADDRESS,
REM HI BYTE FIRST, THE HLHL AT THE END I
S EXECUTE
REM ADDRESS AND THE HLR'S IN THE MIDDLE
ARE TEXT
REM BYTES. THE R BEING A CARRIAGE RETURN
REM THE ./G ARE THE SAME AS THE COMMANDS
TO THE MONITOR
```

```
 200 REM H AND THE L ARE ASCII CODE FOR THE
EX DIGITS
 205 REM 0THROUGH F.
 210 REM FOR READ TAPE PROGRAMS USE
 215 REM WAIT Q,1:X=PEEK(R)
OK
```

The above program saves machine language codes located in memory between address 0222 hex and 02FF hex on tape.

To change it for other memory locations you have to change line No. 5, line No. 100 and line No. 116.

Example: Saving memory locations from 06FF hex to 07FF hex. In this case you have to change the line numbers as follows:

```
5      N=256
100    DATA 46,48,54,70,70,47:REM.06FF/
116    S=1791:E=S+N
```

Note:

Before loading the program, you have to protect your machine language area from BASIC. In the above example type in

```
      1790
```

for memory size.

```
                ; MACHINE CODE TAPE DUMP (CH-1P, C2-4P, C2-8P ROM BASIC)
                ;
                ;
                ;    EQUATES :
                ;
                START:=$00DD          ; START OF TRANSFER ADDRESS
                ENDADR=$00DF          ; END OF TRANSFER ADDRESS (NON-INCLUSIVE)
                GOADDR=$00E1          ; GO ADDR AT LOAD TIME ($FE03 TO MONITOR)
                WARMST=$FFFC          ; JMP INDIRECT HERE TO ENTER MONITOR
                OUTFLG=$0205          ; (◇0 ) SAYS OUT TO TAPE & SCREEN
                OUTPUT=$FFEE          ; OUTPUT ROUTINE (SCREEN & TAPE)
                RETURN=$0D            ; CARRIAGE RETURN CHR CODE
                ;
                ;
                ;
                     *=$0230
                ;
                ;
                ;
                ENTRY: CLD            ; RST TO BIN MATH
                       LDA #$FF
                       STA OUTFLG
                       LDA START:+1   ; OUTPUT START ADDR
                       LDY START:
                       JSR ADROUT
                       LDA #'/        ; SELECT DATA ENTRY MODE
                       JSR OUTPUT
                ;
                       LDY #$00       ; RST FOR TRANSFER
                DATAOU LDA (START:),Y ; GET BY INDIRECT
                       JSR DOBYTE     ; BYTE OUT TO TAPE
                       LDA #RETURN    ; CR TO MOV TO NXT LOCATION IN MEM
                       JSR OUTPUT
                       INC START:     ; BMP MEM PNTR
                       BNE CHKEND
                       INC START:+1   ; BMP PAGE PNTR
                ;
                CHKEND LDA START:     ; ALL DONE ?
                       CMP ENDADR
                       BNE DATAOU     ; NO-CONT
                       LDA START:+1   ; MAYBE SO
                       CMP ENDADR+1
                       BNE DATAOU     ; NOPE-CONT ON
                ;
                       LDA GOADDR+1   ; OUTPUT GO ADDR TO TAPE
                       LDY GOADDR
                       JSR ADROUT
                       LDA #'G        ; OUTPUT 'GO' COMMAND TO TAPE
                       JSR OUTPUT
                ;
                       LDA #$00       ; RST CASSETTE OUT FLG TO VIDEO ONLY
                       STA OUTFLG
```

```
1520 0271 6CFCFF              JMP (WARMST)    ; GO TO MONITOR
1530 0274             ;
1540 0274             ;
1550 0274             ;
1560 0274             ;
1570 0274             ; COMMON ROUTINES
1580 0274             ;
1590 0274             ;
1600 0274             ; WORD & BYTE TO ASCII THEN TO TAPE
1610 0274             ;
1620 0274             ;
1630 0274 AA          ADROUT TAX              ; SAV A REG
1640 0275 A92E               LDA #'.          ; SELECT ADDR MODE
1650 0277 20EEFF             JSR OUTPUT
1660 027A 8A                 TXA              ; RESTORE A REG
1670 027B A202        DOWORD LDX #$02         ; TWO LAPS 'RND
1680 027D D002               BNE BYTOUT
1690 027F A201        DOBYTE LDX #$01         ; JUST ONCE
1700 0281             ;
1710 0281 48          BYTOUT PHA              ; SAV A REG
1720 0282 4A                 LSR A            ; MS NIBBLE FIRST
1730 0283 4A                 LSR A
1740 0284 4A                 LSR A
1750 0285 4A                 LSR A
1760 0286 2092D2             JSR HEXOUT       ; HEX TO ASCII THEN OUT TO TAPE
1770 0289 68                 PLA              ; GET A BACK
1780 028A 2092D2             JSR HEXOUT       ; LS NIBBLE OUT
1790 028D 98                 TYA              ; DO OTHER BYTE
1800 028E CA                 DEX              ; DONE ?
1810 028F D0F0               BNE BYTOUT       ; NO-CONT ON
1820 0291 60                 RTS
1830 0292             ;
1840 0292             ;
1850 0292             ;
1860 0292             ; HEX TO ASCII THEN TO TAPE
1870 0292             ;
1880 0292             ;
1890 0292 290F        HEXOUT AND #$0F         ; MASK OUT MS NIBBLE
1900 0294 0930               ORA #'0          ; HEX TO ASCII
1910 0296 C93A               CMP #':          ; >9 ?
1920 0298 9002               BCC Z9           ; NO-SO OUT TO TAPE
1930 029A 6906        AF     ADC #$06         ; 'A - F' SO ADD 6+C=7
1940 029C 4CEEFF      Z9     JMP OUTPUT       ; OUT TO TAPE
```

2P Extended Monitor Commands

MEMORY EXPANSION AND MODIFICATION COMMANDS

Commercial at(Shift P) NNNN - opens 16 bit memory location NNNN. 2PEM responds with "/CC" where CC is the contents of location NNNN. Contents of location NNNN may be changed by typing in 8 bit (byte) hexadecimal data DD. Additional commands in this mode are:

(LF) - (line feed) increments to next location.
(up arrow) (Shift N) - decrements to previous location
(") - prints ASCII or graphic character at current location.
(CR) - (carriage return) exits OPEN MODE.
D FRM, TO - Dumps a block of memory between 16 bit hexadecimal addresses specified by FRM and TO.
F FRM, TO=DD - Fill memory block specified by addresses FRM, TO with hexadecimal data byte DD.
M NEW=FRM,TO - Move the block of memory between FRM and TO to a block of the same length beginning at the hexadecimal address specified by NEW.
R NEW=FRM,TO - Relocates a block of executable machine code maintaining code executability. Format is the same as Move.

PROGRAM DEBUGGING COMMANDS
UNNNN - 6502 MACHINE CODE DISASSEMBLER

This instruction disassembles executable 6502 machine code into 6502 mnemonic code. Disassembly starts at the 16 bit hexadecimal address specified by NNNN and continues for 24 lines - a maximum of 72 bytes. Non-executable opcode are assigned a "mnemonic" of ???. Disassembly may be continued in 24 line blocks by pressing linefeed - (LF).

N HEX>FRM,TO - Searches memory block between hexadecimal addresses FRM and TO for the data string HEX. If a string match is found, the 2PEM enters, the commercial at OPEN mode and all commands in the OPEN mode are available. The maximum length of the HEX search string is 8 bytes.

W ASCII>FRM,TO, - Identical to N search except the search is made for an ASCII literal string match.
Bn,NNNN - Install breakpoint "n" at hexadecimal address specified by NNNN. Up to 8 (n=1 to 8) break points may be installed.

En - Eliminate breakpoint n.

T - Print table of breakpoint addresses. Note the address of a non-specified breakpoint defaults to FFFF.

C - Continue program execution from last break point. This command MUST ONLY be used with breakpoint entry into 2PEM.

I - Prints the address at which the 2P EXTENDED MONITOR was last entered by a break. The contents of the processor registers and the stack-pointer location are also printed.

A;X;Y;P;K - These five commands print the contents of the ACCUMULATOR, X INDEX, Y INDEX, STATUS and STACK-POINTER respectively. The commercial at OPEN mode is enteres and the contents of these registers may then be changed prior to continuing execution of program.

AUDIO CASSETTE COMMANDS

S FRM,TO - Saves the block of data between hexadecimal addresses FRM and TO in the following checksum format: LEN ADD DAT CHK. Where ";"

marks the start of the record, "LEN" is the length of the record (hex), "ADD" is the starting address of the current record, "DAT" is the data in the record and "CHK" is Record checksum. The output to the cassette stays on until an "L" command is executed.

L - loads into memory a block of data recorded in the above checksum format. If a checksum error is detected, "ERR" is printed. Stop the cassette machine, rewind past error, replay and type an "L" to re-enter load mode. Typing the space bar will exit the Load mode.

V - This command allows "Viewing" of material on the cassette with no memory modification. Typing the space bar will exit the view mode.

ADDITIONAL COMMANDS

GNNNN - Begin program execution at hexadecimal address NNNN.
DT1,DT2<+,-,*,1>=ANS - Four function hexadecimal calculator. Where DT1 and DT2 are 16 bit data fields followed by an operator <+,-,*,1> and ANS is the 16 bit result of the operation.

O - Prints the overflow or remainder of 16 bit multiplication or division respectively.

TECHNICAL NOTES ON THE 2P EXTENDED MONITOR

The 2PEM has been designed as a 6502 machine language programming design/debugging aid. And, since the 6502 is a hexadecimal number based processor, all numeric entries to the 2PEM are in hexadecimal notation.

In the remainder of this discussion,hexadecimal numbers will be proceeded by a "$" unless other/ wise noted.

PROGRAM MEMORY UTILIZATION

The 2P Extended Monitor resides in approx. 2K of memory - $800 thru $FB7.

Additionally, 48 Zero page locations ($D0-$FF) are used by the 2PEM. There is also a checksum loader used, for loading the 2PEM, at $700-$EF. This is not used after loading is completed. So, this block is free for u programs.

The 2PEM is normally entered at $800. This causes the stack pointer to be set (at $1F and the breakpoint registers to be initial Under certain circumstances, it may be des able to re-enter the 2PEM without initiali tion. This may be done at $81F.

Program Expansion.

There are three free command letters in th 2P Extended Monitor - J, U and Z.

In some advanced applications, a user may desire an additional function. This functi must be a machine language routine ending w a RTS. The function's call address is then serted into one of the following addresses J-$974; U-$98A; Z-$994.

For example to call a routine at $400 with $984 would be set to 00 and $98B to04.

PROGRAM OPERATING CHARACTERISTICS

The 2P Extended Monitor is similar to ROM in that, once the Save on cassette command been issued, all output is directed to the cassette as well as to the video monitor. condition is easily recognized by the relat ly slow speed with which data is transferr to the video monitor.

The SAVE function may be disabled by typing an "I" or "V" followed by the space bar.

In all 2PEM commands using the "FRM,TO" format, the "TO" is non-inclusive. For example, if the user wished to write 00 in locations $400 thru and including $4FF, the command "F0400, 0500=00" would be used.

In most 2PEM commands, user prompting is automatic. In the above example the "F" and "=" were output by the 2P Extended Monitor. However, in the case of the hex and ASCII string searches, (commands N and W), a ">" must be entered by the user to inform the 2PEM that "FRM,TO" data is to follow.

It should also be noted that there are no backspace entries while entering an ASCII string. If an error is made, return to the command mode is accomplished by typing ">" followed by any non-hex character.

The 2PEM command mode is prompted with the character ":" (colon). If a command error is made, the 2PEM outputs a "?" and re-enters the command mode.

DATA BLOCK MOVES VERSUS PROGRAM RELOCATION

The 2P Extended Monitor contains two very useful memory manipulation commands - Move and Relocate.

The Move command will non-destructively move a block of data in memory without changing any of the data. However, the Relocate command changes operands in order that they fit the area to which they are being relocated. For example, a program at $300 to $380 contains the following code (all hexadecimal):

```
0300        A520          LDA $20
0302        206E03        JSR$036E
036E        SDC6D0        STA$D0C6
037F        60            RTS
```

How, if the command "M0500=0300,0380" was issued, the code would appear as follows:

```
0500        A520          LDA$20
0502        206E03        JSR036E
056E        8DC6D0        STA$D0C6
057F        60            RTS
```

And, if the command "F0300,0380=00" was al issued, the program at $500 to $580 would not run since there is no subroutine at $5 However, if command $R0500=0300,0380" had issued initially, the code would appear as

```
0500        A520          LDA$20
0502        206E05        JSR$056E
056E        8DC6D0        STA$D0C6
057F        60            RTS
```

Note that the subroutine call at $502 was changed to fit the program's new location. However, if that subroutine had had an add outside the range of the location ($300 to $380), it would not have been changed. Suc was the case with the code at location $5

It should also be noted that not all progr are directly relocatable. For example, a program with data tables may not relocate properly since the 2PEM's relocation routi is likely to misread some data as executab op-codes.

In both the Move and Relocate routines, ca must be taken that the distance of a forwa move is greater than the length of the ori block. Otherwise, data in the original blo will be overwritten.

USING BREAKPOINTS FOR PROGRAM DEBUGGING

As the name implies, a breakpoint is a point where the execution of running program may be "broken". Using the 2P Extended Monitor's breakpoints, up to 8 breaks may be placed into a program prior to execution. Then, when the program is run and a breakpoint is encountered, the 2PEM is re-entered, printing the following:

```
@(commercial at)NNNN
n) X/CC Y/CC P/CC K/CC
```

where "n" is the breakpoint's number (1 thru 8), NNNN is the location where the break was encountered and the rest are the processors registers and their contents, "CC", at the time the break was encountered. These registers may be changed before continuing.

To illustrate the use of a breakpoint, consider the following portion of a program:

```
0350    B003    BCS$0355
0352    4C8003  JMP$0380
0355    200050  JSR$0500
```

If a breakpoint were installed at $350 - "1,0350", the opcode "B0" would be removed and saved by the 2PEM and the BRK code, "00", would be installed at $350. Then, when the program is executed, breakpoint 1 will be encountered and the 2PEM re-entered responding

```
@(commercial at)0350
0) X/00 Y/45 P/35 K/FC
```

Additionally, breakpoint 1 will be eliminated, the BRK code removed from location $350 and the opcode "B0" re-installed.

Examination of the status register (P) discloses

that the carry flag is set. The program would branch to $355 and JSR$0500.

If it would be more desirable to JMP$0380, the user simply issues the "P" command and changes the contents of the status register to $34, clearing the carry flag. Now, when the continue command is given, the branch will not be taken.

Breakpoint addresses and removed opcodes are stored at Page Zero locations $E8-$FF. Care should be taken not damage the locations with the program being debugged.

If the 2P Extended Monitor is entered by a BRK that was not installed as a breakpoint, a "?" will be printed. The location of the BRK may be determined with the "I" command.

2P Extended Monitor Command Reference List

COMMAND	FUNCTION	COMMENT
@	Open memory location NNNN	<LF>Nxt;(up arrow)Last "Pr.ASCII;/re-open
B	Print breakpoint Accumulator	Enters <commercial at> Open mode
B	Enter breakpoint n	n=1 to 8
C	Continue from last breakpoint	2PEM must have been entered by break
D	Dump memory FRM,TO	
E	Eliminate breakpoint n	
F	Fill memory block FRM, TODAT	
G	Go at Address NNNN	
H	Hexadecimal calculator	Four function <+,-,*,/>
I	Print location of last brk entry	
K	Print breakpoint stackpointer	Enters (commercial at) Open mode
L	Load memory from cassette	Checksum; any key exits
M	Move memory block NEW=FRM,TO	
N	Number string search; Hex>FRM,TO	8 BYTE Max.
O	Overflow/remainder from Hexcal	
P	Print breakpoint status register	Enters (commercial at) Open Mode
U	Disassemble from NNNN	<LF> continues
W	Relocate code NEW=FRM,TO	
X	Save memory block FRM,TO	Cassette checksum
Y	Print breakpoint address table	
V	View contents of cassette	Any key exits
W	Word string search;ASCII>FRM,TO	8 BYTE max.
X	Print breakpoint X register	Enters (commercial at) Open Mode
Y	Print breakpoint Y register	Enters (commercial at) Open Mode

Useful Subroutines In 2PEM:

INCH	Input to acc. with echo	$853
OUTCH	Output character in acc.	$861
PRBYT	Output byte in acc.	$AAC
CRLF	Output <CR>, <LF>	$807

Points, 65VK Monitor

ADDRESS	FUNCTION	USE
$FEE9	INPUT ROUTINE	RETURN CHR IN A FROM KEYBOARD OR TAPE
$FEDA	ROLL	MOV LSD IN ACC TO LSD IN 2 BYTE NUMBER $00FC+X X MUST BE SET BEFORE CALLING
$FECA	DIGIT	OUTPUT LSD (IN HEX) IN ACC TO SCREEN $D0C6+Y Y MUST BE SET BEFORE CALLING
$FEB0	OUI	OUTPUT (X) BYTES STARTING $00FC+X TO SCREEN STARTING $D0C6+Y X AND Y MUST BE SET BEFORE JSR X DECREASES, Y INCREASES
$FEAC	OUTPUT	OUTPUT ADDRESS DATA IN MONITOR FORMAT ADDRESS = ($FF,$FE)DATA=($FC)
$FE93	LEGAL	CONVERT CHR IN ACC FROM ASCII TO HEX IF CHR IS "0" THROUGH "9" or "A" THROUGH "F".ELSE RETURNS (ACC)=$80
$FE80	OTHER	INPUT CHR FROM TAPE AND RETURN CHR IN ACC
$FE00	START	MONITOR START ENTRY POINT

ADRESS	FUNCTION	USE
$FCCF	KEYBOARD LOAD (LDAKBD)	LOADS COMPLEMENT OF KEYBOARD AND RETURN IT IN ACC
$FCC6	KEYBOARD LOAD (LDXKBD)	SAME AS ABOVE EXCEPT RETURNED IN X-REG WITH CONDITION CODES SET ON THE CONTENTS OF X-REG. ACC IS PRESERVED
$FCBE	KEYBOARD WRITE (STAJBD)	WRITE COMPLEMENT OF ACC INTO KEYBOARD
$FCB1	ACIA OUT (OUTCH)	OUTPUTS CHR IN ACC TO TAPE
$FCA6	SERIAL INIT (SERINT)	INITIALIZES CASSETTE PORT
$FC9C	READ BYTE (READ)	READS A BYTE FROM DISC.BYTE RETURNED IN ACC
$FC91	TIME DELAY (MRT)	TIME DELAY SUBROUTINE Y & X LOST, Y & X RETURNED = 0, TIME DELAY = 1.25 Ms * X
$FC8B	UNLOAD HEAD (UNLOAD)	UNLOADS FLOPPY HEAD
$FC06	MANUAL BOOT (LOAD 0)	LOADS TRACK ZERO INTO $2200 UP AND RETURNS TO MONITOR
$FC00	AUTO BOOT	BOOTS OPERATING SYSTEM IN AND EXECUTES IT

Superboard II/C-1P Monitor Routines

BASIC Support Routines

Adress	Function	Use
$FFFE	IRQ VECTOR	6502 IRQ TO $01C0
$FFFC	RESET VECTOR	6502 RESET TO $FF00
$FFFA	NMI VECTOR	6502 NMI TO $0130
$FFF7	SAVE CMD JMP ($0220)	SETS I/O TO CASSETTE OUTPU VIA A JMP ($0220)
$FFF4	LOAD CMD JMP ($021E)	SETS I/O TO CASSETTE INPUT VIA A JMP ($021E)
$FFF1	CBTRK-C CHECK JMP ($021C)	RETURNS $03 IF CNTRL-C IS
$FFEE	BASIC OUTPUT JMP ($0218)	OUTPUT CHR IN A TO SCREEN, OR SCREEN & CASSETTE
$FFEB	BASIC INPUT JMP ($0218)	INPUT CHR INTO ACCUMULATOR FROM KEYBOARD OR CASSETTE
$FF00	COLD START "D/D/W/M"	RESET ("BREAK") ENTRY POIN

SUPERBOARD II - C-1P MONITOR ENTRY POINTS
65 K POLLED KEYBOARD ROUTINE

Adress	Function	Use
$FD00	KEYBOARD INPUT	RETURNS ASCII OF CHR ENTER AT KEYBOARD. HOLDS UNTIL A IS PRESSED.

Bringing up BASIC

Once you have dabbled a little with machine language programming, you will be eager to get on with full BASIC. If your machine is equipped with BASIC in ROM, it will not be necessary to connect any additional devices in order to BASIC. Simply reset the computer and type "C" in response to "C/W/M?". The computer then asks "TERMINAL WIDTH?" to which you may also reply with a carriage return. The BASIC prompter, "OK" will then come up indicating that BASIC is directly accessible. You can then proceed to the BASIC sample program in this section and the BASIC Operating Manual attached to the end of this manual. You may then wish to connect an audio cassette interface, if one is present in your system.

If your computer is configured for floppy disk, it will be necessary to connect a disk drive before you can bring BASIC in. The computer must have at least 16K RAM, floppy disk bootstrap PROM (indicated by the message "D/M?" when the computer is reset), and a D/M floppy disk controller board present in the computer. When the floppy disk controller board is present in the computer, it is usually the rear-most board in the computer system. With the computer turned off, connect the ribbon cable coming out of the rear of the floppy disk drive in one of the openings in the rear of the Challenger and mate it with one of the connectors coming out of the back of the 470 Board.

This should be accomplished such that the ribbon cable falls into the case instead of sticking up out of it. The boards should be mated tightly together and the connector should be backed off about 1/8" to preclude

the possibility of the Molex pins touching the PC board foils on the A-12 adapter boar of the cable. That is, you should be sure that the Molex pins on the 470 Board are no touching the A-12 adaptor cable board. Then make sure that all parts of the computer ar plugged into a common three wire grounded outlet or distributor box on one circuit. Then power up the floppy disk drive. Place a diskette with OS-65D in the upper disk drive with the label side up and the notche side in first. Follow the dialog and procedures on pages 1 - 4 of the OS65D Version 2.0 Manual. After you have obtained the BASIC prompter "OK", proceed with the examp in this manual, if desired.

If your system is equipped with neither the floppy disk nor BASIC in ROM, you must load BASIC via paper tape or audio cassette. You must have at least 12K memory to do so. Paper tape versions of BASIC are specifical designed for use with Teletypes while the audio cassette version is for use strictly with video-based computer systems. Follow t instructions at the end of the 8 K Basic on paper tape. Follow the instructions included here on the use of Auto-Load^{tm} audio cassettes, and then proceed to the instructions for loading 8K BASIC audio cassettes the end of the 8K BASIC User's Manual. This procedure is only necessary if you do not have BASIC in ROM or disk BASIC.

Introduction to Small Computer Software

In order for a computer to perform even simple operations, it obviously needs a means by which the user can communicate instructions to it. Any such means which consists of a set of rules to convey information, is called a computer language. The numerous languages in use today offer a wide range of specific applications and varying degrees of understandability for the user. They can provide direct communication with the computer at the complex level of machine language, or enable the programmer to use an indirect communication by means of a higher level language which corresponds more closely to human speech.

Machine level languages are really the most practical device from the computer's point of view, because when you use them, you are really speaking the computer's own jargon, and are thus making more efficient use of memory space. On the other hand, when you use an upper level language, every instruction you give, in what resembles "plain English", has to be converted into one or more separate instructions in a machine level language. Therefore it is obviously less wasteful in terms of time and effort to write in machine language, and skip translations from other languages altogether. The major drawback in machine languages, however, is that they are difficult for the average person to learn.

Machine languages consist of binary codes used in all of the commands which are entered by the programmer. These codes are 8-bit groups of on-or-off switches, which in various combinations, serve as instructions for the computer. Although the majority of users will have

no need or desire to learn these combination
there are some who, for one reason or anothe
will want to program their computer directly
Since it is quite troublesome to commit se-
veral dozen combinations of numbers to memor
a system of abbreviations (mnemonics) has be
devised which exactly correspond to machine
language instructions. The program which
converts these mnemonics into machine langua
is called an Assembler. By following the in-
structions provided with your computer, you
can make use of the Assembler and write a
program in mnemonic code. This is directly
translated into the binary object code which
the machine understands. After you gain pro-
ficiency, you can even begin to use the
actual object code to do your programming,
examine and change memory locations, etc.,
and thus be in ever greater control of your
machine.

Upper level languages, in contrast to machin
level languages, are much easier for humans
to master. Nevertheless, every upper level
language has to originate at the machine
language stage, and usually represents a
lont, tedious effort on the part of the auth
or authors of that language. Probably the
most common upper level language is BASIC
(Beginner's All-Purpose Symbolic Instruction
Code). An 8K version of BASIC written by
Microsoft, Inc. (i.e. it occupies 8×2^{10}
locations in memory) is used in all of OSI's
6502 computers.

Because of BASIC's popularity, simplicity,
and versatility, OSI has made it a standard
feature in its product line, either by
placing it in a computer's permanent memory
(also known as Read Only Memory (ROM), which
does not "forget" once the power is turned o
or by reserving special tracks for it on

floppy or hard disks. In addition, in OSI products, BASIC is always immediately available to use, because it comes up automatically the instant the computer is reset. Therfore, the programmer is free of the burden of manually bringing in BASIC, which would demand that he be thoroughly versed in the computer's internal thinking processes and machine language. The fact that BASIC comes up automatically is very convenient for computer programmers, most of whom probably have programs they would like to run or write in BASIC.

There are a large number of publications available which describe in detail the commands and functions of BASIC. While this introduction can in no way duplicate such excellent manuals as Schmidt's outline series Programming with BASIC (McGraw-Hill), it can at least give you some insight into the method for writing your own programs in BASIC.

Refer to the instructions provided with your individual unit to bring up BASIC in the OSI Challenger. Establish the memory size and terminal width for your particular program. When you see an OK appear on your video monitor or terminal, the computer is ready to start accepting BASIC commands from the keyboard.

Every statement in your program must begin with a statement number. These need not be typed in numerical order, since the computer will automatically re-arrange them according to statement number when you have finished typing the program. But they must be numbered in the same order in which they are to be run. In OSI's 8K BASIC for the 6502, a variable can consist of one or two characters. If longer variables are to be used,

BASIC will recognize only the first two ch
acters. The first character in a variable
must be alphabetic. The second character,
present, may be either alphabetic or numer
Functions, commands, etc., already used by
BASIC must not be employed as variables. I
order to set a variable equal to a desired
value, e.g., Z equal to 10, you use the LE
statement, as follows:

(line number ——>) 20 LET Z=10

Since LET is optional in OSI's 8K BASIC,
may also type:

20 Z=10

You may wish the value of the variable to
change each time you run the program, with
having to rewrite the whole program every
time. To take advantage of the option to
alter the values of variables, you make us
of the INPUT statement, for example, as
follows:

```
10 INPUT A,B,C
20 LET X=A
30 LET Y=B+C
```

In this way you can cause X and Y to take
different values each time the program is
run. Later, when you do run the program, y
will see a ? on the terminal. You then typ
the values for A,B and C which are relevan
to the particular program. If there are no
other INPUT statements in the program, it
will begin to run immediately with the val
you have entered, unless some built-in con
prevents this. If the program contains add
tional INPUT statements, BASIC will keep
asking you (by means of a ?) to input what
ever data it needs to run the program, unt
each INPUT statement has been answered.

It can be that a variable has a value which is to change at a regular rate during the course of a single program run. This will require you to set up a loop which makes calculations using these increasing or decreasing values each time a new value is employed. For this you need to use a FOR-NEXT loop. This loop begins with a FOR statement and ends with a NEXT statement. The FOR statement identifies the initial and final values of the variable in question, and includes the constant amount of increase or decrease:

```
100 FOR Z=10 to 20 STEP 3
110 LET A=Z+(2*4)
120 NEXT Z
130 (resumption of program)
```

Step 3 means an increment of 3 upon each pass through the loop. Therefore, the above FOR-NEXT loop will be run four times, namely, when Z=10,13,16 and 19. When the value of Z exceeds 20, BASIC resumes the program by going to the first statement following the FOR-NEXT loop. In addition to signifying the end of the loop, the NEXT statement also contains the variable identifying which loop it terminates. As you may later discover, this is most useful in nesting one loop inside another.

Sometimes you will want the program statements to be run in a different order, if a certain condition is met. In order to change the order of execution, you may use the IF... GOTO statement, for example,

```
100 IF X=10 GOTO 150
110 (another program line)
140 (another program line)
150 LET Y=X+5
```

Here, the IF ... GOTO diverts execution to a non-consecutive statement, line 150, omi ing lines 110 and 140, provided only that value of X is equal to 10. If X is not equ to 10, the program would resume with line 110. A simple GOTO command may also be em- ployed without an accompanying IF, if no condition must first be met.

An IF ... THEN statement is used to jump to a statement other than the one directly following. It can also be used to issue an other statement allowed in BASIC. For exam

```
100 IF X>10 THEN PRINT "X IS GREATER THAN
```

This will cause the terminal to display X GREATER THAN 10 only if X>10. The program then proceed as normal, with the next cons cutive line. If X≤10, the program will, of course, proceed as normal, ignoring the PRINT command.

A PRINT statement will cause the terminal display whatever follows. If you type:

```
100 PRINT A
```

the value of some previously defined varia A will be printed. If you type:

```
100 PRINT "A"
```

the simple letter A will be printed.

The END statement terminates the program a allows you to run the program, change it, or start to write a new program. As in the case of the LET statement, the END stateme is optional in OSI's 8K BASIC.

If you want to erase a program and start a new one, simply type NEW and enter your ne program.

BASIC is provided with a large number of mathematical functions, such as sine (SIN(X)), square root (SQR(X)), and absolute value (ABS(X)). These functions automatically cause the computer to calculate the pertinent value without figuring by the user. For example, in the following statements:

```
100 X=121
110 PRINT SQR(X)
```

the value of the square root of 121 will appear on the terminal when the program is run.

At any time while entering your program, or after you have finished entering it, you can list all the statements up to that point by typing LIST. You can thus list the whole program, or by typing a specific line number after LIST, such as LIST 140, you can display just the one line. If you desire to see a certain block of program lines only, then you can specify the desired range, such as:

```
LIST 100 TO 200
```

If you want to correct a line previously typed, simply type the correction, using the same line number. It is recommended that you number the program statements by jumps of 10 rather than consecutively, so that you can later easily insert additional lines if you wish. To do this, type a line number which falls between the interval where you want the new statement to appear, and add the missing line. If you want to delete a line, simply type the line number, then <return>. By using the LIST command, you can easily verify any changes you have made. This will cause every program line to scroll up the terminal, with each line number in

consecutive order. If you want to stop the scrolling, type Control-C, examine the listing to your satisfaction, then type CONT (= continue), after you see BREAK IN LINE on the terminal.

The following example gives an illustration of editing procedures: Suppose you want to modify the following statements:

```
 90 INPUT A
100 LET X=2*A
110 PRINT "THIS IS A PROGRAM."
120 PRINT "EXAMPLE"
```

If you want to insert a line Y=A between lines 90 and 100, you could, at this point type:

```
91 LET Y=A
```

If you want to delete line 110, simply type 110, then <return>. If you want line 100 to read LET X=3*A, simply type the correction using the same line number. At any time you could confirm the alteration by typing LIS

Following these corrections, if you are ready to run the program, type RUN <return> If your program contains any INPUT statements, you will now see a ? on the terminal Type in the data desired, as explained above, and the program will run. Following program execution, you can start over again by typing RUN, or enter a new program by typing NEW.

The following sample program demonstrates the INPUT, LET, PRINT, GOTO and END statements, the FOR-NEXT loop, and the IF ... GOTO command, as well as the SQR function.

Problem: Print the square root of a number; increase the number by five six times, and each time print the square root. If the largest square root is less than twice the first square root, indicate this. Otherwise, indicate only the fact that the program prints square roots.

After the programmer has typed the above program in BASIC, he will see an OK on the screen, signifying that the computer is ready for the next command from the user. If he wants to run the program, he types RUN. The computer will show a ? on the terminal. The user types on the keyboard that number with which he wants to begin the program. The six values (with constant increments of 5, see line 40) will scroll up the screen, each accompanied by its square root. If by chance you have made an error in typing (not including improper spacing), you will probably see an error message on the screen. If you do, simply edit the line containing the error, as explained above. You can always run the program again by typing RUN. Here are the lines of the program:

```
10 INPUT A
20 LET Y=SQR(A)
30 LET Z=A+30
40 FOR X=A TO Z STEP 5
50 PRINT "THE SQUARE ROOT OF";X;"IS";SQR(X)
60 PRINT
70 NEXT X
80 IF SQR(Z)2*Y GOTO 110
90 PRINT "THIS PROGRAM PRINTS SQUARE ROOTS."
100 GOTO 120
110 PRINT "THE LAST ROOT IS LESS THAN TWICE
    THE FIRST ROOT."
120 END
```

BASIC and Machine Code Interfaces

In the process of working on some utility programs for OS-65U, a very important point came to light. That point being that the best compromise between ease of programming and speed of execution can often be obtained by aBASIC/machine code combo. This article will attempt to illustrate the mechanics of interfacing BASIC and machine code. There are several steps to the interfacing and they are:

1. Define the job to be accomplished.
2. Define the functions to be implemented in machine code.
3. Define the functions to be implemented in BASIC.
4. Create and debug machine code.
5. Debug the BASIC program.
6. Debug the BASIC/machine code interface.

The job to be accomplished:

Generate a random star pattern on the screen with a super fast screen clear.

Functions to be handled in machine code:
Super fast screen clear
Functions to be handled in BASIC:
Generate random star fields

The machine code:
The machine must clear the screen as fast as possible. This can be easily accomplished by storing the ASCII code for a space into the video memory. The video RAM resides from $D000 for 2K ($D000 through SD7FF). Therefore the routine would be as follows:

In line 120, the accumulator is loaded with the ASCII code for a space. In line 130, the Y register is loaded with the number of pages to be cleard (each page equals 256 memory lo-

rations). In line 140, the X register is set to zero.

The 6502 has a form of addressing called indexed absolute. What this means is that a base address immediately follows the op-code. This base address is added together with the contents of the X register to form the real address. So in line 150, the code STA SCREEN, X really means store the contents of the accumulator at the address $D000 plus the contents of the X register. The first time through this loop, a space would then be stored at $D000+X or $D000. The X register is now incremented by one in line 160. Then line 170 tests to see if the X register has been incremented through until it "rolls over" to zero (the X register can only represent up to 255, i.e., if X=255 and one is added to X, then X "wraps around" to zero). Therefore, the inner loop consisting of lines 150 thru 170 clears one page of memory at a time. So how do we clear more than one page? We should simply duplicate the code in line 150 thru 170 eight times, (once per page), but that would be poor programming. O.K., so what arethe alternatives? If the inner loop is examined closely, one can see that the only difference between a routine to clear any of the eight pages is the base address (SCREEN). Therefore, by simply incrementing the base address (SCREEN) each time we have cleared one page, we can clear the entire eight pages using the inner loop. To implement this "clear screen" routine, one then only needs make a slight modification to the "clear page" routine. That is the function of lines 180 thru 200. Line 180 increments or adds one to the page pointer (the high byte of screen, shown as $D0). The Z register is then decremented by one in line 190 and a branch, if not equal zero (BNE) is used in line 200 to branch back to the point called "LOOP". This branch occurs until Y=0 or all eight pages of memory have been cleared. The final step in this

program is a simple "housekeeping" function, Lines 210 thru 230 are used to "reset" the base address "screen" to point back at the beginning of the video memory. This is required because of the fact that "screen" is incremented by this program and must be "reset" so that this routine may be used more than once.

On to the BASIC program. As can be seen from the listing, the "star field" portion of the BASIC program is very straightforward. The FOR-NEXT LOOP uses the RND (Random Number Function) to randomly "POKE" a period onto the screen. The BASIC program uses a simple "FOR NEXT" LOOP to POKE random video RAM locations with a period. The code responsible for this appears as follows:

```
FOR X = 53312 TO 55231
POKE X + INT(RND(X)*63), ASC(".")
NEXT X
```

So much for the BASIC program. Now for the BASIC/machine code interface. The first question to be answered is how the machine code will get into RAM. The easiest way to accomplish this is by placing the decimal values of the machine code program in data statements. Then by using the POKE statement in a "FOR-NEXT" LOOP, the machine code can be POKED into memory. All that remains to be writtenis the statements that "POKE" the USR vector to point to the start of the machine code routine. The program sequence will execute in the following order:

1) POKE the machine code program into high memory. (This memory is "FRE-ED" by setting memory size to 4050).
2) POKE the USR vector to point at the star of the machine code program.
3) POKE the star field

4) Clear the screen via the USR function,
   i.e., X=USR(X)
5) Loop back to number 3

One final note:
The sequence for the USR function is the
following:

1) The statement X=USR(X) is executed
2) Jump to the machine code program occurs
3) The machine code program clears the screen
4) Finally, a return from subroutine returns
   execution back to BASIC

```
0000                ; CLEAR SCREEN SUB
0000                ;
0000                ; EQUATES:
0000                SCREEN=$D000
0000                PAGONT=$08
0000                SPACE=$20
0000                ;
0000                ;
0FE8                *=$0FE 8
0FE8                ;
0FE8                ;
0FE8 A920    ENTER  LDA #SPACE
0FEA A008           LDY #PAGONT
0FEC A200           LDX #0
0FEE 9D00D0 LOOP    STA SCREEN,X
0FF1 E8             INX
0FF2 D0FA           BNE LOOP
0FF4 EEF00F         INC LOOP+2
0FF7 88             DEY
0FF8 D0F4           BNE LOOP
0FFA A9D0           LDA #SCREEN/256
0FFC 8DF00F         STA LOOP+2
0FFF 60             RTS
```

```
10  REM BASIC/MACHINE CODE INTERFACE
20  REM
30  REM MEMORY SIZE MUST BE SET TO 4050
40  REM MAC CODE POKED INTO RAM
50  REM
60  FOR PTR=4072 TO 4095
70  READ MAC: POKE PTR,MAC
80  NEXT PTR
90  REM
100 REM POKE USR VECTOR
110 POKE 11,232: POKE 12,15
120 REM
130 REM STARFIELD CODE
140 FOR X+53312 TO 55231
150 POKE X+INT(RND(X)*63),ASC(".")
160 NEXT X
165 REM CODE TO CLR SCREEN
170 FOR TIME=1 TO 500: NEXT TIME
180 X=USR(X): GOTO 140
190 REM MAC CODE STORED IN DATA STAT.
200 DATA 169, 32, 160, 8, 162, 0, 157, 0
210 DATA 208, 232, 209, 250, 238, 240
220 DATA 15, 136,208, 244,169,208
230 DATA 141,240,15, 96
```

CA-15 UNIVERSAL TELEPHONE INTERFACE

The Universal Telephone Interface (UTI) provides the broadest range of computer/telephone utilization options ever offered in a single product. The UTI occupies one slot of a C8P, C2-OEM or C3 series computer and connects directly to a normal telephone line via a FCC approved isolation module called a CBT. CBT's are available from many telephone companies on a monthly charge basis, however, Ohio Scientific also offers CBT's for user connection to telephone lines. The UTI can be connected in conjunction with one or more telephones on the line and can also operate as the only device on the line. No manual intervention is required to initiate or answer calls. The UTI is compatible with Touch Tone or Pulse Dial (rotary dial) lines. The UTI's features are summarized below.

Computer to telephone interface

- Initiates connect
- detects ring (incoming calls)
- detects dial tone
- detects busy signal
- automatic failsafe hangup after 90 seconds of inactivity
- Initiates disconnect (hangups)
- pulse dialer
- touch tone dialer

Output to telephone multiplexer

- outputs touch tone
- outputs 300 baud modem signals via on-board modem with "originate" or "answer" protocalls
- outputs taped message with motor control signal
- outputs audio from auxillary jack
- outputs Votrax ® generated synthetic speech when Votrax equipped (also has amplifier and circuitry to operate Votrax stand alone)

Input from telephone multiplexer

- decodes touch tone signals
- accepts 300 baud modem signals via on-board modem in either "originate" or "answer" protocalls
- routes audio to auxillary tape recorder with motor control
- routes audio to auxillary audio output

The UTI's abilities coupled with Ohio Scientific's broad line of computer accessories open up new areas of computer applications utilizing "plug together products", Here are just a few of the products applications,

Computer to computer communication

- can operate as a conventional modem with hands off operation
- allows OSI computers to be remotely timeshared economically
- can be used as the basis of computer bulletin board services
- two UTI based computers can automatically interrogate each other for
  * remote process control
  * remote data acquisition

Computer to human communication

- when coupled with OSI's security monitor it can provide a complete security system
  * monitor home, business, warehouse, vacation home, boat, etc,
  * can notify any telephone number(s) with Votrax or tape recorded message

Human to computer communication

- touch tone decoder allows the caller's telephone to act as a
  "computer terminal"
  caller interrogation of security or systems status
- when used in conjunction with the A.C. remote control and/or parallel I/O it allows

* caller control of home lights and appliances
* caller control of automated processes

when used with a remote rewind tape deck, endless tape or A/D D/A system it can relay voice messages
* caller initiated playback of recorded messages
* caller stored message for relay to another number at another time (and/or attempted until message gets through)

The UTI comes complete with documentation on how to use each feature as well as the following programs:

home monitor demo allows a touch tone equipped caller to interrogate the AC-17P home security system status and to send commands to AC-12P A.C. remote control (under OS-65D)

automatic dial modem program allows UTI equipped computer to function as a timeshare terminal with hands off modem operation (under OS-65D)

- timeshare user protocall for Level 3 system allows one partition (per UTI) of a Level 3 timeshare system to be available for telephone communications (overlay to OS-65U Level 3)

- remote computer interrogation: demo software for two UTI equipped computers. One machine automatically interrogates the other, useful as the basis of remote monitoring and process control systems.

NEW PRODUCTS FROM OSI COMMING IN MID 1980

CA-20
through CA-25 This new family is based on the 16-pin BUS connector on the back of C4P MF and C8P DF products and used on the CA-12 96 line parallel I/O.

Product		Availability
CA-20A	8 Port Board	April 1980
CA-21	48 Line I/O (from CA-12set)	April 1980
CA-22	High Speed Analog I/O	April 1980
CA-25	Security and A.C. Interface	April 1980
CA-20	8 Port Board with Calendar Clock	May 1980
CA-24	Interface Board	June 1980
CA-23	PROM Blaster	July 1980

Software	
Process Control Basic	June 1980
Security Basic	June 1980
OS-Vocalizer I	July 1980
OS-Vocalizer II	July 1980

PROMs & ROMs:

65 A Serial Monitor
65V Video PROM Monitor for 440B Board
65V2P Video PROM Monitor for 540 Board
65-500F2 Floppy Disk Bootstrap Monitor PROM
CG4 Upper and Lower case and graphics character generator for 540 Video Board
65AB 8K BASIC in ROM for Serial System & Support E-PROM (You will still require 65A)
65VB 8K BASIC in ROM for Video Systems & Support E-PROM (Designed for use with 440B ONLY--Still require 65V)
65VB2P 8K BASIC in PROM for 540 Based Video & Support E-PROM (Designed for use only with 540, still require 65V2P).

RS-232 Interface for Challenger C-1P

Parts List

The following parts need to be obtained to perform the modification. These parts can be obtained from a general electronics supplier, (Radio Shack, Lafayette, etc.) at a nominal cost. If you have any questions, ask the salesman.

Part #	Part
R1	470 resistor, 1/4 watt, 5%(Yellow, Violet, Brown, Gold)
R2	10K " " "
R3	10K " " "
R4	10K " " "
R5	4.7K " " (Yellow, Violet, Red, Gold)
R6	1K " " (Brown, Black, Red, Gold)
D1	1N914 Diode, silicon switching diode, Radio Shack # 276-1122 or equiv.
T1	NPN Transistor, gen. purpose amp and high speed switch, Radio Shack # 276-2009 or equiv.
T2	PNP Transistor, gen.purpose amp and high speed switch. Radio Shack # 276-2023 or equiv.
S1	Switch, SPDT, Radio Shack #275-662 or equiv.
P1	DB25S or other required serial interface connector (buy whatever you need to interface your device)
P2	12 pin male molex connector, .156 in. o.c., 7 pin or more will work (same

	kind that is used to connect video and tape I/O to the board)
Misc.	Solder and Soldering iron
Misc.	22 ga. stranded wire or similar hook up wire.

DISASSEMBLY

Disconnect the power cord, cassette leads and the video lead. Remove the six screws on the bottom of the case and remove the cover. Mark the nylon molex connector on the rear of the circuit board with an identifying mark to facilitate reassembly. Unplug that connector and the connector to the power supply.

Remove the screws that hold the circuit board to the case. Remove the circuit board and place it on a flat non-conductive surface.

CIRCUIT BOARD MODIFICATION

Before you begin, do some reading and practicing of soldering techniques. This modification requires good clean solder connections.

Refer to figure 1; it is a picture of the upper center of the rear of the circuit board. Use this figure as a reference in locating the correct location of the six resistors, one diode and two transistors. Solder points A-T are identified in the figure. You can use sheet 6 of the circuit diagram in the appendix of your Users Manual as an information aid in this modification.

All six resistors are installed standing up; that is bend one lead of each resistor 180 degrees, (see figure 2).

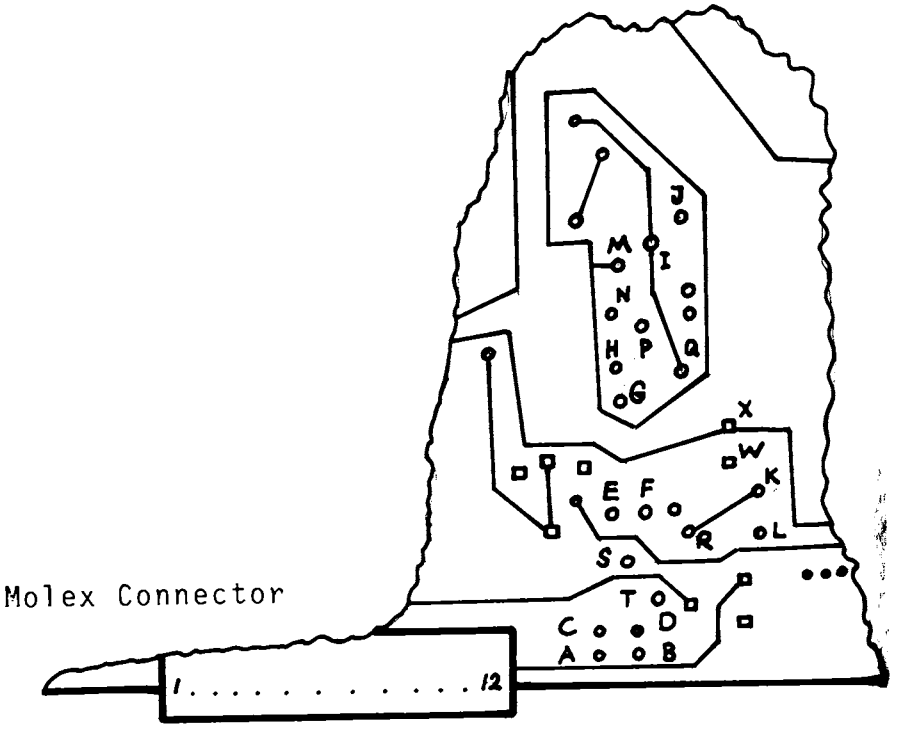


Figure 1

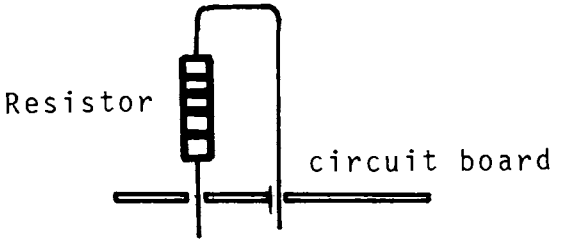


Figure 2

Part #	Solder Points (refer to figure 1)
R1	A,B
R2	C,D
R3	E,F
R4	G,H
R5	I,J
R6	K,L

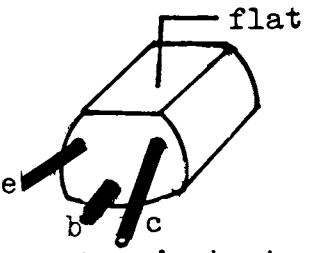
The resistors are to be installed as follows:

A. Bend resistor leads as mentioned above.
B. Insert resistor in top of circuit board. Make sure the leads go in the correct holes.
C. Solder the leads from the underside of the circuit board.
D. Clip off the excess lead length.

The diode is installed as above except when bending the lead, be careful not to break the glass case. The diode is installed between points "M" and "N". The end of the diode with the stripe must be inserted at point "N".

The transistors have three leads: e, b and c. (see figure 3)

Figure 3



Install them about a quarter inch above the circuit board as follows:

Part #	Lead	Solder Point
T1	e	O
T1	b	P
T1	c	Q
T2	e	R
T2	b	S
T2	c	T

The final step is to scratch out the circuit board surface wire on the bottom of the board between points "U" and "V". Use a sharp knife. Scratch out enough to make sure current cannot flow.

CONNECTOR AND SWITCH FABRICATION

The first six pins on the molex connector, part # P2, are to be connected as follows:

PIN #	To Location
1	Pin # 1 of DB 25S connector, part #P1
2	PIN # 2 of DB 25S connector, part #P1
3	Pin # 3 of DB 25S connector, part #P1
4	Pole of SPDT switch, part # S1 (generally center terminal)
5	Out 1 of SPDT switch, part # S1
6	Out 2 of SPDT switch, part # S1

To wire this, cut six wires about 12 inches long each. Then solder ends of each wire as indicated.

ASSEMBLY

Mount the switch, S1, and the connector, P1, to the back of the computer cabinet.

Plug in the new molex connector into the center rear of the circuit board. Refer to figure 1 for location of pin # 1.

Plug all other connectors into the circuit boar
Reinstall the circuit board into the cabinet.
Reinstall the cabinet cover.
Reinstall all video and tape leads.

OPERATION

The installed switch is used to control arriving data. In one position, data will come in through your new RS-232 C interface. The LOAD command will allow the computer to receive data.

Data transmitted through the SAVE command will be routed to both the cassette interface and the RS-232C interface. So, for example, if you hooked up the RS-232C interface to a printer, whenever a SAVE was in effect, all information put on the screen would also be printed.

The DB25S connector as wired will have the following pin definitions:

Pin #	Definition
1	Ground
2	Computer transmits through this pin
3	Computer receives through this pin

Some serial hardware may have different pin definitions. e.g. pins 2 and 3 may be reversed.

The voltage swing of the transmitted signal is 0 to + 5 volts. Some hardware may require a negative voltage swing. To do this you will need to scratch out the surface wire on the underside of the circuit board between points "W" and "X" and add a negative voltage supply to pin # 7 of connector P2. A standard nine volt transistor battery tied between pins 1 and 7 (negative on 7) will do. Put a switch on it to stop the power drain when not in use. Out of four things I have tied the interface to, only one required this modification.

600 Baud Cassette and Printer Conversion for the C1-P and Superboard

PARTS

The parts list is short. You will need (1) a DPDT switch, (type is not critical - chose for size and price), (2) enough wire to run 6 ten to 12 inch runs and (3) one .047 microfarad capacitor.

You will need something to cut a foil path - an Exacto knife or razor blade is best, a soldering iron, solder, wirecutters, a screwdriver and whatever hardware you need to mount the switch to the case completes the list (A VOM is handy to have).

STEP 1 - DISASSEMBLY OF THE C1-P

Remove the six screws on the bottom of the cas Remove the bottom and mark the connector at the rear left edge of the case for easy reassembly. (If you don't mark it, you'll forget that the blank pin is in the fourth from the outside edge and won't know how to put it back in.) Remove the six screws around the keyboard unplug the connector to the recorder and TV, and set the case on edge so that you can lay out the circuit board next to it with the part side up.

STEP 2 - IDENTIFY THE PARTS

We need to find three chips and a capacitor. Place the board so that the keyboard is to you right and the cassette interface connector is to your left. From that position, the 74LS163 just to your side of the fuse is U59. Now look back at the metal can that holds the crystal, look to the right, just past the smal

proto area, and you will see another 74LS163 at the bottom edge of the board. That is U30.

Now verify that a trace runs between pin 14 of U59 and pin 2 of U57. It takes off from the top of the board underneath the chip at pin 14 of U59 and runs through two plated through holes to pin 2 of U57. IF YOU DON'T KNOW HOW TO TELL WHICH PIN IS WHICH - STOP - GET HELP BEFORE YOU HURT YOUR COMPUTER !!

The final part we have to identify is C 11. It is a .1 microfarad capacitor which sits at the rear of the board (if you haven't already done so, put the board back into the same position it was to start - keyboard at your right.) near the cassette connector (to your left). You will see a pot next to the connector and a .1 microfarad capacitor in the corner made by the pot and a 74123 chip. Note that one lead of the capacitor has two extra holes connected to it by a foil run (see DIAGRAM # 2).

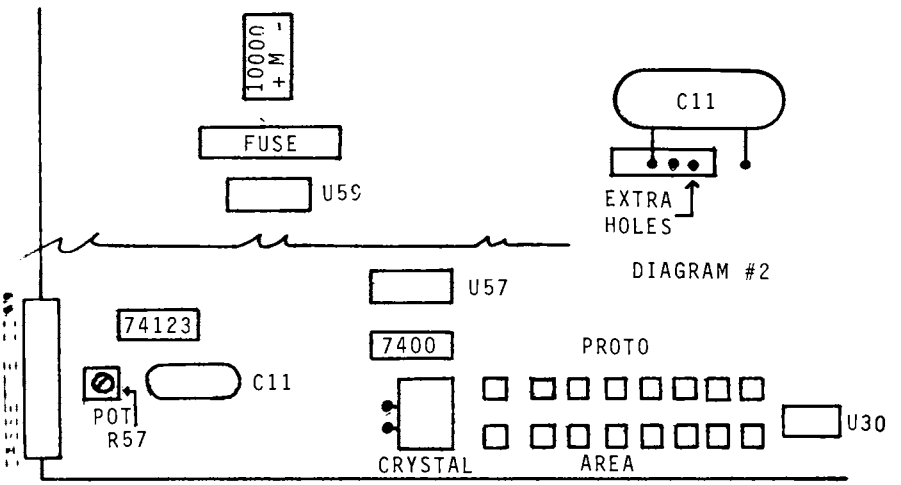


DIAGRAM #1

STEP 3 - COURAGE

Now we cut a foil trace. We have to cut the line between pin 14 of U59 and pin 2 of U57. I suggest that you cut it between, but close to, one of the plate through holes. The closeness to the plate through will make it easy to repair if you decide to go back to stock configurations and by cutting between the holes, you leave one attached to each pin, making it easy to attach wires to those pins.

STEP 4 - CHANGE CAPACITORS

Carefully heat and lift the end of C11 that ha only one hole attached to it. Leave the other end attached to the board. In one of the plate through holes next to the end you left attache solder in one lead of the .047 MF capacitor. Leave the other end loose for a moment.

STEP 5 - OPTIONS YOU HAD BETTER TAKE

Strictly speaking, you could skip this step, but you could end pulling foil loose eventuall We are going to run wires to the proto-pad beside the crystal to prevent strain on the foil runs. It doesn't matter which pads you use (just keep track), but run a wire from the lifted end of C11 to one pad, from the free end of the new .047 MF to a second pad, from pin 2 of U57 to a third pad, from pin 14 of U59 to a fourth pad, from pin 11 of U30 to a fifth pad (note - a trace runs from pin 11 of U30 under the chip and to a plate through hole about half an inch to the left of the chip. It is a hany place to put a wire, but as with the other plate through holes, verify with an ohmmeter first. If you can't verify that you have the right one, attach to the pad at the pin.), and from the hole you re-moved the lead of C11 from to a sixth pad.

STEP 6 - WIRE THE SWITCH

Wire up the DPDT switch following DIAGRAM #3. The switch is symetrical, so it won't matter if you look at the diagram as though it were from the top or the bottom. Wires should run from the switch to the other hole in the appropriate proto pad. Use minimum wire length to the switch

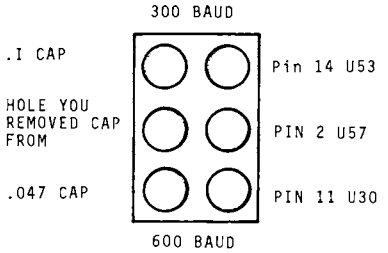


DIAGRAM #3

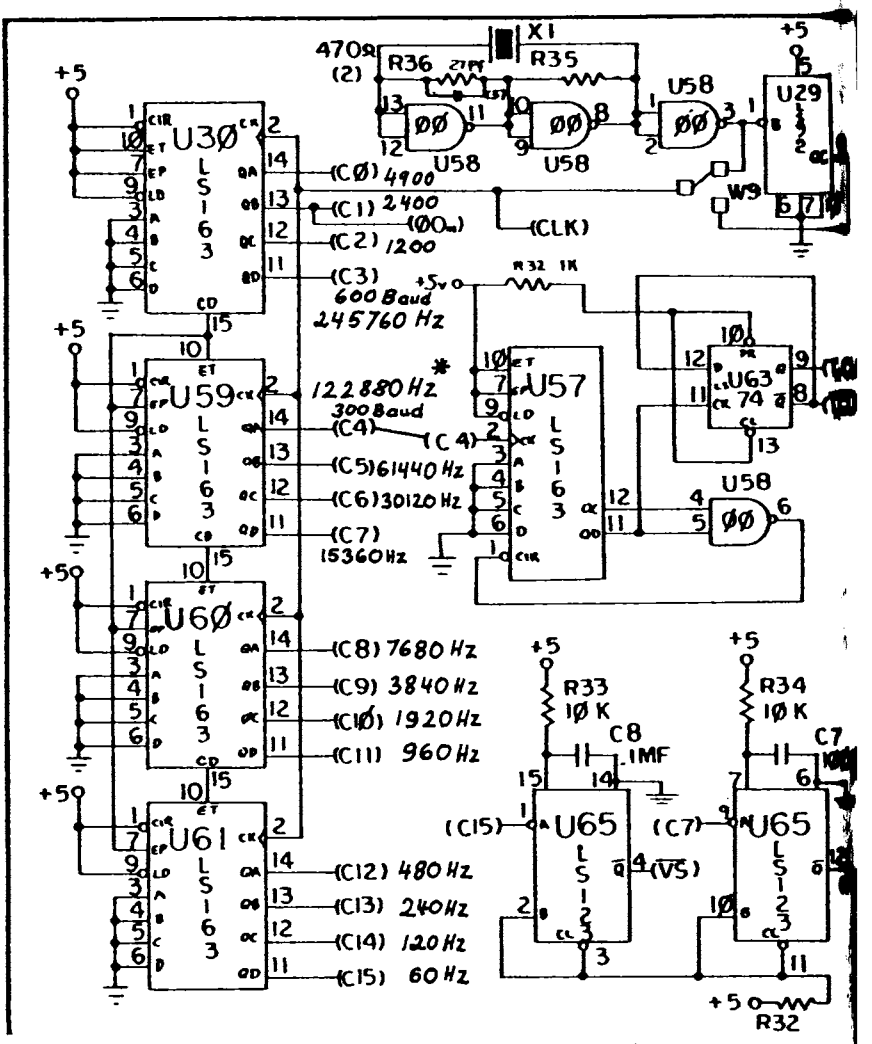
STEP 7 - THE HARD ONE

The hard part is often the mechanical placement of the part. Install the switch in a firm position. I used one of the circular holes in the C1 case to hold it.

STEP 8 - MAYBE

The cassette interface is a little more choosy about duty cycle on the 600 BAUD setting, so that it may be necessary to adjust R57 to get reliable reads. (That's the pot right next to C11). This affects only receive, so you can skip this if you are using this only to interface with a printer. Mark R57 where it is so that you can always return to the factory setting if you get frustrated. A little, let me repeat - a LITTLE - fiddlin should get you a setting that will reliabley read both 300 and 600 BAUD tapes.

On the transmit end, we just moved up the clock divide chain to the next higher step. By going to pin 12 of U30 we could have gotten 1200 baud, but OSI systems do not read BASIC reliably at that speed. We changed the capacitor that determined the frequency of the input clock (U69 and U63) so that it would match the new output frequency.



Some Real Products

A basic system can be built from a case (size 12x 17.22inches), motherboard, and one of two configurations.
1) 502, 542, 540, (527 optional)cassette system
2) 505, 542, 540, 527 floppy 8"

To convert the cassette system into an 8" Floppy-505 system you have to exchange the 502 board with a 505 board, or add a 470 diskcontroller board. The universal I / O board A15 is mounted in the cabinet of the C2 8P system.
The C2 8P is housed in a full size case with an eight slot motherboard. The Challenger C2-8P or C8 is a full size computer like the Challenger 3, but has only the 6502 processor instead of three processors of the Challenger 3 (6502, 6800, Z80).

502 CPU Board: Contains the 6502 CPU, System Monitor, 8K BASIC in ROM, 8K RAM, Cassette Interface (serial Interface)

542 keyboard : Keyboard polled, Audio Generator, Digital to Analog Converter, complete analog I/O portion for an audio cassette port so that by simply having an ACIA or UART somewhere in the system, you have a complete Kansas City audio cassette port.

New Features

Audio Output
The 542 Rev B board contains a programmable devider audio output. By specifying the contents of one register, the user can select the pitch of a continuous tone. This tone will be output automatically by the computer without any additional program intervention until changed. One location specifies the frequency and one bit of another location turns the audio output on and off.
The following pages specify the operation of the audio output and include a sample program.

USE OF 542 REV B AUDIO OUTPUT

The 542 Rev B keyboard has a programmable divider i used to generate audio tones. The output frequency determined by the equation:

$$\text{Frequency Out} = \frac{49152}{I}$$

where I is any integer between 1 and 255.

On power up of the computer, the tone is disabled. Setting bit 1 of location $DE\emptyset\emptyset_{16}$, (56832_{10}), will enable the output and clearing bit 1 of that location will disable it. To enable the output simply:

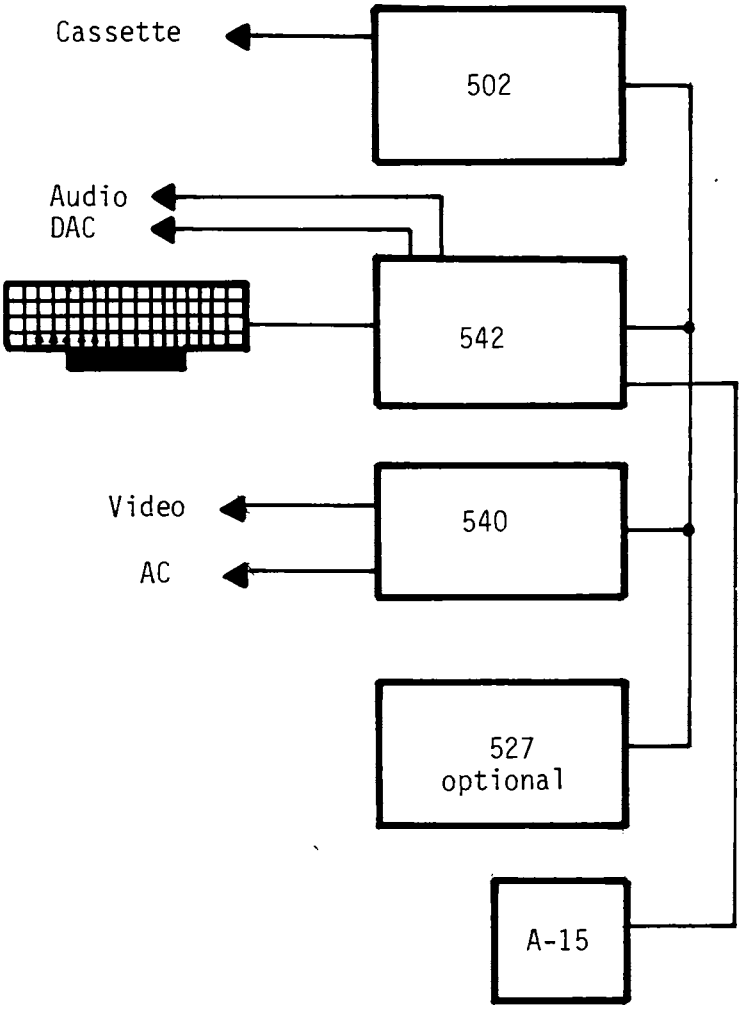
```
POKE 56832,3
```

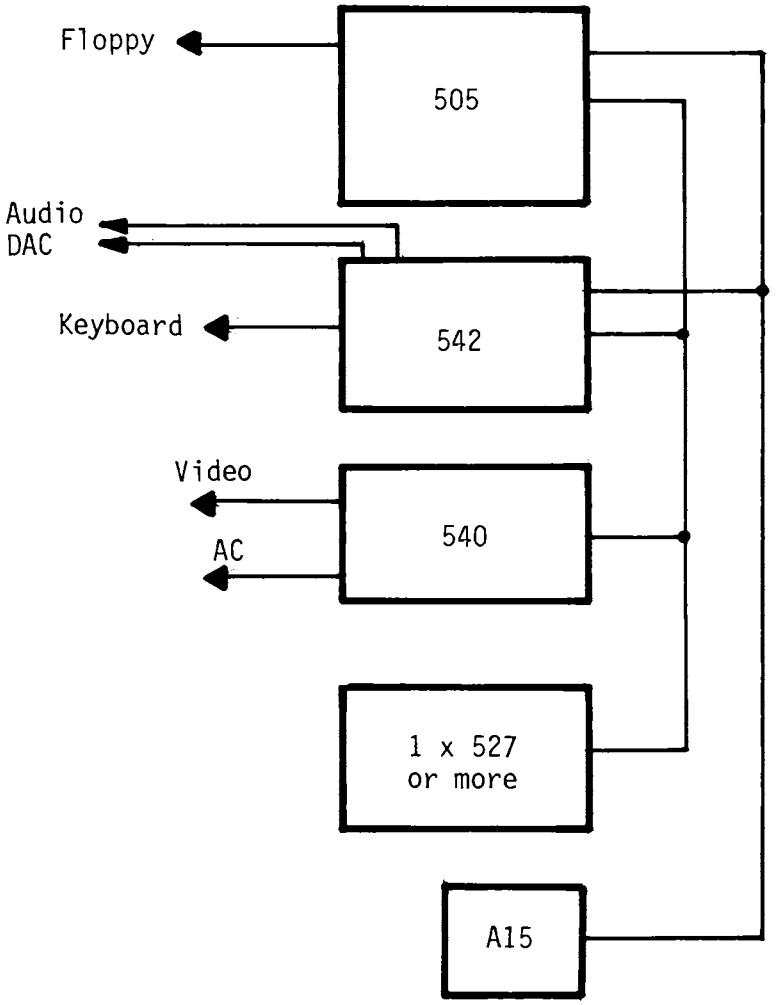
At this time whatever value was poked into location 57089 previously will determine the output freque cy. To disable output: POKE 56832,1

When enabling or disabling the sound output, care must be taken as bit Ø of this register also switch the display mode. The following values poked into l tion 56832 have the listed effects on the computer:

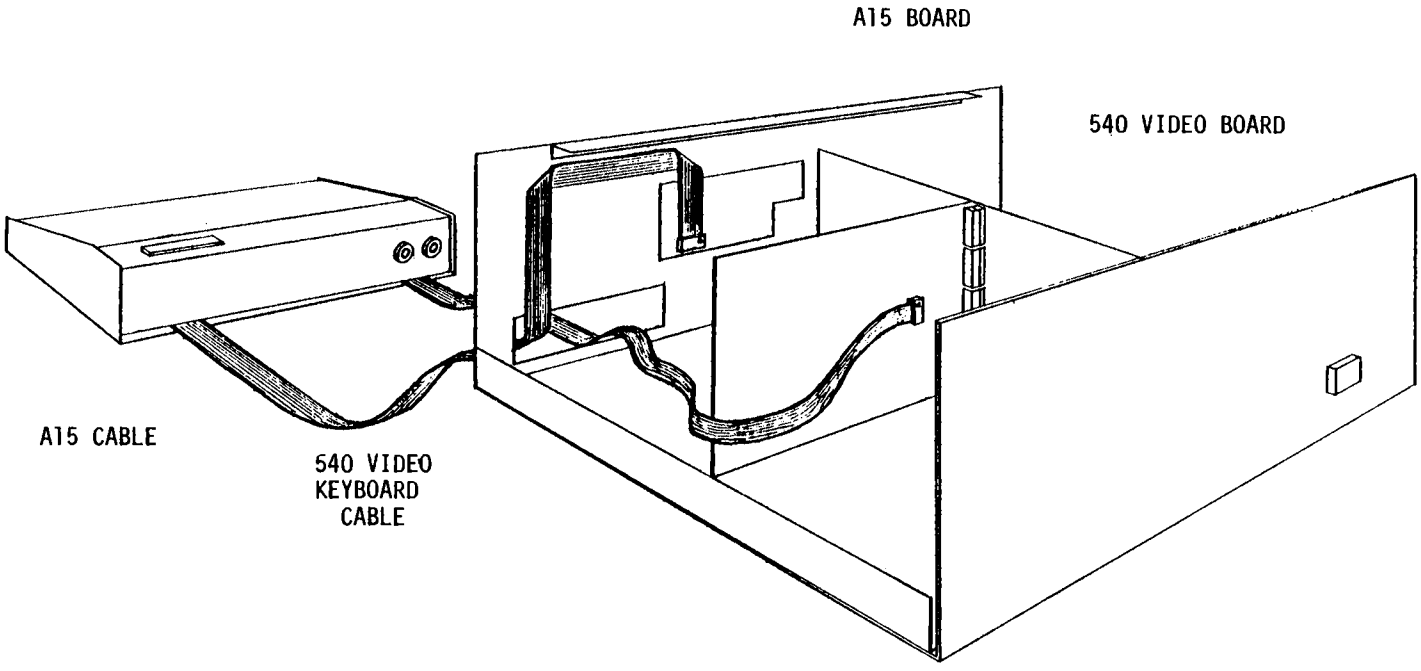
Ø	32 character mode and no sound
1	64 character mode and no sound
2	32 character mode and sound
3	64 character mode and sound

Location 57089 as mentioned, determines the outpu frequency using the equation stated previously. Bot of these locations are write only and cannot be PEE It is recommended that sound be disabled currently dividing by 1, that is, POKE 57089,1 which outputs 49152 Hz and is not audible.


Cassette
502
Audio
DAC
542
Video
AC
540
527
optional
A-15


Floppy
505
Audio
DAC
Keyboard
542
Video
AC
540
1 x 527
or more
A15

CABLE PLACEMENT



Sample Audio Output Program

```
 9 REM SET OUTPUT = 49152
10 POKE 57089,1
19 REM ENABLES OUTPUT
20 POKE 56832,3
30 B=Y : E=255:D=10
39 REM GENERATE RAMP
40 FOR I = B TO E
49 REM POKE DIVISOR
50 POKE 57089,I
59 REM DELAY
60 FOR X = 1 TO D
70 NEXT X
80 NEXT I
90 GOTO 40
```

Connection of Audio Output

The audio output from the computer is compatible wi the auxiliary input of normal audio amplifier equip ment. It can be directly connected to the audio in- put jack on a standard AC-3P video monitor if desir

Software

Ohio Scientific has developed and is currently de- veloping several programs on cassette and diskette which incorporate audio output. Projects include se veral games which incorporate audio sound effects a a musical composition system called "Music Box" whi allows persons to compose musical pieces which the stem then plays.

D/A Converter

The Mod 3 computer system also includes an 8 bit co panding D/A converter for advanced sound and voice put experiments. The companding D/A converter is ca ble of producing elaborate voice and musical output comparable in sound quality to a typical AM radio. However, to accomplish this feat, the D/A converter must have the constant attention of the microproces sor. This is typically accomplished by the use of

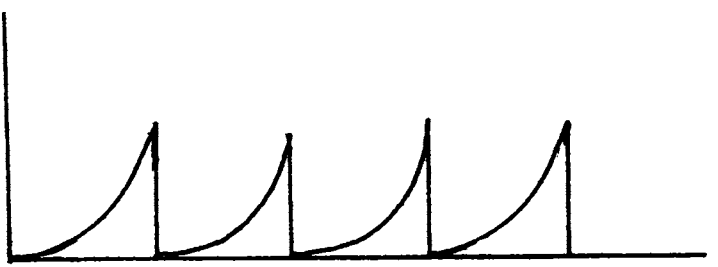
machine code or by assembler code subroutines interfaced by the USER function of BASIC. Thus, while the computer system is generating complex audio outputs by the D/A converter, it cannot be doing other tasks. The following page includes a simple program in machine code to generate a ramp, the simplest form of audio DAC programming. Ohio Scientific is working on software which will allow limited voice output and complex musical chord generation primarily on mini-floppy disks.

DAC Programming Example

Machine code routine which can be entered via the C1P monitor. (All numbers are in HEX)

	Location	Contents	Comments
Start	0300	E8	Increment X
	0301	8E	Store X at
	0302	01	Location DF01
	0303	DF	
	0304	4C	Jump back
	0305	00	to start
	0306	03	

This program continually increments the value fed to the DAC by 1 from o to 255 (decimal) over and over, producing the saw-tooth waveform below. The ramp of the saw-tooth is not straight because the DAC has approximately a log response for improved audio dynamix range. Changing the increment X to a decrement X would reverse the pattern etc.



D/A Converter Connections

The D/A converter is compatible with the auxiliary input of audio amplifiers and should be connected singularly to an audio input. That is, it should not be tied together with the other audio output of the computer.

Note: The D/A converter shares the same register assignment with the other audio output so that the D/A converter should not be used simultaneously with the other audio output.

AC Control and Real Time Clock

The connector in the lower right hand corner of the Mod 3 computer is the AC control interface connector. This connector mates with the connector on the side of the AC-12 remote control console. All software and documentation for this option accompanies the AC-12 remote control accessory package. On mini-disk systems a real time clock is present on the 505 CPU board. However, real time clock utilization is complex and requires a special real time support operating system. The support software for real time operation and time of day maintenance is incorporated as part of the AC-12 remote control accessory package.
Note: Support of the real time clock requires extensive modification to standard OS-65D V3.0 software. Ohio Scientific cannot assist in the conversion of this software package. If you are interested in utilizing a real time clock, we recommend the real time operating system which accompanies the AC-12 remote control package.

Color

The color option is an additional option above and beyond those specified here. Please refer to the accompanying color hook-up and operating instructions, if your computer system is equipped with color.

Rev.B = Color

540 Board Video Board: (CA-11)

The model 540 is used in all standard products. There is a black and white and a color version. The 540 board is a highly sophisticated video display device, which is capable of displaying up to 32 rows of 64 characters symbols, graphics elements and gaming elements. The board supports a conventional ASCII keyboard or a polled keyboard.
The display can be programmed for 32 x 32 or 32 x64 character display capability, a unique feature in the microcomputer world. The processor can access the screen memory by addressing the board in the range $D000-$OFFF.
Additionally the 540 board contains a separate keyboard input port, which can be configured either to accept the standard 7 line input ASCII keyboard or it can be configuered as a bidirectional 8 line keyboard port for the 542 board.
The 540 board uses an extremely high dot clock frequency of approximately 12 MHz. This was done to allow for approximately a 30% overscan guard band on both sides of the picture such that all 64 characters can be seen on normal television sets with a normal amount of overscan without requiring any adjustment of the horizontal width of the video monitor.

The 540 board can utilize a standard 2513N character generator ROM to provide 64 upper case ASCII characters in conjunction with a 2K by 6 bit memory. It is also set up to accept Ohio Scientific proprietary 256 character generator ROM, part No. CG-4.
All new boards now incorporate the CG-4 character generator. It contains 256 numeric, graphics and gaming elements including all 64 upper case characters and letters, all lower case alpha, a full set of graphics and plotting characters and a full set of gaming elements including race cars, airplanes, tanks, houses, trees and even the starship Enterprise.

Another added feature of the 540 for real hardware buffs is that the character generator ROM has the same

pinout as the 2716 EPROM so that the ambition programmer could program in his own fonts, if desired.
The keyboard No. 542 can be easily connected to the 540 board. The monitor PROMs in the computer have to be adapted to work with the hardware configuration:

SUPPORT EPROM CONFIGURATIONS

540 ROM BASIC WITH ASCII KEYBOARDS

Monitor	65V2P	at $	FE00
Support	65VB 7,2	at $	FF00

540 Disc based with ASCII keyboard

Monitor	65V2P	at $	FE00
Support	500-F3	at $	FF00

540 ROM BASIC with polled keyboard

Monitor	65VK	at $	FE00
Support	65VB7.6	at $	FF00
Support	65K	at $	F000

540 Disc based with polled keyboard

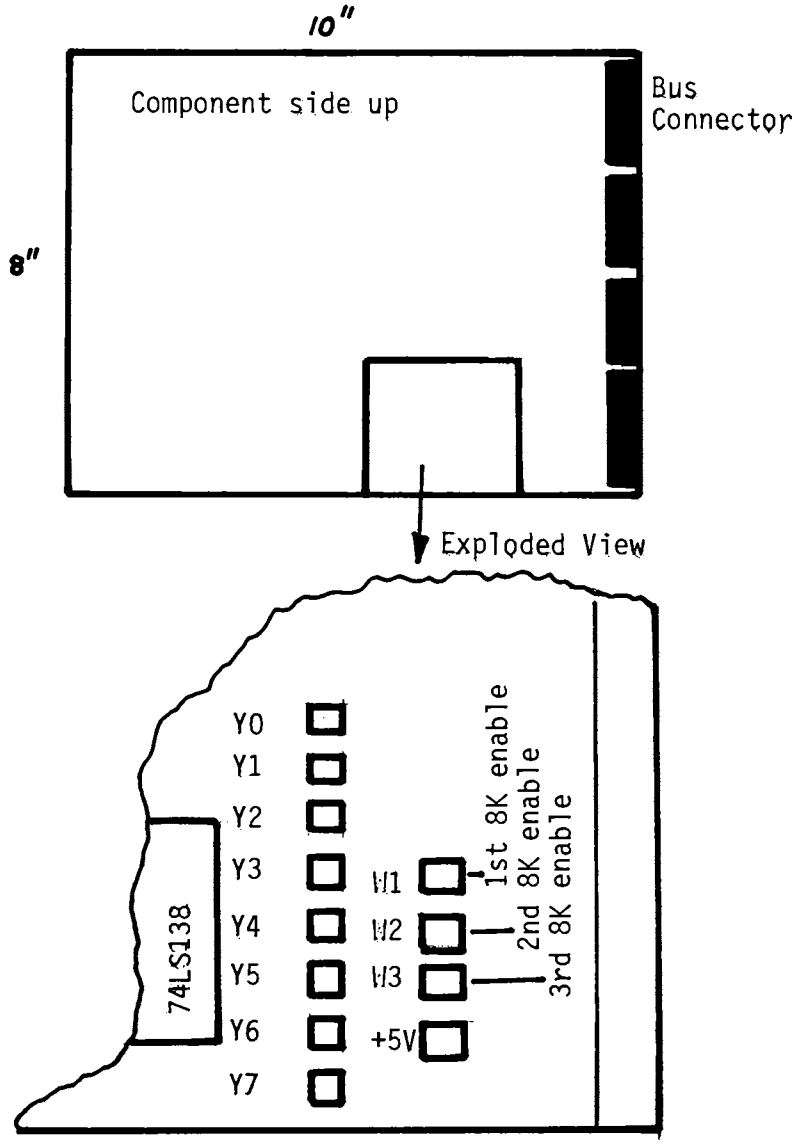
Monitor	65VK	at $	FE00
Support	500-F3	at $	FF00
Support	65K	at $	FD00

In addition, if the polled keyboard is used, polled keyboard support PROM at $ FDXX must be added.

527 Static RAM Board (CM-9)

For our configuration 1 (cassette system) you don't need a 327 RAM board, because you have 4K RAM on the 502 board and you can expand it to 8K on board very easily. If you cant to go to the disk system you need the 527 RAM board and you also have to exchange the 502 board with a 505 board.

The 527 board may be populated as an 8K, 16K or 24K RAM board. It uses the industry standard 2114 static RAM chip.


10"
Component side up
Bus
Connector
8"
Exploded View
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
74LS138
W1
W2
W3
+5V
1st 8K enable
2nd 8K enable
3rd 8K enable

Strapping for 527 (CM-7)
NOTE: The 527 maybe populated as an 8K, 16K, or 24K RAM board.

Strapping for 8K 527 board (CM-7)

* indicates jumper between row and column. For 1st 8K slot, 2nd 8K slot, etc.
X indicates don't care

		W1	W2	W3
$0000	Y0	1	X	X
$2000	Y1	2	X	X
$4000	Y2	3	X	X
$6000	Y3	4	X	X
$8000	Y4	5	X	X
$A0000	Y5	6	X	X
$C000	Y6	7	X	X
$E000	Y7	8	X	X

e.g. to Jumper for 2nd 8K slot

Jumper W1 to Y1

W2-don't care

W3 - don't care

If you are using 8K on the CPU-Board you have to move the first block to the second 8K-row.

Strapping for 16K 527 (CM-8)

// Indicates Jumper between row and column. For 1st 16K slot, 2nd 16K slot, etc.

X - indicates don't care

		W1	W2	W3
$0000	Y0	1		X
$2000	Y1		1	X
$4000	Y2	2		X
$6000	Y3		2	X
$8000	Y4	3		X
$A000	Y5		3	X
$C000	Y6			X
$E000	Y7			X

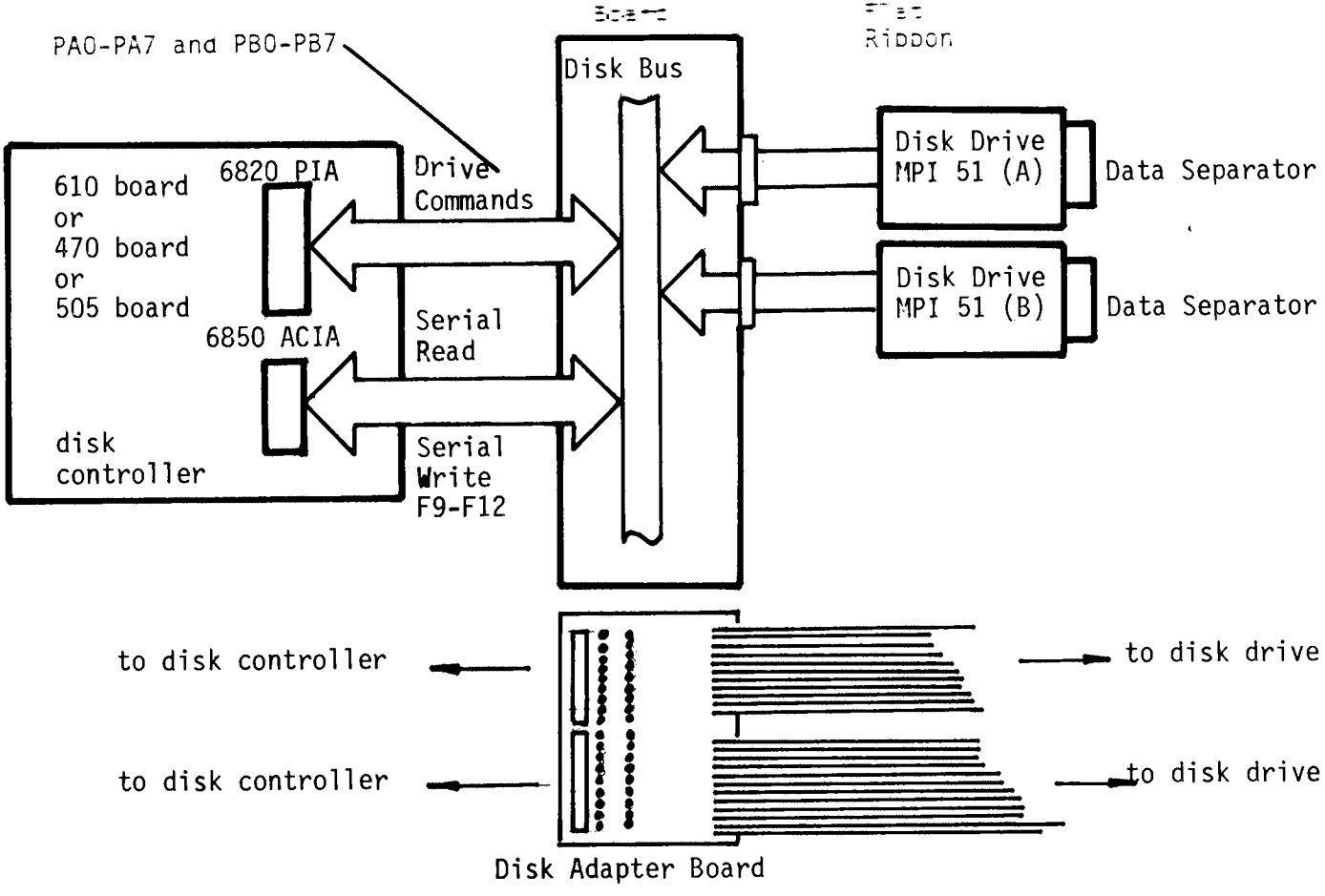
Strapping for 24K 527 (CM-9)

// indicates jumper between row and column. For 1st 24K slot, or 2nd 24K slot.

		W1	W2	W3
$0000	Y0	1		
$2000	Y1		1	
$4000	Y2			1
$6000	Y3	2		
$8000			2	
$A000				2
$C000	Y6			
$E000	Y7			

The OSI 48 Line BUS

• Challenger II CPU BASIC-in-ROM 6502 based CPU with serial I/O 4K RAM, machine code monitor	• Can use four 2716 EPROMS instead of BASIC or can be configured for disk	C2-0	500	+ 5/ – 9
• Challenger III CPU has 6502A, 6800 and Z80 micros, RS-232 serial port, machine code monitor	• 1 megabyte memory manager, software programmable vectors	C3-0	510	+ 5/ – 9
• 560Z multi-processing CPU expander runs PDP-8, Z80 and 8080 code	• Runs concurrently with another OSI CPU	NA	560Z	+ 5/ – 9
• 16K static RAM (Ultra low power)	• 215NS access time automatic power down standby mode	CM-3	520	+ 5/ + 12/ – 9
• 8K static RAM (low cost)	• Expandable to 16K	CM-7	—	+ 5
• 16K static RAM (low cost)	• Can be expanded to dual port operation	CM-8	525	+ 5
• 24K static RAM (high density)	• 20 address bits	CM-9	527	+ 5
• 4K static RAM (2102 based)	• Can be populated for 4K by 12 bits	CM-2	420	+ 5
• 16K dynamic (ultra low cost)	• Uses 4027 RAMS	CM-4	530	+ 5/ + 12/ – 9
• 32K dynamic	• 20 address bits	CM-5	530	+ 5/ + 12/ – 9
• 48K dynamic (high density)	• 20 address bits	CM-6	530	+ 5/ + 12/ – 9
• 8K 6834 EPROM board	• 16 line parallel port and on board programmer	NA	450	+ 5/ – 9
• 4K 1702A EPROM board	• 16 line parallel port	NA	455	+ 5/ – 9
• Audio Cassette interface Kansas City standard 300 baud	• Expandable to CA-7C	CA-6C	430	+ 5/ – 9
• RS-232 port board	• Expandable to CA-7S	CA-6S	430	+ 5/ – 9
• Combination audio cassette two 8 bit DACs, one fast A/D and 8 channel input mux	• Also Features 8 parallel I/O lines	CA-7C	430	+ 5/ – 9
• Combination RS-232 two 8 bit DACs, one fast A/D and 8 channel input mux	• Also features 8 parallel I/O lines	CA-7S	430	+ 5/ – 9
• 32 by 32 character video display interface	• Keyboard input port	NA	440	+ 5/ – 9
• 32 by 64 character video display interface	• Upper/lower case graphics and keyboard port	CA-11	540	+ 5
• 16 port serial board RS-232 and/or high speed synchronous	• 75 to 19,200 baud and 250K and 500K bit rates individually strappable	CA-10X	550	+ 5/ – 9
• Parallel (Centronics) Line Printer Interface	• With cable	CA-9	470	+ 5/ – 9
• 96 Line Remote Parallel Interface	• Interface "Front End" remotable via 16 pin ribbon cable	CA-12	—	+ 5
• Voice I/O board with Votrax* module	• Fully assembled voice output, experimental voice input	CA-14	—	+ 5/ – 9
• Single 8" floppy disk, 250 Kbytes storage	• Complete with operating system software and disk BASIC	CD-1P	470	+ 5/ – 9
• Dual 8" floppy disk, 500 Kbytes storage	• Complete with operating system software and disk BASIC	CD-2P	470	+ 5/ – 9
• 74 Million byte Winchester disk and interface	• Complete with OS-65U operating system	CD-74	—	+ 5/ – 9
• 8 slot backplane board with connectors	• Can be daisy-chained to n-slots	NA	580	—
• Prototyping board	• Handles over 40 16 pin IC's	—	495	—
• Card Extender	• With connectors	—	498	—


PA0-PA7 and PB0-PB7
Disk Bus
Ribbon
610 board
or
470 board
or
505 board
6820 PIA
6850 ACIA
Drive
Commands
Serial
Read
Serial
Write
F9-F12
disk
controller
Disk Drive
MPI 51 (A)
Data Separator
Disk Drive
MPI 51 (B)
Data Separator
to disk controller
to disk controller
to disk drive
to disk drive
Disk Adapter Board

The CM-9 or 527 board is a 24K, 2 MHz medium power memory board. It is usable in computers with booster supplies or high current switchers. (20 address bits)

530 dynamic RAM board (48K, 1 MHz (CM-6)

This high density dynamic RAM board can be used in C2-OEM and some timeshare systems. The board uses 4072 RAM chips.

470 Floppy disk controller board (CA-9)

The 470 board is a remarkably simple yet extremely flexible and reliable diskette interface. The 470 design philosophy is that all possible disk control functions are performed via software, not hardware. This yields an extremely intelligent but very low cost system. It mates with the 400 series backplane. The controller could be used with any drive having separated data and separated clock outputs but the software would not necessarily be compatible.
The 470 also includes a real time clock, derived from the crystal controlled write circuit.

Table 1. Floppy Disk Connections

470 Board	PIA Assignment	Signal Name
F_1	PB7	Head Load
F_2	PB6	Low Current
F_3	PB5	Select Drive 1
F_4	PB4	Fault Reset
F_5	PB3	Step
F_6	PB2	Step In
F_7	PB1	Erase Enable
F_8	PB0	Write Enable
F_9		Write Data
F_{10}		Separated Clock
F_{11}		Separated Data
F_{12}		Ground
F_{13}		Ground
F_{14}		+5
F_{15}		-5
F_{16}		+24
F_{17}	PA7	Index

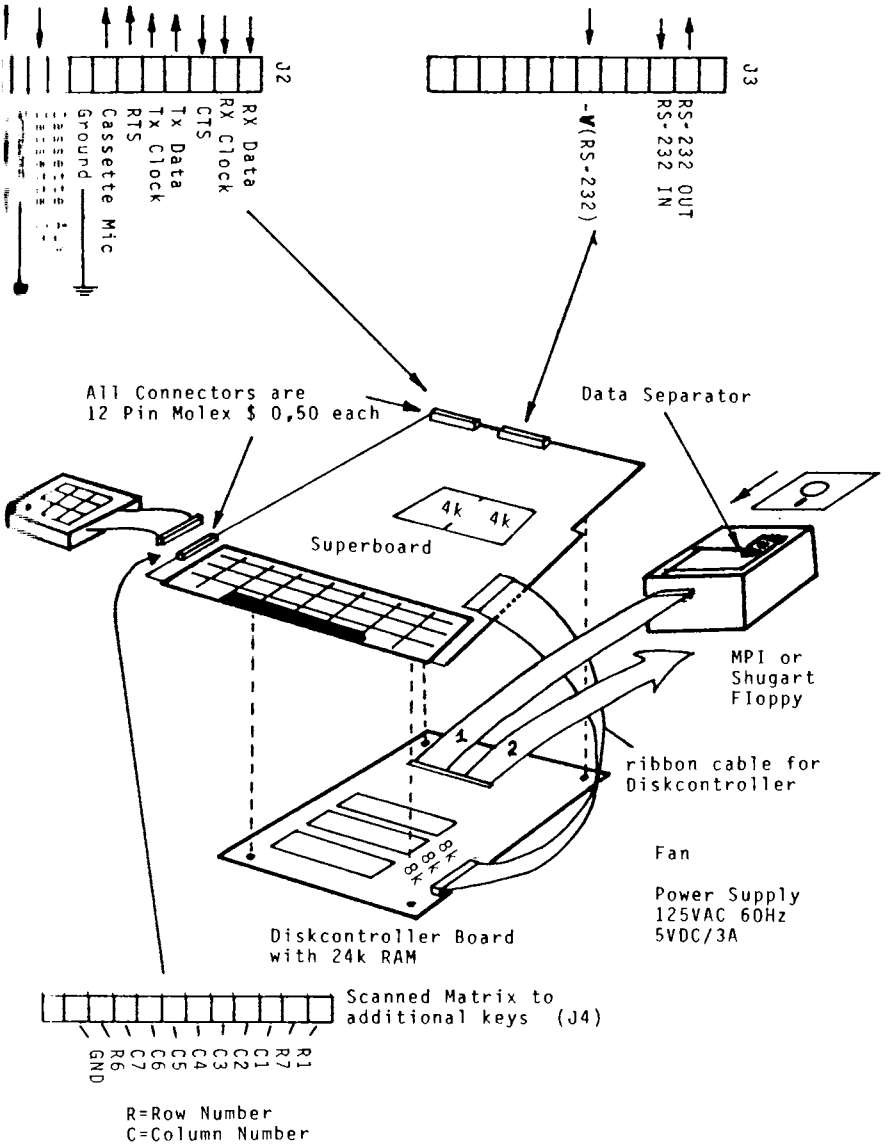
I/O Board	PIA Assignment	Signal Name
F_{18}	PA6	Select Drive 2
F_{19}	PA5	Write Protect (optional)
F_{20}	PA4	Ready (Drive 2)
F_{21}	PA3	Sector (optional)
F_{22}	PA2	Fault
F_{23}	PA1	Track 00
F_{24}	PA0	Ready (Drive 1)

twisted pair grounds should be terminated at the disk and the G connectors at the 470 board. Power connections can alternately be directly to the disk drives. When two drives are used, all lines except for Ready and Select are simply fed to both drives!

The OSI-48 pin-Bus

Pin No.	Description
A1	WAIT. When pulled low by a system board, causes processor clock to slow down to speed of approximately 500 kHz. Used to service slow memory and I/O devices.
A2	NMI (non-maskable interrupt). When brought low, a non-blockable interrupt occurs, causing the processor to stop its operation and service this interrupt, that is, go to a specific memory location and execute an interrupt service routine.
A3	IRQ (interrupt request). An interrupt that can be masked by the processor. The processor can choose to ignore this interrupt under program control. If the interrupt is not masked, it will function as NMI above.
A4	DD (data direction). When pulled low by a system board, it changes the data direction of the 8T26 buffers on the CPU board, switching the processor from outputting data to the bus to listening to the bus

Pin No.	Description
B5-B12	Bidirectional, eight-bit data bus for communication between the processor and system boards.
B13-B16	Upper data bits on some systems.
B17	Optional reset line used to clear all PIAs and similar I/O circuitry in the system.
B18	Spare
B19-B22	Memory management address lines (the OSI system can address memory in 64K blocks up to at least 768K).
B23	+ 12 volt power connection.
B24	=9 volt power connection.
B25-B26	+ 5 volt power connection.
B27-B29	Ground.
B29-B38	Ten low-order address lines.
B39	Ø2. Used to clock external circuits or external I/O interfaces, such as the A/D converter (see a 6502 data sheet for more details).
B40	R/W (read/write). Originates at the microprocessor and specifies read or write operations on the data bus.
B41	VMA (valid memory address). Only used in conjunction with the 6800. The 6502 always has this line high.
B42	Ø2-VMA. Master timing signal for enabling memory and I/O in the system.
B43-B48	Six high-order address lines.


J2
RX Data
RX Clock
CTS
Tx Data
Tx Clock
RTS
Cassette Mic
Ground
J3
RS-232 OUT
RS-232 IN
-V(RS-232)
All Connectors are
12 Pin Molex $ 0,50 each
Data Separator
4k 4k
Superboard
MPI or
Shugart
Floppy
1
2
ribbon cable for
Diskcontroller
8k
8k
8k
Fan
Power Supply
125VAC 60Hz
5VDC/3A
Diskcontroller Board
with 24k RAM
Scanned Matrix to
additional keys (J4)
R1
R7
C1
C2
C3
C4
C5
C6
C7
R6
GND
R=Row Number
C=Column Number

OHIO SCIENTIFIC C1-P MINI-FLOPPY EXPANSION ACCESSORIES

No.	Item
0	Challenger 1P with manuals, Demo program tape, Available with 4K or no memory.
1	Model 600 Superboard with manuals, Demo tape, Available with 4K or no memory
2	Model 610 Mini Floppy Expansion Board (expandable to 24K), Available with 8K or no memory.
3	Model 620 C1-P to OSI 48-pin C3 Bus Converter (optional).
4	5V 3A Power Supply (for either 600 or 610 Board), Fits cabinet, item 10 directly.
5	MPI 40-track Disk Drive in Metal Cabinet, with or without power supply (12V).
5a	MPI 40 - track Disk Drive (no cabinet), Includes data separator
6	Cable Adapter, 610 board to disk drive. Includes interface schematic
7	Cable Adapter, 600 boad to 610 board
8	4K Memory (8-2114's)
8a	8K Memory (16-2114's)
9	Sprite Fan (Needed for 12K and up)
10	Challenger 1P Cabinet w. power cord and fuse

ELPACK Data Separator for MPI Model

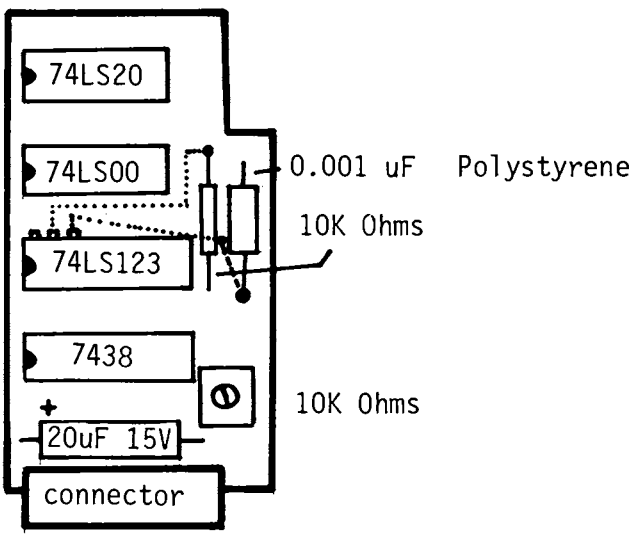
Component Location diagram

The data separator board is installed at connector J5 on MPI 51 circuit board.

The connector is a 10 pin Molex connector. The purpose of the module is to provide separate Read-Data and Read Clock.

It is required in OSI Systems. The data is recorded in the FM Mode.

TOP VIEW (Component Side)



A CRUDE, (BUT PRE-PREPARED) MEMORY TEST ROUTINE
WRITTEN IN ASSEMBLY LANGUAGE.
IT USES OS65D'S STRING OUTPUT HANDLER AND ASSEMBLER
AND WILL TEST ANY MEMORY ABOVE $3600 AT RATHER HIGH SPEED.
IT TESTS EVERY PATTERN IN EVERY BYTE, AND OUTPURS THE ERRORS.
IT MAY DRIVE A PRINTER USING OS65D'S 'IO 02,03 PROVISION.
AND TO SAVE EXCESSIVE PRINTING, YOU MAY SELECT (AT ASSEMBLY TI
THAT UPON HITTING AN ERROR IN A 256 BYTE PAGE, IT SKIPS OVER
THE REST OF THE PAGE. THIS ENABLES YOU TO LOCATE BAD OR SLOW
CHIPS WITHOUT USING WHOLE REAMS OF PAPER.
MULTIPLE LOOPS ARE ALSO PROVIDED FOR, AND RECOMMENDED.

FULL ASSEMBLY SOURCE LISTINGS, DISSASSEMBLED LISTINGS,
AND SAMPLE RUNS ARE INCLUDED.
THIS IS A QUICK AND DIRTY ROUTINE.

OK

```
HERE ARE THE INSTRUCTIONS FOR THE ASSEMBLY LANGUAGE
MEMORY TEST ROUTINE

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

1. BOOT SYSTEM AND EXIT, LOAD THE ASSEMBLER USING
   THE 'ASM' COMMAND TO OS65-D
2. TYPE '!LOAD MEMTST' TO LOAD SOURCE LISTING
3. TYPE '!IO 02,03' TO ENABLE THE SERIAL PORT
4. AT THIS POINT YOU MAY OBTAIN ASSEMBLY LISTING
   BY TYPING AN 'A', IF YOU DESIRE ONE
5. TYPE AN 'A3' TO ASSEMBLE PROGRAM
6. TYPE  '!GO 3410' TO EXECUTE PROGRAM
7. THE PROGRAM TERMINATES AT OS65D, SO YOU MAY
   RE-RUN IT BY TYPING SIMPLY 'GO 3410'
8. NOTE THAT THE ASSEMBLY PROCESS DESTROYS THE
   SOURCE CODE, SO TO REASSEMBLE YOU MUST RE-LOAD
   THE SOURCE FILE CALLED 'MEMTST'.
9. THE ASSEMBLY LISTING OF STEP 4 IS FULLY COMMENTED
WANT MEMORY MAP OF INPUT PARAMETERS? Y




  ------------------------------------------------
INSTALL NUMBERS USING YOUR EXTENDED MONITOR, OR
MODIFY THE SOURCE FILE USING THE ASSEMBLER'S EDITOR

$341F = first PAGE to test in memory
        must be ) 36 to avoid over-writing OS65D
        and the memory test program itself!!
$3427 = one greater than the last page to check

 SAMPLE VALUES:
FOR TEST FROM LOWEST ALLOWED UP TO....
........32K SET $341F=$36 (AS IS)
            SET $3427=$80

........40K SET $341F=$36 (AS IS)
            SET $3427=$A0 (AS IS)

........48K SET $341F=$36 (AS IS)
            SET $3427=$C0
PROGRAM PRESENTLY SKIPS REMAINDER OF PAGE UPON
FINDING AN ERROR.  SEE YOUR ASSEMBLY LISTING FOR
MODIFICATION LOCATIONS.  ALSO NOTE THAT IT IS
TO LOOP PROGRAM INDEFINITLY AS NOTED IN THE 'DONE'  ROUTINE

A*ASM
```

```
FULL ASSEMBLY LISTING----
.A

                        ;*********************************************
                        ;ASSEMBLY LANGUAGE MEMORY TEST ROUTINE
                        ;
                        ;(USES OS65D'S OUTPUT HANDLERS, SO DONT MESS
                        ; WITH OS65D)
                        ;
                        ;*********************************************
                        ;
                        ; **PRE-SET TO TEST FROM $3600 TO $9FFF**
                        ;
                        ;EXTERNAL LABELS
                        ;
2D6A=                     CRLF = $2D6A OS65 ROUTINE
2A51=                   OS65D3 = $2A51 OS65 COMMAND LOOP ENTRY
2D73=                   STROUT = $2D73 OS65 ROUTINE
                        ;
00EE=                   TADRLO = $00EE ZERO PAGE POINTER
00EF=                   TADRHI = $00EF ZERO PAGE POINTER
340F=                   MAXPAG = $340F U SET TO 1+MAXPAG TO TEST
340E=                     NBIT = $340E CURRENT TEST VALUE
340D=                     TEMP = $340D SCRATCH
340C=                   TEMPTO = $340C SAVE PAGE ZERO FOR RESTORE
340B=                   TEMPTL = $340B SAME AS ABOVE
                        ;
3410                     * = $3410              SET ORIGIN
                        ;
                        ;                     INITIALIZATIONS OF POINTER
                        ;
3410 18                   CLC
3411 D8                 START  CLD
3412 B8                   CLV
3413 58                   CLI
                        ;
3414 A5EF               BEGIN  LDA TADRHI
3416 8D0C34                    STA TEMPTO   PRESERVE PAGE ZERO
3419 A5EE                      LDA TADRLO
341B 8D0B34                    STA TEMPTL
                        ;
                        ;
                        ;                   PUT FIRST PAGE AS OPERAND I
                        ;                   NEXT INSTRUCTION
                        ;*********************************************
341E A936                      LDA #$36
3420 85EF                      STA TADRHI START PAGE
3422 A900                      LDA #$00
3424 85EE                      STA TADRLO
                        ;
                        ;                  PUT LAST PAGE AS OPERAND IN
                        ;                  NEXT INSTRUCTION
                        ;*********************************************
3426 A9A0                      LDA #$A0
3428 8D0F34                    STA MAXPAG
                        ;
342B A000                      LDY #$00 INIT Y
                        ;
342D A200                      LDX #$00
```

```
570 342F 8E0E34          STX NBIT
580                      ;
590 3432 AD0E34 STLOOP   LDA NBIT
600 3435 91EE            STA (TADRLO),Y
610 3437 B1EE            LDA (TADRLO),Y GET RETURN DATA
620 3439 CD0E34          CMP NBIT               ERROR???
630 343C D02B            BNE ERROR BRANCH WITH BAD IN ACC.
640                      ;                              GOOD IN NBIT
650 343E EE0E34 CONT     INC NBIT
660 3441 F003            BEQ UPADDR  INCR ADDR IF DATA LOOP DONE
670 3443 4C3234          JMP STLOOP
680                      ;                 INCREMENT ADDRESS
690                      ;
700 3446 C8      UPADDR  INY
710 3447 F003            BEQ PAGE  IF Y'S=0 THEN INCR PAGE
720 3449 4C3234          JMP STLOOP RTN TO TEST IF NO PAGE
730                      ;           INCR PAGE IF NESS.
740                      ;
750 344C A6EF    PAGE    LDX TADRHI
760 344E E8              INX
770 344F EC0F34          CPX MAXPAG
780 3452 9010            BCC KLUGE
790                      ;----------------------EXITS HERE
800 3454 AD0C34 DONE     LDA TEMPTO  RESTORE PAGE ZERO
810 3457 85EF            STA TADRHI
820 3459 AD0B34          LDA TEMPTL
830 345C 85EE            STA TADRLO
840 345E 206A2D          JSR CRLF INSERT JMP BEGIN AFTER HERE IF
850                      ;                YOU WISH REPETITION OF TEST
860 3461 4C512A          JMP OS65D3
870                      ;
880 3464 86EF    KLUGE   STX TADRHI
890 3466 4C3234          JMP STLOOP IFPAGE WITHINLIMIT THEN CONT
900                      ;
910                      ;--------------------ERROR PRINT ROUTINE
920                      ;
930 3469 A204    ERROR   LDX #$04 USE X AS STRING POSIT POINTER
940 346B 8D0D34          STA TEMP PRESERVE BAD DATA PATTERN
950 346E AD0E34          LDA NBIT
960 3471 29F0            AND #$F0  SELECT 4 MSB'S
970 3473 200B35          JSR SHIFT
980 3476 20FF34          JSR ASC  GOTO ASCII CONVERTER
990 3479 9DD834          STA PUNT,X PUT ASCII IN STRING AT PUNT
1000 347C E8             INX MOVE POINTER
1010 347D AD0E34         LDA NBIT
1020 3480 290F           AND #$0F SELECT 4 LSB'S
1030 3482 20FF34         JSR ASC
1040 3485 9DD834         STA PUNT,X
1050 3488 A20D           LDX #$0D MOVE POINTER
1060 348A AD0D34         LDA TEMP GET THE BAD DATA FROM TEMP
1070 348D 29F0           AND #$F0
1080 348F 200B35         JSR SHIFT
1090 3492 20FF34         JSR ASC
1100 3495 9DD834         STA PUNT,X
1110 3498 E8             INX
1120 3499 AD0D34         LDA TEMP
1130 349C 290F           AND #$0F
1140 349E 20FF34         JSR ASC
```

```
1150 34A1 9DD834          STA PUNT,X
1160 34A4 A214            LDX #$14
1170 34A6 A5EF            LDA TADRHI GET ERROR PAGE #
1180 34A8 29F0            AND #$F0
1190 34AA 200B35          JSR SHIFT
1200 34AD 20FF34          JSR ASC
1210 34B0 9DD834          STA PUNT,X
1220 34B3 E8              INX
1230 34B4 A5EF            LDA TADRHI
1240 34B6 290F            AND #$0F
1250 34B8 20FF34          JSR ASC
1260 34BB 9DD834          STA PUNT,X
1270 34BE E8              INX
1280 34BF 98              TYA PUT ADLO OF BAD DATA IN ACCUM
1290 34C0 29F0            AND #$F0
1300 34C2 200B35          JSR SHIFT
1310 34C5 20FF34          JSR ASC
1320 34C8 9DD834          STA PUNT,X
1330 34CB E8              INX
1340 34CC 98              TYA
1350 34CD 290F            AND #$0F
1360 34CF 20FF34          JSR ASC
1370 34D2 9DD834          STA PUNT,X  STRING NOW COMPLETE
1380                ;-------------------------------------
1390                ;                   JUMP TO OS65 STRING OUTPUT
1400                ;
1410                ;
1420 34D5 8C0D34          STY TEMP PRESERVE Y FROM MODIFICATION
1430 34D8 20732D PUNT     JSR STROUT
1440 34DB 24              .BYTE '$   GAVE $   AT $    ',0
1440 34DC 20
1440 34DD 20
1440 34DE 20
1440 34DF 47
1440 34E0 41
1440 34E1 56
1440 34E2 45
1440 34E3 20
1440 34E4 24
1440 34E5 20
1440 34E6 20
1440 34E7 20
1440 34E8 41
1440 34E9 54
1440 34EA 20
1440 34EB 24
1440 34EC 20
1440 34ED 20
1440 34EE 20
1440 34EF 20
1440 34F0 00
1450 34F1 206A2D          JSR CRLF
1460 34F4 AC0D34          LDY TEMP RESTORE Y
1470 34F7 A000            LDY #$00 REM********MODIFIED TO GO TO
1480                ;                   NEW PAGE FOLLOWING ERROR
1490                ;                   DETECTION, DELETE THIS LINE
1500                ;                   TO RESTORE DETAILED CHECK
1510 34F9 4C4D34          JMP PAGE   ALSO MODIFIED AS ABOVE
```

```
1520 34FC 4C3E34        JMP CONT   EXIT BACK TO TEST
1530              ;-------------------------------------
1540              ;  MAKE ASCII CODE OF HEX VALUE IN ACCUM.
1550              ;
1560 34FF C90A    ASC     CMP #$0A  CHECK FOR DEC OR LETTER HEX
1570 3501 9004            BCC STAY BRANCH IF DEC NUMB
1580              ;
1590 3503 18              CLC
1600 3504 6937            ADC #$37 CONVERT TO ABCDE OR F
1610 3506 60              RTS RETURNS WITH ASCII IN ACCUM
1620 3507 18      STAY    CLC
1630 3508 6930            ADC #$30 CONVERT TO 0 - 9 ASCII
1640 350A 60              RTS RETURNS WITH ASCII IN ACCUM
1650 350B 4A      SHIFT   LSR A
1660 350C 4A              LSR A
1670 350D 4A              LSR A
1680 350E 4A              LSR A
1690 350F 60              RTS
1700                     .END


DISSASSEMBLY BY OS65D EXTENDED MONITOR.....




A3


.ESI_  _  _


EXIT
02 TRACK(S)
A*EM

EM V2.0
:03410


3410 18      CLC
3411 D8      CLD
3412 B8      CLV
```

```
3413 58      CLI
3414 A5EF    LDA $EF
3416 8D0C34  STA $340C
3419 A5EE    LDA $EE
341B 8D0B34  STA $340B
341E A936    LDA #$36
3420 85EF    STA $EF
3422 A900    LDA #$00
3424 85EE    STA $EE
3426 A9A0    LDA #$A0
3428 8D0F34  STA $340F
342B A000    LDY #$00
342D A200    LDX #$00
342F 8E0E34  STX $340E
3432 AD0E34  LDA $340E
3435 91EE    STA ($EE),Y
3437 B1EE    LDA ($EE),Y
3439 CD0E34  CMP $340E
343C D02B    BNE $3469
343E EE0E34  INC $340E
3441 F003    BEQ $3446
3443 4C3234  JMP $3432
3446 C8      INY
3447 F003    BEQ $344C
3449 4C3234  JMP $3432
344C A6EF    LDX $EF
344E E8      INX
344F EC0F34  CPX $340F
3452 9010    BCC $3464
3454 AD0C34  LDA $340C
3457 85EF    STA $EF
3459 AD0B34  LDA $340B
345C 85EE    STA $EE
345E 206A2D  JSR $2D6A
3461 4C512A  JMP $2A51
3464 86EF    STX $EF
3466 4C3234  JMP $3432
3469 A204    LDX #$04
346B 8D0D34  STA $340D
346E AD0E34  LDA $340E
3471 29F0    AND #$F0
3473 200B35  JSR $350B
3476 20FF34  JSR $34FF
3479 9DD834  STA $34D8,X
347C E8      INX
347D AD0E34  LDA $340E
3480 290F    AND #$0F
3482 20FF34  JSR $34FF
3485 9DD834  STA $34D8,X
3488 A20D    LDX #$0D
348A AD0D34  LDA $340D
348D 29F0    AND #$F0
348F 200B35  JSR $350B
3492 20FF34  JSR $34FF
3495 9DD834  STA $34D8,X
3498 E8      INX
3499 AD0D34  LDA $340D
349C 290F    AND #$0F
```

```
349E 20FF34 JSR $34FF
34A1 9DD834 STA $34D8,X
34A4 A214   LDX #$14
34A6 A5EF   LDA $EF
34A8 29F0   AND #$F0
34AA 200B35 JSR $350B
34AD 20FF34 JSR $34FF
34B0 9DD834 STA $34D8,X
34B3 E8     INX
34B4 A5EF   LDA $EF
34B6 290F   AND #$0F
34B8 20FF34 JSR $34FF
34BB 9DD834 STA $34D8,X
34BE E8     INX
34BF 98     TYA
34C0 29F0   AND #$F0
34C2 200B35 JSR $350B
34C5 20FF34 JSR $34FF
34C8 9DD834 STA $34D8,X
34CB E8     INX
34CC 98     TYA
34CD 290F   AND #$0F
34CF 20FF34 JSR $34FF
34D2 9DD834 STA $34D8,X
34D5 8C0D34 STY $340D
34D8 20732D JSR $2D73
34DB 2420   BIT $20
34DD 202047 JSR $4720
34E0 4156   EOR ($56,X)
34E2 4520   EOR $20
34E4 2420   BIT $20
34E6 202041 JSR $4120
34E9 54     ???
34EA 202420 JSR $2024
34ED 202020 JSR $2020
34F0 00     BRK
34F1 206A2D JSR $2D6A
34F4 AC0D34 LDY $340D
34F7 A000   LDY #$00
34F9 4C4C34 JMP $344C
34FC 4C3E34 JMP $343E
34FF C90A   CMP #$0A
3501 9004   BCC $3507
3503 18     CLC
3504 6937   ADC #$37
3506 60     RTS
3507 18     CLC
3508 6930   ADC #$30
350A 60     RTS
350B 4A     LSR A
350C 4A     LSR A
350D 4A     LSR A
350E 4A     LSR A
350F 60     RTS
G3410
$AA GAVE $8A AT $5433
$20 GAVE $00 AT $5533
$AD GAVE $8D AT $56B1
```

SAMPLE RUN, THESE ERRORS FROM SLOW MEMORY
CONTINUES.....

```
$ED GAVE $CD AT $57B1
$21 GAVE $00 AT $8010
$21 GAVE $01 AT $8100
$41 GAVE $01 AT $830A
$21 GAVE $01 AT $8811
$21 GAVE $01 AT $8900
$60 GAVE $62 AT $8A29
$41 GAVE $01 AT $8E4B
$20 GAVE $00 AT $9C71
$20 GAVE $00 AT $9D63
$40 GAVE $00 AT $9E6E
$21 GAVE $01 AT $9F61


A*

A*IO 02,03

EM

EM V2.0
:@342_  10


3410 18      CLC
3411 D8      CLD
3412 B8      CLV
3413 58      CLI
3414 A5EF    LDA $EF
3416 8D0C34 STA $340C
3419 A5EE    LDA $EE
341B 8D0B34 STA $340B
341E A936    LDA #$36
3420 85EF    STA $EF
3422 A900    LDA #$00
3424 85EE    STA $EE
3426 A9A0    LDA #$A0
3428 8D0F34 STA $340F
342B A000    LDY #$00
342D A200    LDX #$00
342F 8E0E34 STX $340E
3432 AD0E34 LDA $340E
3435 91EE    STA ($EE),Y
3437 B1EE    LDA ($EE),Y
3439 CD0E34 CMP $340E
343C D02B    BNE $3469

:@341F

341F/36 D0
:@3427                CHANGE TO TEST FROM $D000 TO $E1FF

3427/A0 E2
:G3410
$00 GAVE $FE AT $D800
$00 GAVE $FE AT $D900
$00 GAVE $FE AT $DA00
$00 GAVE $FE AT $DB00
```

```
GAVE $FE AT $DC00
GAVE $FE AT $DD00
GAVE $FE AT $DE00
GAVE $FE AT $DF00
GAVE $00 AT $E000
GAVE $00 AT $E100

A>

WITI REGARDING LAST TEST--SYSTEM HAS NO MEMORY FROM $D800 TO $FFFF
```

OSI 65V Monitor MOD 2

The 65V Superbug Video Monitor resides in a 1702A occupying locations FE00 to FEFF. It requires 256 bytes of RAM at location 0100 up and 5 bytes at the top of Page 0 (FB to FF) and a 440 Video Board at DXXX with the keyboard at DF01. The PROM contains the IRQ, NMI, and RST vectors.

IRQ = 01C0
NMI = 0130
RESET = FE00

The program, when started, will set the stack pointer to 28, clear decimal mode, initialize a UART at BF00 (used for the Audio Cassette loader) and clear the memory space for the video board (i.e. fills with spaces).
The Monitor displays the following on the screen:

LLLL DO

i.e., location-space-space-Data. Its function is much like the keyboard/LED KIM-1 monitor. There are two different command modes, the Address Mode and the Data Mode. Each mode has commands specific to its function.

Commands

Address Mode Commands:
/ - Change to Data Mode
G - Go - - Jump to location seen on screen and execute program found there.
L - Transfer control to audio cassette. This means that the audio cassette supplies ASCII commands instead of the keyboard. This command enters data mode and ignores the keyboard and only listens to the audio cassette UART. To transfer control back to the keyboard, press reset or load 00FB with 00. 00FB is a flag. If 00FB = 00,

will then accept commands from keyboard. Otherwise, commands are accepted from Audio Cassette UART.

Data Mode Commands:
- Change to Address Mode
RETURN - Open next address. In other words, increment location pointer by 1.

If the 65V is in address mode, typing 0 - 9 or A - F will cause that number to be rotated into the LSD of the location pointer. Typing a 4 causes 0123 XX to become 1234 XX.

If it is in Data Mode, the number is rotated into the data contents and memory is thus modified. This permits the easy correction of errors. If, for example, the user typed 0478 when intending to look at location 047B, would simply type 047B.

All of the non-command keys and non-hexadecimal characters are ignored by the monitor.

Label	Program Entry Points
VM	FE00 - Restart Location
	FE0C - Bypasses UART and Stack Pointer initialization and the clearing of decimal mode but does clear the screen.
IN	FE43 - Entry into address mode, bypass initialization
INNER	FE77 - Entry into data mode, bypass initialization

Label	Subroutines
OUTER	FE80 - Input an ASCII character form Audio Cassette UART
HEXAL	FE93 - Returns stripped ASCII number if 0-9 or A-F. Otherwise returns a FF.

INPUT FEED - Input an ASCII character from keyboard.

Required Hardware

The 65V Monitor requires as a minimum the following hardware: an OSI Model 400 board with a 6501, 6502, or 6512 microprocessor, 1,024 words of RAM memroy located from 0000 to 03FF, and the 65V monitor itself. It also requires an OSI Model 440 Board propulated for alphabetic display and keyboard input. The 440 Video Board must be located at DXXX which will automatically locate the keyboard input at DFXX.

The keyboard must be a seven bit high true ASCII keyboard with a positive or negative going strobe pulse at least 100 microseconds long.

The 65V Monitor will additionally support input from a generalized serial communications subsystem of an OSI 430 board located at FBXX. Specifically, the monitor contains a load program for a 430 board based audio cassette interface. The same program can be used with a 430 board configured for digital cassette or ASCII teletype input.

** OSI 65V **

```
FE00
A2 28 9A D8 AD 06 FB A9
FF 8D 05 FB A2 D4 A9 D0
85 FF A9 00 85 FE 85 FB
A8 A9 20 91 FE C8 D0 FB
E6 FF E4 FF D0 F5 84 FF
F0 19 20 E9 FE C9 2F F0
1E C9 47 F0 17 C9 4C F0
43 20 93 FE 30 EC A2 02
20 DA FE B1 FE 85 FC 20
AC FE D0 DE 6C FE 00 20
E9 FE C9 2E F0 D4 C9 0D
D0 0F E6 FE D0 02 E6 FF
A0 00 B1 FE 85 FC 4C 77
FE 20 93 FE 30 E1 A2 00
20 DA FE A5 FC 91 FE 20
AC FE D0 D3 85 FB F0 CF
AD 05 FB 4A 90 FA AD 03
FB 8D 07 FB 29 7F 60 00
00 00 00 C9 30 30 12 C9
3A 30 0B C9 41 30 0A C9
47 10 06 38 E9 07 29 0F
60 A9 80 60 A2 03 A0 00
B5 FC 4A 4A 4A 4A 20 CA
FE B5 FC 20 CA FE CA 10
EF A9 20 8D CA D0 8D CB
D0 60 29 0F 09 30 C9 3A
30 03 18 69 07 99 C6 D0
C8 60 A0 04 0A 0A 0A 0A
2A 36 FC 36 FD 88 D0 F8
60 A5 FB D0 91 AD 01 DF
30 FB 48 AD 01 DF 10 FB
68 60 30 01 00 FE C0 01
```

The following is a program which may be entered using the 65V Monitor from the keyboard or audio cassette. An "*" indicates a return key depression.

```
.0002  Loads the ASCII Message Starting at
       Location 0002
/4F * 53 * 49 * 20 * 36 * 35 * 56 * 2E * 5F

.0200  Loads the Main Program at 0200
/A9 * 02 * A2 * 00 * 20 * 00 * 03 * A2 * 00
* 20 * ED * FE * 9D * 24 * D2 * E8 * 4C * 0
* 02

.0200  Loads the Subroutine at 0300 to Out-
       put an ASCII Character String
/85 * 00 * A9 * 00 * 85 * 01 * A0 * 00 * B1
* 00 * C9 * 5F * F0 * 0A * 9D * E4 * D1 *
E8 * E6 * 00 * D0 * F2 * E6 * 01 * 60

.0200G Loads the Starting Address of the Pro-
       gram and Execute it.
```

You should see the message "OSI 65V." on the screen. Now, you may type any keys and they will be echoed just below the message. Press reset to re-enter the 65V Monitor.

If this were entered off of the audio cassette, it would be self-loading and auto-starting. Since the cassette is in complete control, it can load the starting address and execute the program without user interruption.

Creating Data Files in BASIC

Both 8K BASIC under OS-65D and the new ROM version of 8K BASIC are capable of files based on mass storage devices. 8K BASIC in ROM is capable of simple cassette-based sequential files. The BASIC for disk is capable of sequential, random access, indexed sequential, and other advanced file structures.

Let's first discuss why files based on mass storage devices are desirable. The first obvious reason is that we would like to store more data in the machine than our RAM memory will allow. Secondly, we would like to store information on a permanent basis so that if we enter it into the machine once it is always there. Thirdly, we would like to provide reference material or a library of data or information for programs to act on. Files are a necessity for applications such as small business programming. There are also applications in personal and home computing and educational computing which require the use of data files.

The simplest file from an organization and performance point of view is the sequential file. In this type of file, entries of variable length are simply made sequentially. To access any entry, the user must sequentially read through the file until he finds it. Sequential files are fine for applications such as mailing lists where one will normally want to output the file in a sequential manner. However, they have extremely long seek times when the user is looking for a particular piece of data. Manually operated audio cassette systems are capable only of sequential files because the recorder must be advanced, or read, at normal speed to get to the desired piece of data. For this reason, cassette-based files are very slow and not really practical for any business applications on the computer.

If the data in a file can be formated in some manner and put in some order such that the position of the data

can be predicted by some mathematical equation, then a random access file can be used. Random access files are extremely fast because you can directly read the data that you desire. However, in many cases it is not feasible to provide a logical, calculable organization for your data file such as in business applications. The net result is that random access files are fast, but do not lend themselves to many applications while sequential files are easy to use, but,very slow.

The index sequential file is a two-level file system merging the features of random access and sequential files. Index sequential files are most commonly used in applications such as business computing. An index sequential file is actually made up of two separate files: one file with the indexes, and one with the actual entries. An index sequential file works on the same principle as a standard library. The user has a catalog of available documents and the documents themselves. The catalog is the index, and the documents are the actual data file.

The index file is typically a short sequential file which has a key word entry followed by the index which points to the larger data base. This index is then used to randomly access the large data base. To clarify this, consider the example of a student report card record. Each student would have a group of entries (e.g., name, social security no., date of birth, etc.), followed by the classes he is enrolled in and his grades for those classes. The record for each student may contain thousands of bytes. Students may be placed in the file in any order as they are admitted to and withdrawn from the school. As a simple file, it would take an extremely long time to locate any student in the file because one would have to search through a large data base to find a student's name. If the file were set up as a random access file, the entire file would have to be reorganized every time there was a new enrollment or withdrawal of student. This operation may take several hours if the data base is particularly large.

The solution is an index sequential file. The index file simply contains the student's name and the index which

points to the location of his actual file. The large data base contains the files. To access an individual student's records, the student's name is typed in, the index file is brought into memory, searched for the index, and the index is used to pull in the student's file.

There are more advanced file systems which utilize the index sequential concept.For instance, it is possible to devise files which are both random access and sequentially addressable. This can be accomplished on an OSI computer system by preloading the file with a field of null characters. Then a random access file with variable length entries can be accessed sequentially if desired. There are techniques where files are inverted, or the data in files are placed in different orders to allow easy sorting and merging and accessing. However, the user would do well to master the concepts of the simple sequential, random access and index sequential file systems.

GUIDELINES FOR FILES WITH OSI BASIC

As mentioned above, simple sequential files are possible in ROM BASIC. Here are two routines that can be incorporated into any program to write an array out to a cassette and to read an array in from a cassette. These examples demonstrate fairly well the capability of the sequential file system in ROM BASIC. The reserved word SAVE switches are output from video display or serial port only to also include the audio cassette, so that anything that goes out onto the terminal will also go out to the audio cassette. The reserved word LOAD switches input from being solely from the keyboard of the terminal to being from either keyboard or cassette. The first time the keyboard is actuated by the user after the LOAD command is executed, the input will revert solely back to the keyboard.

In summary, the reserved word SAVE turns the cassette output on; the reserved word LOAD turns if off and turns cassette input on; and any intervention at the

keyboard by the user shuts off cassette input. Data input and output to cassette is by simple PRINT and INPUT statements. It is important that the user remember to follow each data output by a carriage return so it can be accepted by an INPUT statement later. It is a good idea to prompt the user in cassette operation of these routines, as in lines 500, 550, and 560 below.

SIMPLE CASSETTE SEQUENTIAL FILES WITH ROM BASIC

Routine for storing an array

```
500 PRINT "TURN CASSETTE RECORDER ON RECORD."
510 SAVE
520 FOR X=1 TO I
530 PRINT T(I)
540 NEXT
550 LOAD:PRINT"TURN RECORDER OFF."
560 INPUT "TYPE ANY KEY TO CONT.";A$
```

Routine for recalling the array

```
1000 PRINT"TURN CASSETTE RECORDER ON."
1010 LOAD
1020 FOR X=1 TO I
1030 INPUT T(I)
1040 NEXT
1050 PRINT"TURN RECORDER OFF."
1060 INPUT "TYPE ANY KEY TO CONT.";A$
```

OS-65D BASED FILES

OS-65D allows full user data files in 8K BASIC and machine language programs. The file system is totally open-ended, allowing sequential, random access, index sequential, and complex file structures, such as indirect command files, and multiple files of different length to be open at any given time. Because of OS-65D generality in having variable sector lengths and varia numbers of files open at a time and total control of f operations, it is both extremely powerful and very dif cult to implement. The difficulty in implementation is mainly due to the fact that there are virtually no err messages or error traps for operator errors because th are practically no illegal operations. In other words,

since it is possible to have variable sector lengths and any number of files open simultaneously, it is impossible for the operating system to determine whether the user is attempting a legal or illegal operation since virtually any operation would be legal under some circumstances. For this reason, it is advised that the beginner with OS-65D confine himself to modifying existing file-based programs until he gets a good understanding of what is going on and refines his own personal programming procedures. Otherwise, he will be constantly plagued with system crashes and pissibly the loss of important data on diskettes.

From here on, we will confine our discussion to data files in BASIC on OS-65D. A simplified picture of what is required of a data file in BASIC in OS-65D is that the BASIC program performs conventional INPUT and OUTPUT statements to a memory buffer. This buffer is periodically transferred to and from disk. It is possible to have any number of these memory buffers existing with different memory files present in them at any one time, limited only by the total amount of memory your system has. This discussion will limit itself to one- and two-buffer systems, however.

CREATING A FILE

The procedure to create a simple file is, first, to create a memory-resident file buffer. This is accomplished simply by allocating space for it at the top of memory when BASIC asks "MEMORY SIZE?" Secondly, you must set the I/O index pointer to the beginning of the file. Thirdly, you must, by use of the I/O distributor, output the desired data to the file. Finally, when you are ready, or that particular buffer is full, you must initiate a transfer from the buffer to the disk itself.

READING A FILE

First you must create a memory-resident file buffer, again, by simply allocating space for it when the machine asks "MEMORY SIZE?" upon power-up configuration.

Secondly, you must transfer data desired from the d to memory. Third, you must set the input/index poin (I/O pointer) to the beginning of the file, or wher ever in the file you would like to gain data from. And finally, you must by use of the I/O distributor input to an INPUT statement from a file.

To implement a simple sequential file is extremely straightforward, because with sequential files, the indexes on input and output simply move sequentially In OS-65D, the disk indexes, or file buffer indexes, automatically increment after each character is out- puted, To implement a random access file, you must have a way of calculating the input and output in- dexes by some formula. An example of this is dis- cussed at the end of this article.

To implement an index sequential file, two files ar needed, an index file and a data file. It is possibl to have the index file and data file both utilize th same buffer by swapping in the index when desired, a then the data. However, with OS-65D, it is fully pos sible to allocate memory space for both the index fi and part or all of the data files concurrently. Thus extremely fast access to any block of data is possib The index file is typically organized as a simple se quential file, and the data file is organized as a simple random access file. The index obtained from t index file is used to directly access data in the da file. Of course, other, more complex file systems ar possible under OS-65D. Some examples of this are in- direct command files which allow the disk file to ac as the executive of the computer system. Also, becau of the multiple simultaneously open file capability OS-65D, merges and sorts of multiple files are both straightforward and extremely fast.

OS-65D Version 2.0 diskettes contain three examples of data file-based programs in BASIC. A simple tele- phone directory program is located on track 18. It i documented as Example 5 in OS-65D Version 2.0 manual This telephone program simply allows a user to enter a person's name and telephone number and create a si

track or 3,000-byte long file containing these names on disks. He can then at any later time go back to the program and call up a phone number based on a person's name. This program is a good example of a simple, single-track sequential file and of course can be readily changed for multiple entries such as for mailing lists where three or four entries would be combined to form a record, such as a person's name, street address, city, and state. It has two limitations, however. It does not have a delete function, and it limited in size to be only one sector, that is 256 to 3000 words. It cannot exceed one track on length. A refinded version of the same telephone program is located on track 44. This is an extension of the simple telephone program which, again, can be easily modified and converted for any application requiring a sequential file. This program has a delete function and is set up for maximum file length of four tracks or 15,000 bytes. It can easily be changed to utilize up to an entire diskette so that a very large mailing list, for instance, could be stored. To conserve space, the program is not listed here. Instead, we ask that you list the program on track 44 in conjunction with this discussion. Make sure that you have a standard Version 2.0 diskette. The extended phone program is very similar to the original phone example except for the delete and multiple track capability. A delete is accomplished by replacing each entry, that is, name and phone number, with the character S, so that if you search for S, you will find all the deleted entries. The process of deleting an entry is to delete the old entry and place the new entry on the end. The program is set up to start at disk file on track 72 and expand downward to track 76. The program can be expanded to any length maximum by simply specifying the variable II starting at some track lower than 72. It accomplishes its multiple track nature by constantly checking for end of sector, which in this case equals one track in length. When it approaches the end of a track, it transfers that track out, resets all pointers, increments the track counter, and starts to fill

up the buffer for the next track. The program can be directly converted to place the data files on disk drive B by simply changing the variable DN from input and output to drive A to input and output to drive B. With this method, it will be possible to utilize an entire disk as a file for this program.

Using anentire diskette as a file system, and assuming approximately 60 bytes per mailing label, the user should be able to place up to 3800 entries on a single diskette.

The disk-based random access file demonstration program is located on track 42 of OS-65D Version 2.0 diskettes, and the data file, including data, is located on track 43. Please list the program on track 42 in conjunction with this discussion. An operational printout from the file is listed on p.7. The demonstration program utilizes a 3K byte data file which stores a functional description of the 7400 series TTL components. The user inputs a 7400 series TTL part, of which he wants a description and from the part number, the index or description's address is calculated so that the program can virtually instantaneously obtain the desired data. The data file provided contains descriptions of 7400 series devices from 7400 to 7450. To run the program, be sure that you respond to MEMORY SIZE? with the number 16000 in order to allocate space for the disk buffer and that you use it with a serial terminal or modify the I/O to return control to a video system, if video is used. A sample printout is given on p.7. This simple sample of a random access file can be expanded to any random access application.

By utilizing a sequential file to obtain indices, such as the simple phone program listed in the OS-65D manual and using a random access file such as the 7400 series reference file, one can construct a general purpose indexed sequential file system which will access data as fast as any standard small business computer on the market today and significantly faster than most.

OS-65D places a lot of responsibility on the programmer's shoulders. However, if a program is properly de-

veloped by using OS-65D, it will achieve an extremely high level of performance with a minimal amount of memory overhead. We believe this to be the most important aspect of small computing. That is, in applications situations such as small business computing, the program must be developed once, then used many times. It is not efficient, or particular intelligent, to compromise the computer's performance on a day-to-day basis for the sake of making the task of a programmer a little easier. This unfortunately has been the case on the few other small computer systems which offer true disk file capability.

Ohio Scientific is in the process of developing resale arrangements for several applications packages written in OS-65D. The software packages currently under development include a word processor (now being utilized in the production of the Journal), a complete mailing list program (which was used to generate your mailing label), a complete business package, including accounts receivable, accounts payable, general ledger, inventory, and income tax; and a data-based management system for use with user programs. We would be very interested to hear any recommendations you might have for software packages to use under OS-65D. Please send your suggestions to ELCOMP Publishing, Inc.

```
RUN
OSI DISK BASED 7400 SERIES REFERENCE FILE THE AVAILABLE
COMMANDS ARE AS FOLLOWS - -

NEW
ADD
END
SEARCH

IF FILE INFORMATION IS DESIRED TYPE 'SEARCH' IN RES-
PONSE
TO THE PROMPTING 'COMMAND' PRINTOUT

INPUT COMMAND MODE
? SEARCH
INPUT 7400 SERIES PART NUMBER - E.G. 7400, 740
4, ETC.
? 7420
```

```
? 7420 DUAL 4-INPUT NAND GATES
? END

INPUT 7400 SERIES PART NUMBER - E.G. 7400, 740
4, ETC.
? 7450

? 7450 DUAL 2-WIDE 2-INPUT AND-OR-INVERT GATE
S
? END

INPUT 7400 SERIES PART NUMBER - E.G. 7400, 740
4, ETC.
? 7460

NUMBER OUT OF RANGE
THE AVAILABLE COMMANDS ARE AS FOLLOWS -
NEW
ADD
END
SEARCH

IF FILE INFORMATION IS DESIRED TYPE 'SEARCH' IN RES-
PONSE
TO THE PROMPITNG 'COMMAND' PRINTOUT
```

COPYING OS-65D V3.1 MINI-FLOPPY DISKETTES ON SINGLE DRIVE SYSTEMS (REQUIRES 20 K RAM MIN.)

Many people have run into the problem of backing up their mini-floppy diskettes on a single drive system. The following description should explain the proper procedure for backing up one's software.

The basic scheme for backing up involves initializing the new diskette and then transferring each sector from the original diskette to the new diskette.

To initialize a blank diskette:

1) Boot up under OS-65D as one normally would.

2) Enter the D.O.S. by exiting basic i.e. type exit
   ❮CR❯
   After entering the line above one should see a A* on the screen.

3) Initialize the blank diskette by entering
   IN ❮CR❯
   the question:
   ARE YOU SURE?
   will appear on the screen, answer this question by entering
   Y
   when the diskette has been fully initialized the computer will print
   A*
   to the screen

4) Insert the original diskette into the floppy drive

5) Type the following:
   CALL Ø2ØØ=13, 1 ❮CR❯
   this calls the disc copy utilities into memory

6) Now enter:
   GO Ø2ØØ ❮CR❯
   A menu will appear on the screen. Select item number two (2) from the menu

7) Now type:
   R42ØØ ‹CR›
   this calls the contents of track Zero into memory

8) Insert the new diskette into the floppy drive

9) Now enter:
   W42ØØ/22ØØ,8 ‹ CR ›
   At this point track zero has been placed on the new diskette

The remaining steps concern themselves with 'CALL'ing each sector on the original diskette into memory and then transferring that sector onto the new diskette.

The commands used to do this are:

```
DIR TRACK NUMBER ‹ CR ›
     (TWO DIGITS)
CALL 42ØØ=TRACK NUMBER, SECTOR NUMBER ‹ CR ›
          (TWO DIGITS), (ONE DIGIT)
SAVE TRACK NUMBER, SECTOR NUMBER=42ØØ/LENGTH ‹ CR ›
     (TWO DIGITS),  (ONE DIGIT)
```

The DIR command is used to determine the number of sectors on a given track and the sectors length. The CALL command is used to bring a sector into memory from the original diskette. While the SAVE command provides a means for placing the sector 'CALL'ed into memory on the new diskette.

10) Insert the original diskette into the floppy drive

11) Enter a DIR command for the track to be transferred into memory. For example:

IF TRACK 13 CONTAINED 4 SECTORS EACH ONE PAGE LONG
THE COMMAND:
DIR 13 ‹ CR ›
WOULD PRINT:
TRACK 13
Ø1-1
Ø2-1
Ø3-1
Ø4-1

the first two digits tell the sector number and the last digit defines the sector's length in pages

13) From the information provided by the DIR command one should now proceed to 'CALL' each sector off the track to be transferred to the new diskette one at a time.
Using track 13 as an example:
For the first sector enter:
CALL 42ØØ=13,1 ⟨CR⟩
This calls the first sector on track 13 into memory

14) Now insert the new diskette into the floppy drive

15) Proceed by "SAVE"ing the sector to the new diskette by entering:
SAVE 13,1-42ØØ/1 ⟨CR⟩
This saves the sector in memory to track 13 sector one with a length of 1 page.

16) Step 10-15 should be repeated for tracks 1-13 and for any other tracks containing programs one desires to place on the new diskette.

9- DIGIT BASIC VARIABLES

This article shall attempt to explain how BASIC stores and accesses variables. There are three types of variables in BASIC. They are floating point, and string variables. The three types of variables may be non-subscripted, i.e. simple variables or sub-scripted variables.

VARIABLE STORAGE

Diagram one shows where in memory variables are stores. The BASIC program starts at the address pointed to by TXTTAB. Simple variables start just after the BASIC program. Immediately following simple variables in memory are array variables. Following the array variables in memory is the free memory space. After the free memory the actual string data is stored.

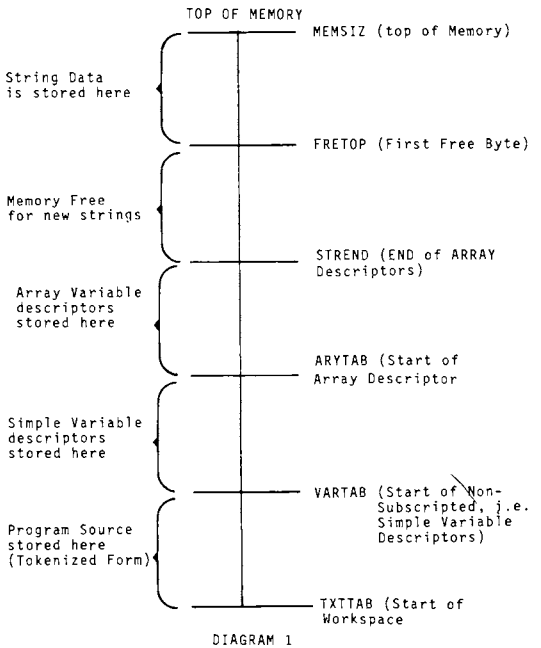


DIAGRAM 1

BASIC PROGRAM AND VARIABLE STORAGE

DIAGRAM I

VARIABLE DESCRIPTION
BASIC must have a way of "keeping track" of all variables. In order to do this, BASIC retains a descriptor for each variable in the program. A descriptor contains, in the case of numeric variables, the current value of that variable. String descriptors contain information on the length of the string and it's location in memory.

Upon encountering a variable for the first time, BASIC constructs a descriptor for that variable. If the variable is a subscripted variable, BASIC constructs an array descriptor. New simple variables are tacked on to the end of the other simple variables. Insertion of a simple variable requires that the array variables first be moved to make room. Subscripted variables are "tacked" on to the end of existing subscripted variables. Adding a subscripted variable decreases the amount of free memory space. Because array variables must be shifted upward to accommodate new simple variables, simple variables should be defined early in the program. DIM statements, when encountered, force BASIC to set up an array descriptor large enough to accommodate the size given in the DIM statement.

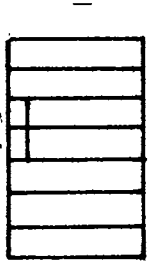
SIMPLE FLOATING POINT DESCRIPTION
Simple floating point numeric variables have a descriptor containing the variable name and it's value. Diagram 2A shows the descriptor. Two bytes are always reserved for the variable name regardless of it's length in the program.

Floating point numbers are always stored in memory in a "NORMALIZED" form. "NORMALIZED" means that the binary number is shifted to the left until the most significant bit (MSB) is a one. The fractional value's sign bit is then placed into the MSB of the fractional value.

The MSB of the fractional value is implied to be a one unless the number is zero. A variable whose value is zero has it's exponent set to zero. In all cases, the binary point is implied to be to the left of the MSB in the fractional value.

SIMPLE INTEGER VARIABLE DESCRIPTORS

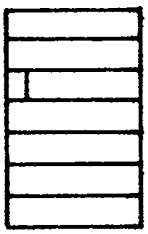
A



SIMPLE FLOATING POINT VARIABLE DESCRIPTOR
First character of name
Reserved for second character of name
Signed exponent with the sign bit complemen
Fraction value sign bit, MSB fraction valu
Fraction value
Fraction value
LSB of fractional value

Sign bit →

B

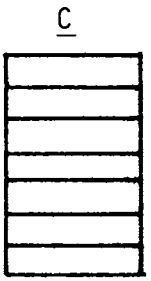


SIMPLE INTEGER VARIABLE DESCRIPTOR
First character of name +$8Ø
Second character of name +$8Ø
Sign bit, MSB
LSB
Ø - meaningless
Ø - meaningless
Ø - meaningless

Diagram 2B shows the format of a simple integer descriptor. As with floating point variables, two bytes are always allocated for the name. BASIC requires a method by which it may differentiate between floating point, integer and string variables. The method used is a simple one involving the variable name. Floating point names are not changed, however string and integer names are affected. Integer names always have $8Ø added to the two bytes reserved for it's name. String variables have $8Ø added to the second byte reserved for the name. Integers are stored in two's complement form with bit 7 of the

most significant byte reflecting the sign. Integers are stored in two bytes with 15 bits reserved for the value. This limits the range of an integer variable to be between -32767 and +32767.

SIMPLE STRING VARIABLE DESCRIPTORS
Diagram 2C illustrates the format used for simple string descriptors.



SIMPLE STRING VARIABLE DESCRIPTOR
First character of name
Second character of name +$8Ø
Length of string
Low address of string
High address of string
Ø - meaningless
Ø - meaningless

DIAGRAM 2

A string variable always has $8Ø added to the secondbyte reserved for the name. String descriptors, unlike numeric variable descriptors, do not contain the value of the variable. A string descriptor contains the length of the string and a pointer to the actual string data. Strings maybe located within the actual program next or within the string space stored at the top of memory. The location of the string is dependent on how the string was defined. If a string variable is equated to string data within quotes, then that string's descriptor would point into the program text. A string variable equated to a CHR$, LEFT$, RIGHT$, or MID$ function will create a string

in string space. INPUT statements will also cause a string to be formed in string space. If one string variable is equated to another, that string's pointer may or may not point into string space. Where the string descriptor points to is dependent on where the strin variable to the right of the equal sign is located. If it is located within the program, then both descriptors will point into the wor space. If the string that is being equated to resides in string space, then that string wil be copied to the bottom of free space and the second string's descriptor will point to the copy.

ARRAY VARIABLE DESCRIPTORS

Diagram 3 illustrates the format of array variable descriptors. The array descriptors are basically the same as simple variable descriptors. The differences are information that defines the length of the array, the number of subscripts and the maximum value for each subscript, i.e. DIM A$ (2Ø, 1Ø) would set the maximum subscripts to be 2Ø and 1Ø respectively. If an array variable is encountered that has not been DIMensioned, BASIC will default to a DIMension of 1Ø. In other words the array will be set up as if a DIMension of 1Ø had been executed. It is important to note that DIM statements must be executed to be effective. In memory arrays are stored as sequential lists. The formula for accessing an array entry is:

For a List Array A$ (N):

Address of descriptor	= N *	Length of descriptor	+	Starting Array addr

e.g., N=50: Array data starts at $6ØØA

Address = (5Ø*3)+$6ØØA=$96+$6ØØA=$6ØA

Multiple subscripted variables are slightly more involved. Diagram 4 shows a string array in memory. The array has been DIMensioned to 2 by 3. The formula for calculating the offset into an array to access a specific variable is shown below.

```
Offset=Length of descriptor*(S1+((S1max
                +1)*S2))
```

Working through an example should clarify the procedure. The example will use the array A$ (X,Y). The array has been DIMensioned as DIM A$(2,3). The descriptor length will equal 3, one byte for the string length plus two bytes for the pointer to the actual string data. Since BASIC permits use of a subscript of zero, there are actually 12 variables in the array. The legal subscripts range from A$ (Ø, Ø) to A$ (2,3). We shall refer to the first subscript as S1 and to the second subscript as S2. The maximum subscript for S1 and S2, as defined the the DIMension statement is 2 and 3 respectively. The array is organized with A$ (Ø, Ø) at the front and A$ (2,3) as the last entry. If subscript S1 increases by one the offset increases by the length of one descriptor i.e. by 3 bytes. If S2 increases by one the offset would increase by (S1max+1)*3. This may seem a little strange, so let's examine the array format a little closer. The first variable in the array is A$ (Ø,Ø). The next variable is A$ (1,Ø). In other words the array is in the order A$ (Ø,Ø), A$(1,Ø), A$(2,Ø), A$ (Ø,1), A$ (1,1), etc.

This then indicates that when S1 increases by one, we are moving forward by one descriptor. However, if S2 increases by one, we must move past the descriptors containing the values for A$ (Ø,S2) through A$(S1max,S2).

Using the values in our example, the relative offset to A$ (2,3) would be:

```
  Relative offset = descriptor length
  * (S1+(1)*S2))

    Relative offset=3*(2+((2+1)*3))

 Relative offset + 3*(2+9)=3*11 = 33
```

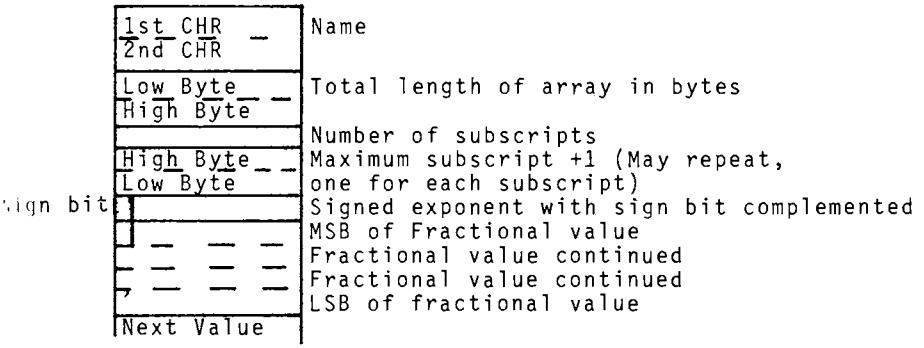
Therefore, if one listed to access A$ (2,3) one would set up a pointer to the start of the array descriptors. Then the relative offset would be added to the array pointer. Next one would pick up the length of the string and the two byte pointer to the actual string data. At this point we have all the information required to access the string data of A$ (2,3). Variable descriptors for array variables with more than two subscripts follow the same BASIC format. A generalized form for accessing a descriptor of a variable having N subscripts is shown below.

```
Relative offset + DL (S1+(S1max+1)*S2+(S2
max+1)*S3+.......)
```

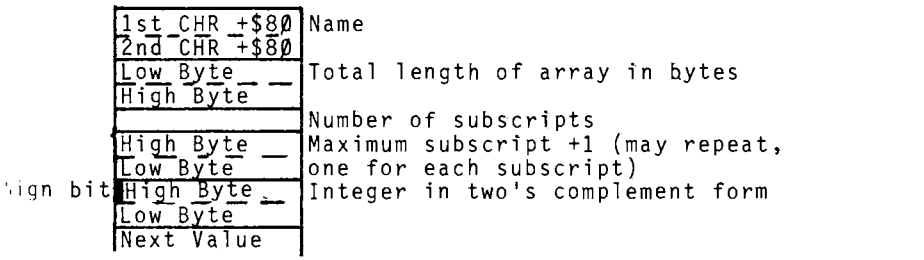
Where DL is the length of the descriptor

```
S1 is the first subscript
S1max is the maximum value of S1
S2 is the second subscript
S2max is the maximum value of S2
S3 is the third subscript
etc.
etc.
```

FLOATING POINT ARRAY VARIABLE DESCRIPTOR



INTEGER ARRAY VARIABLE DESCRIPTOR



STRING ARRAY VARIABLE DESCRIPTOR

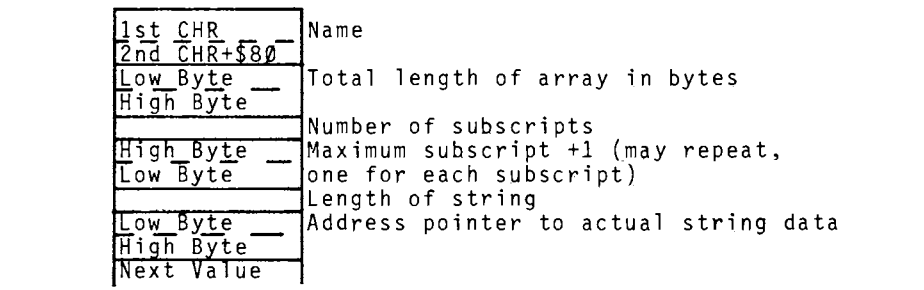
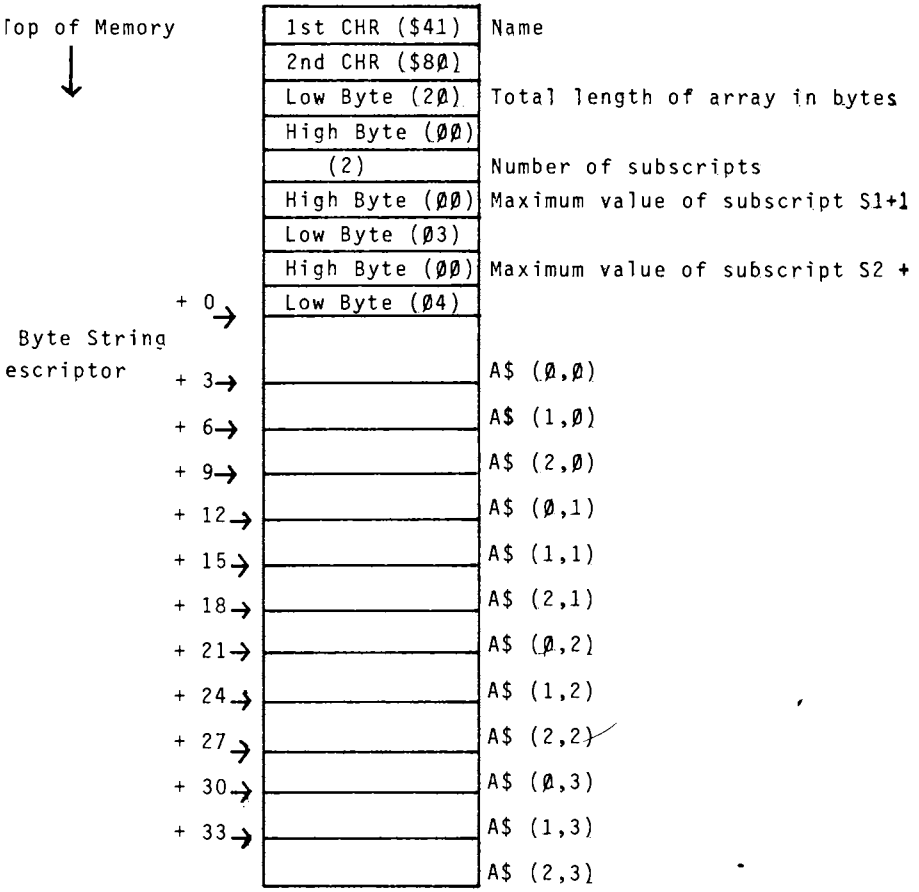


DIAGRAM 3



ARRAY A$ (2,3) IN MEMORY

DIAGRAM 4

High-Resolution Display Conversion for Challenger 1P

by Steven Chalfin

The CHALLENGER 1P (SUPERBOARD II) computer system is one of the best values in hobby computers today. Its many outstanding features are well known. However, its low price has necessitated several design compromises. One of these compromises involves the video display circuitry. The advertised 32 x 32 format is actually only 24 x 24 or at best 26 x 26 on a standard television. Even if a high quality monitor is used and the display size can be increased, the BASIC in ROM will still limit the line length to 24 characters.

Adding guard band circuitry would give a true 32 x 32 display on a standard television, but would be both expensive and technically difficult to implement. The most cost-effective solution this author has found, and the one which is described in the conversion instructions, is outlined below:

1) Double the dot clock frequency. This has the effect of putting out twice as many dots per scan line (twice as many characters per line)

2) Since there will be twice as many characters per line but the same number of lines on the screen, and additional 1K of video memory must be added.

3) Associated circuit changes for memory decoding, sync pulses, etc.

4) A software patch so that BASIC will use the entire screen to display.

These modifications yield a 32 line x 64 character display. Since this circuitry still lacks guard band capability, some characters are lost off the screen due to overscan. However, the resultant 30 line x 50 character display is a vast improvement.

It is assumed that the person undertaking this conversion is familiar with such basic electronic construction techniques as soldering and reading schematics. Components required for this conversion are:

QUAN	DESCRIPTION
4	2114 or 2114L 450nS or faster RAM*
1	8.0 MHz crystal (actually, 7.86432 is ideal)
1	74LS163 or 74LS161 4 bit binary counter +
1	74LS139 2 to 4 decoder (dual)+
2	16 pin DIP sockets

*This is critical! Don't buy slower (cheaper) RAMs as they will not work.

+ Be sure to use the 74LS series, not the 74 series as these are slower and may not work.

Also required: Model 600 schematic, solder #30 AWG wire wrap wire, soldering iron, wire stripper, razor blade or X-acto knife.

These parts may be had for $ 40 or less, depending on how well stocked your junk box is.

The complete conversion instruction is available from myself or from Silver Spur at a cost of $ 12.00. The crystal is $6.95, from Silver Spur.

Here is an update of the CIP Video Modification, containing an important correction, as well as tried and true data for owners of mini-floppy drives using the OS65-D operating system.

1. Error in original instructions:
   Instructions now being mailed out are correct, but some of the original instructions fail to disconnect the old Sync one-shot (U65). (U61 pin 11 to U65 pin 1). This trace is easily located with an ohmmeter near U65. As I recall, it passes from pin 1, on the component side of the board, beneath the body of U65, to a plated-thru hole near pin 7 of U65. It is easily cut at that point. Failure to cut this trace results in both U61 and U27 (the new chip you added) driving the Vertical Sync circuit. This results in a loss of Vertical Sync and your display will roll vertically for ever and ever and...

2. Changes to OS65-D for 52 X26 Video Display
   Owners of mini-floppy CIP's: Take heart!!
   a) You need not make the PROM change suggested in the modification instructions, though you may if you desperatly desire a neat D/C/W/M? prompt on reset, or a super convenient display from the PROM ("M") monitor.
   b) Included herein are the necessary software changes for the I/O drivers in OS65-D.
   c) If your memory is fast enough and you got a 6502A 2 MHz processor (or wanna spend 19 bucks and plug one in), then I enclose the single, simple, modification to the OSI #610 Disk controller/memory board.
   d) Mine works, and at 2 MHz too. (Note: If you do not wish to run at 2 MHz, then just perform the original modification, complete with the 1 MHz "step-down", and keep your hands off the 610 board).
   e) The Software modification is accomplished within the bounds of track 0. Therefore, we load track 0 into high memory, operate on it, place it on a fresh disk, using the TRACK 0 READ/WRITE UTILITY on track 13. The rest of OS65-D is then duplicated onto this disk, and the alterations verified by booting this disk. Track 0 may then be loaded and written onto all disks you wish to update to 52-character video. The driver works with BASIC, the ASSEMBLER, and the EXTENDED MONITOR. Clearly, you could modify the operating system currently active in memory using pokes from BASIC. In that fashion, the changes could easily be incorporated into BEXEC*. In such a case, you would necessarily adjust the HEX addresses given downwards by their current offset of $ 2000. Such an approach would be easy and safe for debugging a newly modified machine.

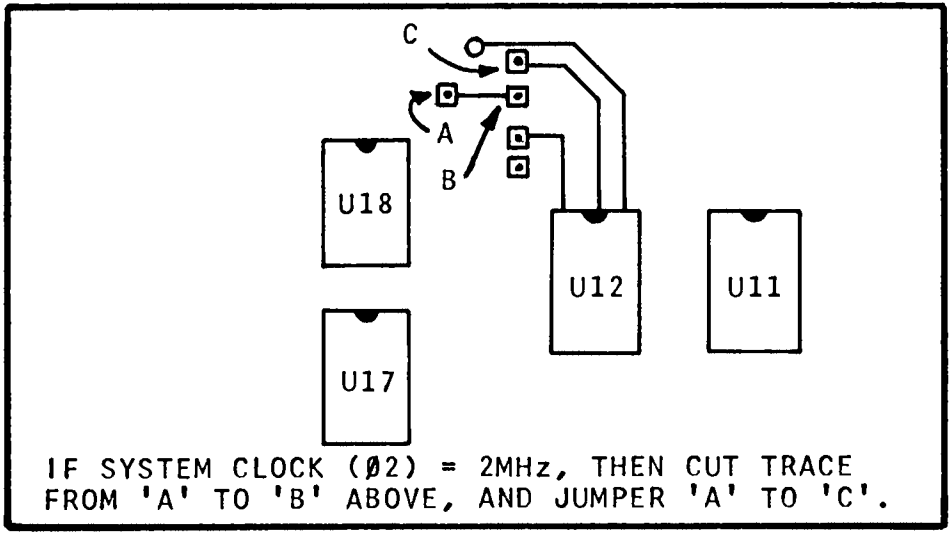


FIG 1. COMPONENT SIDE: 610 BOARD (DISK CONTROLLER)

Video Update to OS65-D, For C1PMF

VIDEO UPDATE TO OS65-D, FOR C1PMF, accompanying 52 char.mod:

First commandment: Never write on your original OS-65D diskette!

```
Boot Disk
UNLOCK (CR)
EXIT (CR)
A*CALL Ø2ØØ=13,1(CR)              NOTE: Operator entries are
A*GO Ø2ØØ (CR)                          underscored

- DISKETTE UTILITIES -
1) COPIER
2) TRACK 0 READ/WRITE

?2 (CR)

-TRACK 0 READ/WRITE UTILITY -
COMMANDS:
Rnnnn-READ INTO LOCATION nnnn
Wnnnn/gggg,p-WRITE FROM LOCATION
      nnnn FOR p PAGES WITH
      gggg AS LOAD VECTOR
E-EXIT TO OS-65D
COMMAND?R4200(CR)
(repeat of above command listing by computer)
COMMAND?E(CR)

A*EM(CR)                          (boot extended monitor)

EM V2.0
:@45A4(CR)
45A4 / 65 48(CR)                  (changes this location from $65 to
                                  $48.)
                                  =ADDRLO of STARTCURSOR
                                  (similarly, make the following changes)
```

ADDRESS	ORIGINAL	NEW	NOTE
45C3	D3	D7	= ADDRHI of STARTCURSOR
45C6	7D	7C	= ADDRLO of ENDCURSOR
45CD	D3	D7	
45D5	D3	D7	
45FB	1F	3F	= test right 5 bits for 64 char. line
45FD	04	07	= left overscan -1
4602	08	0C	= left+right overscan
4608	1F	3F	
460A	1D	3C	= #used char per line+left overscan
460F	07	0B	= left+right overscan -1
4616	D3	D7	
461D	D3	D7	
4624	D3	D7	
4626	65	48	= ADDRLO of START CURSOR
462A	65	48	

```
4630                D3          D7
4637                7D          7C
463A                D3          D7
4648                20          40 = scroll by 64 addresses
4649                D3          D7
4653                D3          D7
45D2  put hex. equiv. of desired cursor character here!!
      (See Character Graphics Ref. Manual for favorite).

having made the above changes,

:EXIT(CR)           (NOW PUT A BLANK DISKETTE IN THE 'A' DRIVE....
                                              BE SURE!!!)
A*INT(CR)
ARE YOU SURE?Z      (THE blank disk is now formatted)

A*GO 0200(CR)
- DISKETTE UTILITIES -
1) COPIER
2) TRACK 0 READ/WRITE
YOUR SELECTION
?2(CR)

- TRACK 0 READ/WRITE UTILITY -
COMMANDS:
Rnnnn-READ INTO LOCATION nnnn
Wnnnn/gggg,p-WRITE FROM LOCATION nnnn FOR p PAGES
            WITH gggg AS LOAD VECTOR
E-EXIT TO OS-65D
COMMAND?W4200/2200,8(CR)

(Repeat printout of above prompts, then:)
COMMAND?E(CR)

A*GO 0200(CR)       (SINGLE DRIVE OWNERS OMIT BEGINNING HERE,
-DISKETTE UTILITIES - AND INSTEAD, "CALL" EACH TRACK, SECTOR
1) COPIER             INTO MEMORY FROM ORIGINAL DISK, INSERT
2) TRACK 0 READ/WRITE THIS NEW DISK, AND "SAVE" THE TRACK,
YOUR SELECTION        SECTOR FROM MEMORY ONTO THE NEW DISK).
                      (I.e.:
?1(CR)                Insert old disk
- DISKETTE COPIER -   CALL 01,1=4200
FROM DRIVE (A/B/C/D)?A(CR)  Insert new disk
                      SAVE 4200=01,1/8
(put an old disk in 'A'     NOTE THAT THE "SECDIR" UTILITY WILL
TO DRIVE (A/B/C/D)?B(CR)    GIVE YOU THE NEEDED DATA AS TO THE
STARTING TRACK?01(CR)       # OF SECTORS PER TRACK, AND THE NUMBER
ENDING TRACK?39(CR)         OF PAGES LONG EACH SECTOR IS).
READY?Y (CR)                (FOR EXAMPLE, ON ORIGINAL SYSTEM DISK,
.                           TRACK12 IS DIVIDED INTO 4 SECTORS, EACH
.                           ONE PAGE LONG, WHICH MUST BE CALLED AND
.                           SAVED INDIVIDUALLY).
.
```

A* (You have now created an OS-65D disk which will boot up
with the correct video driver intact)

To update an old disk (and I recommend that you keep your paws off your original system diskette!!!!!) simply use the track 0 utility to Read track 0 into memory (R4200 is good) Then insert the disk to be updated into the drive and type COMMAND?W4200/2200,8 (CR)
Insert another disk to be updated and type
COMMAND/W4200/2200,8(CR)

However I recommend against trying to update all your disks until after you have produced the one tried and true re-make of the system disk, and have demonstrated to yourself that is boots up correctly.

Program to Circumvent the Garbage Collection Problem in OSI BASIC in ROM Computers

THE PROBLEM:
OSI's BASIC in ROM computers have an error in the Garbage Collection routine that severely restrict their use for programs with string arrays, particularly if concatenation of strings is extensive.

Strings that are not defined between quotes in a BASIC statement are stored in the "string space" at the top of memory. For example, A$="A" is stored in the BASIC statement. However, INPUT B$ puts B$ in the string space with pointers to B$ stored in the BASIC variable table.

On concatenation, say C$=C$+A$, C$ is stored in string space. Suppose we start initially with C$="" and A$="A". We then execute C$=C$+A$. C$ will become "A" and be stored in string space. String space will be "A". Suppose we execute C$=C$+A$ again. Now C$ is "AA" and string space is "AAA". Two of the A's in string space are the new C$ and one is the A left behind in forming the new C$. If we do it again, C$ is $AAA" and string space is "AAAAAA". In this example, three A's in string space are for storing the last designation of C$ and the other A's are previous designations of C$ which are not removed from memory.

Suppose the following program is run.

```
20 C$="":A$="A!
30 INPUT K
40 FORI=1TOK: C$=C$+A$:NEXT
```

C$ is now K bytes long and take K bytes to store. However, we have "used" K(K+1)/2 bytes of memory in generating and storing C$. Thus, in the above program, if K=255(the maximum string length), we need 255 bytes to store the final C$ but we have tried to use 32,640 bytes! Available memory vanishes very quickly if a program contains repetitive operations of this kind.

A memory rearrangement is needed if memory is to be properly utilized. This is accomplished through the Garbage Collection)GC) routine. This routine is called by BASIC when the string space is full. The GC relocates the valid strings back to the top of memory and defines new pointers in BASIC variable space. Using the above numeric example, after GC is executed, 32,385 bytes of memory have been recovered for further use.

OSI's BASIC in ROM GC routine works fine for programs containing numeric variables, string variables, and numeric arrays. The above program can be run on the smallest memory machine without a problem. However, if the program also contains a string array, then the internal GC will not work properly. If, for example, we add the following line to the above program,

```
10 DIM I$(6)
```

and K=255, the internal GC will cause the screen to "pulse" several times at a 1.6 sec. period as the GC routine walks through memory. This pulsing is characteristic of GC failure along with a "dead" keyboard. Extraneous characters may show up on the screen and the BASIC program altered. The execution time without line 10 is under 2 sec. and exceeds 12 sec. with line 10. If the program were entered exactly as written, the pulsing may continue until the computer is reset. Even if the program finally executes, C$ is not placed properly at the top of memory.

A more general program to demonstrate the GC problem is:

```
10 INPUT Q,K
20 DIM L$(Q)
30 FORI=1TO Q
40 FORJ=1TOK: L$(I)=L$(I)+CHR$(64+J)
50 NEXTJ
70 PRINTL$(I), I:NEXTI
```

Here a first order array with diminsion Q is established. Each element is formed from the ASCII code starting at A and contains K symbols. For K=26, each element of the array contains the upper case alphabet in order. At the conclusion of filling each of the array's Q elements with K symbols the element is printed followed by the element number.

For an OSI BASIC in ROM machine with 8K of memory, and K=26, the program will run for Q<=18. If Q exceeds 18 the GC routine fails. Similarly, if K=62, failure occurs for Q>3.

This garbage collection error is very troublesome for any BASIC program that contains string arrays and does extensive string manipulations.

CIRCUMVENTING THE PROBLEM:
Ideally, one would like to correct the errors in the ROM program. One could reprogram a 2716 EPROM with the correct code and substitute it, after some wiring changes, for the incorrect ROM. This is not a simple solution to execute. There is, fortunately, a simpler approach that is useful.

The enclosed listing is a BASIC program that, when run, places a correct GC program at the top of memory. It protects the program from being written over by other BASIC programs. It also sets USR function pointers so that the program can be called by X=USR(X). Finally, it displays on the screen two useful pieces of information. It provides a POKE statement that may be needed to reset the USR pointers if they are changed by another program. It also provides, in decimal, the LOCATION of the GC program called by USR(X).

The steps to use this approach are as follows:

1. cold start
2. Load the listed program.
3. Run the program ONCE. (Each time the program is run after cold start "memory available" is reduced in increments of the pro-

gram length) Run time is about 15 sec.
4. Record the POKE data and LOCATION data for future use.
5. Type NEW and LOAD the program to be run that contains string arrays.
6. Insert X=USR(X) in the program after each major concentration to call the correct GC. Place the POKE statement before this call if USR(X) is used elsewhere in the program to be run.

A DEMONSTRATION:
Take the general program listed above. Add the following line:

```
60 X=USR(X)
```

This cleans up the garbage left after the completion of each string array element, L$(I). With this addition, an 8K machine with J=62 will now operate for Q=75 instead of 3 as before. For J=26 the program will operate for Q over 200. Unfortunately, after 50 or 60 elements are generated, the program slows down noticeably since the GC is moving a large number of strings. This is the penalty paid for being forced to call the GC more frequently than is necessary. It is better to err on the side of conservatism because if the internal GC is triggered the program bombs.

```
LIST

 10 X=PEEK(133):Y=PEEK(134)
 20 L=256*Y+X:L=L-262
 30 Y=INT(L/256):X=L-256*Y
 40 POKE133,X:POKE134,Y
 50 POKE11,X:POKE12,Y
 55 PRINT"POKE11,";X;":";"POKE12,";Y
 60 PRINTL;:A=45383:B=45644
 70 K=L:FORI=ATOB
 80 IFI<>A+34THEN100
 90 M=K+146:GOTO230
 100 IFI<>A+59THEN120
 110 M=K+140:GOTO230
 120 IFI=A+67THENPOKEL,4:GOTO220
 130 IFI<>A+84THEN150
 140 M=K+209:GOTO230
 150 IFI<>A+137THEN170
 160 M=K+146:GOTO230
 170 IFI=A+216THENPOKEL,2:GOTO220
 180 IFI=A+217THENPOKEL,24:GOTO220
 190 IFI<>A+261THEN210
 200 M=K+4:GOTO230
 210 X=PEEK(I):POKEL,X
 220 L=L+1:NEXT:PRINT"LOCATION":END
 230 Y=INT(M/256):X=M-256*Y
 240 POKEL,Y:POKEL-1,X
 250 GOTO220
OK
```

IMPORTANT ROUTINES

1.	Floppy Disk Bootstrap	FF00
2.	65A Serial Monitor	FE00
3.	65H CD-74 Hard Disk Bootstrap	FD00
4.	ROM BASIC Support for 540 Video with polled Keyboard	FF83
5.	65VK Monitor for 540 Video with polled Keyboard	FE00
6.	65K Polled Keyboard polling Routine	FD00
7.	65AB 3.0 ROM BASIC Support for serial Systems	FF00
8.	68A2 serial Monitor for 6800 Microprocessor	FE00

```
FE00  AD00FC      LDA# FC00              -\
FE03  4A          LSR-A                  J
FE04  90FA        BCC FA        *FE00*   Z
FE06  AD01FC      LDA# FC01              -\
FE09  297F        AND# 7F                )
FE0B  48          PHA                    H
FE0C  AD00FC      LDA# FC00              -\
FE0F  4A          LSR-A                  J
FE10  4A          LSR-A                  J
FE11  90F9        BCC F9        *FE0C*   Y
FE13  68          PLA                    H
FE14  8D01FC      STA# FC01              \
FE17  60          RTS                    @
FE18  2000FE      JSR FE00                ^
FE1B  C952        CMP# 52                IR
FE1D  F016        BEQ 16        *FE35*   P
FE1F  C930        CMP# 30                IO
FE21  30F5        BMI F5        *FE18*   0U
FE23  C93A        CMP# 3A                I:
FE25  300B        BMI 0B        *FE32*   0
FE27  C941        CMP# 41                IA
FE29  30ED        BMI ED        *FE18*   0M
FE2B  C947        CMP# 47                IG
FE2D  10E9        BPL E9        *FE18*   I
FE2F  18          CLC
FE30  E906        SBC# 06                I
FE32  290F        AND# 0F                )
FE34  60          RTS                    @
FE35  A903        LDA# 03                )
FE37  8D00FC      STA# FC00              \
FE3A  A9B1        LDA# B1                )1
FE3C  8D00FC      STA# FC00              \
FE3F  D8          CLD                    X
FE40  78          SEI                    X
FE41  A226        LDX# 26                "&
FE43  9A          TXS
FE44  A90D        LDA# 0D                )
FE46  200BFE      JSR FE0B                ^
FE49  A90A        LDA# 0A                )
FE4B  200BFE      JSR FE0B                ^
FE4E  2000FE      JSR FE00                ^
FE51  C94C        CMP# 4C                IL
FE53  F022        BEQ 22        *FE77*   P"
FE55  C950        CMP# 50                IP
FE57  F034        BEQ 34        *FE8D*   P4
FE59  C947        CMP# 47                IG
FE5B  D0D8        BNE D8        *FE35*   PX
FE5D  AE2D01      LDX# 012D               -
FE60  9A          TXS
FE61  AE2A01      LDX# 012A               .*
FE64  AC2901      LDY# 0129               ,)
FE67  AD2E01      LDA# 012E               -.
FE6A  48          PHA                    H
FE6B  AD2F01      LDA# 012F               -/
FE6E  48          PHA                    H
FE6F  AD2C01      LDA# 012C               -,
FE72  48          PHA                    H
FE73  AD2B01      LDA# 012B               -+
FE76  40          RTI                    @
FE77  20C7FE      JSR FEC7                G^
FE7A  A203        LDX# 03                "
```

```
FE7C  A000      LDY# 00
FE7E  20B5FE    JSR FEB5              5^
FE81  A5FF      LDA-Z FF              %
FE83  91FC      STA-IY FC             \
FE85  C8        INY                   H
FE86  D0F6      BNE F6        *FE7E*  PV
FE88  E6FD      INC-Z FD              F]
FE8A  B8        CLV                   8
FE8B  50F1      BVC F1        *FE7E*  PQ
FE8D  20C7FE    JSR FEC7              G^
FE90  A000      LDY# 00
FE92  A209      LDX# 09               "
FE94  A90D      LDA# 0D               )
FE96  200BFE    JSR FE0B               ^
FE99  A90A      LDA# 0A               )
FE9B  200BFE    JSR FE0B               ^
FE9E  CA        DEX                   J
FE9F  F00B      BEQ 0B        *FEAC*  P
FEA1  20E0FE    JSR FEE0               @^
FEA4  C8        INY                   H
FEA5  D0F7      BNE F7        *FE9E*  PW
FEA7  E6FD      INC-Z FD              F]
FEA9  4C9EFE    JMP# FE9E             L^
FEAC  AD00FC    LDA# FC00             -\
FEAF  4A        LSR-A                 J
FEB0  B08E      BCS 8E        *FE40*  0
FEB2  EA        NOP                   J
FEB3  90DD      BCC DD        *FE92*  ]
FEB5  2018FE    JSR FE18               ^
FEB8  0A        ASL-A
FEB9  0A        ASL-A
FEBA  0A        ASL-A
FEBB  0A        ASL-A
FEBC  95FC      STA-ZX FC             \
FEBE  2018FE    JSR FE18               ^
FEC1  18        CLC
FEC2  75FC      ADC-ZX FC             U\
FEC4  95FC      STA-ZX FC             \
FEC6  60        RTS                   @
FEC7  A201      LDX# 01               "
FEC9  20B5FE    JSR FEB5              5^
FECC  CA        DEX                   J
FECD  20B5FE    JSR FEB5              5^
FED0  60        RTS                   @
FED1  18        CLC
FED2  6930      ADC# 30               I0
FED4  C93A      CMP# 3A               I:
FED6  B004      BCS 04        *FEDC*  0
FED8  200BFE    JSR FE0B               ^
FEDB  60        RTS                   @
FEDC  6906      ADC# 06               I
FEDE  90F8      BCC F8        *FED8*  X
FEE0  B1FC      LDA-IX FC             1\
FEE2  29F0      AND# F0               )P
FEE4  4A        LSR-A                 J
FEE5  4A        LSR-A                 J
FEE6  4A        LSR-A                 J
FEE7  4A        LSR-A                 J
FEE8  20D1FE    JSR FED1               Q^
FEEB  B1FC      LDA-IX FC             1\
FEED  290F      AND# 0F               )
```

```
FEEF  20D1FE      JSR FED1              Q^
FEF2  A920        LDA# 20               )
FEF4  200BFE      JSR FE0B               ^
FEF7  60          RTS                   @
FEF8  40          RTI                   @
FEF9  9D3001      STA-AX 0130           0
FEFC  35FE        AND-ZX FE             5^
FEFE  C001        CPY# 01               @
```

65H CD-74 HARD DISC BOOT STRAP

```
FD00  200CFD      JSR FD0C                 ]
FD03  4C18E0      JMP# E018               L@
FD06  2016FD      JSR FD16                 ]
FD09  4C18E0      JMP# E018               L@
FD0C  A900        LDA# 00                 )
FD0E  8DFFEF      STA# EFFF               O
FD11  A900        LDA# 00                 )
FD13  8DFEEF      STA# EFFE               ^O
FD16  D8          CLD                     X
FD17  A207        LDX# 07                 "
FD19  A900        LDA# 00                 )
FD1B  2C80C2      BIT# C280               ,B
FD1E  9D00C2      STA-AX C200             B
FD21  CA          DEX                     J
FD22  10F7        BPL F7          *FD1B*  W
FD24  ADFFEF      LDA# EFFF               -O
FD27  8D00C2      STA# C200               B
FD2A  ADFEEF      LDA# EFFE               -^O
FD2D  8D01C2      STA# C201               B
FD30  A910        LDA# 10                 )
FD32  2C80C2      BIT# C280               ,B
FD35  8D02C2      STA# C202               B
FD38  A900        LDA# 00                 )
FD3A  8D02C2      STA# C202               B
FD3D  2098FD      JSR FD98                 ]
FD40  A203        LDX# 03                 "
FD42  BDABFD      LDA-AX FDAB             =+]
FD45  2C80C2      BIT# C280               ,B
FD48  9D03C2      STA-AX C203             B
FD4B  CA          DEX                     J
FD4C  10F4        BPL F4          *FD42*  T
FD4E  2C80C2      BIT# C280               ,B
FD51  A980        LDA# 80                 )
FD53  8D07C2      STA# C207               B
FD56  AD07C2      LDA# C207               -B
FD59  30FB        BMI FB          *FD56*  0[
FD5B  AD12E0      LDA# E012               -@
FD5E  4DFFEF      EOR# EFFF               MO
FD61  30B3        BMI B3          *FD16*  03
FD63  AD13E0      LDA# E013               -@
FD66  4DFEEF      EOR# EFFE               M^O
FD69  D0AB        BNE AB          *FD16*  P+
FD6B  A918        LDA# 18                 )
FD6D  85FC        STA-Z FC                \
FD6F  A9E0        LDA# E0                 )@
FD71  85FD        STA-Z FD                ]
FD73  A90E        LDA# 0E                 )
FD75  85FE        STA-Z FE                ^
FD77  A900        LDA# 00                 )
FD79  AA          TAX                     *
FD7A  A8          TAY                     (
FD7B  18          CLC
FD7C  71FC        ADC-IY FC               @\
FD7E  9004        BCC 04          *FD84*
FD80  E8          INX                     H
FD81  F001        BEQ 01          *FD84*  P
FD83  18          CLC
FD84  C8          INY                     H
FD85  D0F5        BNE F5          *FD7C*  PU
FD87  E6FD        INC-Z FD                FJ
FD89  C6FE        DEC-Z FE                F^
```

```
FF00  D8        CLD                          X
FF01  A228      LDX# 28                      "(
FF03  9A        TXS
FF04  2022BF    JSR BF22                      "?
FF07  A000      LDY# 00
FF09  8C1202    STY# 0212
FF0C  8C0302    STY# 0203
FF0F  8C0502    STY# 0205
FF12  8C0602    STY# 0206
FF15  ADE0FF    LDA# FFE0                    -@
FF18  8D0002    STA# 0200
FF1B  A920      LDA# 20                      )
FF1D  9900D7    STA-AY D700                  W
FF20  9900D6    STA-AY D600                  V
FF23  9900D5    STA-AY D500                  U
FF26  9900D4    STA-AY D400                  T
FF29  9900D3    STA-AY D300                  S
FF2C  9900D2    STA-AY D200                  R
FF2F  9900D1    STA-AY D100                  Q
FF32  9900D0    STA-AY D000                  P
FF35  C8        INY                          H
FF36  D0E5      BNE E5          *FF1D*       PE
FF38  B95FFF    LDA-AY FF5F                  9*
FF3B  F006      BEQ 06          *FF43*       P
FF3D  202DBF    JSR BF2D                      -?
FF40  C8        INY                          H
FF41  D0F5      BNE F5          *FF38*       PU
FF43  20B8FF    JSR FFB8                      8
FF46  C94D      CMP# 4D                      IM
FF48  D003      BNE 03          *FF4D*       P
FF4A  4C00FE    JMP# FE00                    L^
FF4D  C957      CMP# 57                      IW
FF4F  D003      BNE 03          *FF54*       P
FF51  4C0000    JMP# 0000                    L
FF54  C943      CMP# 43                      IC
FF56  D0A8      BNE A8          *FF00*       P(
FF58  A900      LDA# 00                      )
FF5A  AA        TAX                          *
FF5B  A8        TAY                          (
FF5C  4C11BD    JMP# BD11                    L=
FF5F  43        ?                            C
FF60  2F        ?                            /
FF61  57        ?                            W
FF62  2F        ?                            /
FF63  4D203F    EOR# 3F20                    M ?
FF66  00        BRK
FF67  202DBF    JSR BF2D                      -?
FF6A  48        PHA                          H
FF6B  AD0502    LDA# 0205                    -
FF6E  F022      BEQ 22          *FF92*       P"
FF70  68        PLA                          H
FF71  2015BF    JSR BF15                      ?
FF74  C90D      CMP# 0D                      I
FF76  D01B      BNE 1B          *FF93*       P
FF78  48        PHA                          H
FF79  8A        TXA
FF7A  48        PHA                          H
FF7B  A20A      LDX# 0A                      "
FF7D  A900      LDA# 00                      )
FF7F  2015BF    JSR BF15                      ?
FF82  CA        DEX                          J
```

65H CD-74 HARD DISC BOOT STRAP

```
FD8B  D0EF      BNE EF        *FD7C*    PO
FD8D  CD18EE    CMP# EE18               MN
FD90  D084      BNE 84        *FD16*    P
FD92  EC19EE    CPX# EE19               LN
FD95  D0F9      BNE F9        *FD90*    PY
FD97  60        RTS                     @
FD98  AD02C2    LDA# C202               -B
FD9B  C9D9      CMP# D9                 IY
FD9D  F00B      BEQ 0B        *FDAA*    P
FD9F  29C4      AND# C4                 )D
FDA1  C9C4      CMP# C4                 ID
FDA3  D0F3      BNE F3        *FD98*    PS
FDA5  68        PLA                     H
FDA6  68        PLA                     H
FDA7  4C16FD    JMP# FD16               LJ
FDAA  60        RTS                     @
FDAB  13        ?
FDAC  00        BRK
FDAD  2507      AND-Z 07                %
FDAF  2016FD    JSR FD16                 J
FDB2  6CFCFE    JMP-IN FEFC             L\^
FDB5  2424      BIT-Z 24      *FDDB*    $$
FDB7  2424      BIT-Z 24      *FDDD*    $$
FDB9  2424      BIT-Z 24      *FDDF*    $$
FDBB  2424      BIT-Z 24      *FDE1*    $$
FDBD  2424      BIT-Z 24      *FDE3*    $$
FDBF  2424      BIT-Z 24      *FDE5*    $$
FDC1  2424      BIT-Z 24      *FDE7*    $$
FDC3  2424      BIT-Z 24      *FDE9*    $$
FDC5  2424      BIT-Z 24      *FDEB*    $$
FDC7  2424      BIT-Z 24      *FDED*    $$
FDC9  2424      BIT-Z 24      *FDEF*    $$
FDCB  2424      BIT-Z 24      *FDF1*    $$
FDCD  2424      BIT-Z 24      *FDF3*    $$
FDCF  2424      BIT-Z 24      *FDF5*    $$
FDD1  2424      BIT-Z 24      *FDF7*    $$
FDD3  2424      BIT-Z 24      *FDF9*    $$
FDD5  2424      BIT-Z 24      *FDFB*    $$
FDD7  2424      BIT-Z 24      *FDFD*    $$
FDD9  2424      BIT-Z 24      *FDFF*    $$
FDDB  2424      BIT-Z 24      *FE01*    $$
FDDD  2424      BIT-Z 24      *FE03*    $$
FDDF  2424      BIT-Z 24      *FE05*    $$
FDE1  2424      BIT-Z 24      *FE07*    $$
FDE3  2424      BIT-Z 24      *FE09*    $$
FDE5  2424      BIT-Z 24      *FE0B*    $$
FDE7  2424      BIT-Z 24      *FE0D*    $$
FDE9  2424      BIT-Z 24      *FE0F*    $$
FDEB  2424      BIT-Z 24      *FE11*    $$
FDED  2424      BIT-Z 24      *FE13*    $$
FDEF  2424      BIT-Z 24      *FE15*    $$
FDF1  2424      BIT-Z 24      *FE17*    $$
FDF3  2424      BIT-Z 24      *FE19*    $$
FDF5  2424      BIT-Z 24      *FE1B*    $$
FDF7  2424      BIT-Z 24      *FE1D*    $$
FDF9  2424      BIT-Z 24      *FE1F*    $$
FDFB  2424      BIT-Z 24      *FE21*    $$
FDFD  2424      BIT-Z 24      *FE23*    $$
FDFF  24AD      BIT-Z AD      *FDAE*    $-
```

```
FF83  D0FA      BNE FA       *FF7F*    PZ
FF85  68        PLA                    H
FF86  AA        TAX                    *
FF87  68        PLA                    H
FF88  60        RTS                    @
FF89  48        PHA                    H
FF8A  CE0302    DEC# 0203              N
FF8D  A900      LDA# 00                )
FF8F  8D0502    STA# 0205
FF92  68        PLA                    H
FF93  60        RTS                    @
FF94  48        PHA                    H
FF95  A901      LDA# 01                )
FF97  D0F6      BNE F6       *FF8F*    PV
FF99  AD1202    LDA# 0212              -
FF9C  D019      BNE 19       *FFB7*    P
FF9E  A901      LDA# 01                )
FFA0  8D00DF    STA# DF00              ←
FFA3  2C00DF    BIT# DF00              ,←
FFA6  500F      BVC 0F       *FFB7*    P
FFA8  A904      LDA# 04                )
FFAA  8D00DF    STA# DF00              ←
FFAD  2C00DF    BIT# DF00              ,←
FFB0  5005      BVC 05       *FFB7*    P
FFB2  A903      LDA# 03                )
FFB4  4C36A6    JMP# A636              L6&
FFB7  60        RTS                    @
FFB8  2C0302    BIT# 0203              ,
FFBB  1019      BPL 19       *FFD6*
FFBD  A902      LDA# 02                )
FFBF  8D00DF    STA# DF00              ←
FFC2  A910      LDA# 10                )
FFC4  2C00DF    BIT# DF00              ,←
FFC7  D00A      BNE 0A       *FFD3*    P
FFC9  AD00FC    LDA# FC00              -\
FFCC  4A        LSR-A                  J
FFCD  90EE      BCC EE       *FFBD*    N
FFCF  AD01FC    LDA# FC01              -\
FFD2  60        RTS                    @
FFD3  EE0302    INC# 0203              N
FFD6  4CEDFE    JMP# FEED              LM^
FFD9  00        BRK
FFDA  00        BRK
FFDB  00        BRK
FFDC  00        BRK
FFDD  00        BRK
FFDE  00        BRK
FFDF  00        BRK
FFE0  40        RTI                    @
FFE1  3F        ?                      ?
FFE2  0100      ORA-IX 00
FFE4  03        ?
FFE5  FF        ?
FFE6  3F        ?                      ?
FFE7  00        BRK
FFE8  03        ?
FFE9  FF        ?
FFEA  3F        ?                      ?
FFEB  4CB8FF    JMP# FFB8              L8
FFEE  4C67FF    JMP# FF67              LG
FFF1  4C99FF    JMP# FF99              L
```

```
65VB 7 6 ROM BASIC SUPPORT FOR 540 VIDEO W. POLLED KEYBOARD

FFF4  4C89FF      JMP# FF89               L
FFF7  4C94FF      JMP# FF94               L
FFFA  3001        BMI 01        *FFFD*    0
FFFC  00          BRK
FFFD  FF          ?
FFFE  C001        CPY# 01                 @
```

```
FE00  A228      LDX# 28                 "(
FE02  9A        TXS
FE03  D8        CLD                     X
FE04  AD06FB    LDA# FB06               -[
FE07  A9FF      LDA# FF                 )
FE09  8D05FB    STA# FB05               [
FE0C  A2D8      LDX# D8                 "X
FE0E  A9D0      LDA# D0                 )P
FE10  85FF      STA-Z FF
FE12  A900      LDA# 00                 )
FE14  85FE      STA-Z FE                ^
FE16  85FB      STA-Z FB                [
FE18  A8        TAY                     (
FE19  A920      LDA# 20                 )
FE1B  91FE      STA-IY FE               ^
FE1D  C8        INY                     H
FE1E  D0FB      BNE FB          #FE1B#  P[
FE20  E6FF      INC-Z FF                F
FE22  E4FF      CPX-Z FF                D
FE24  D0F5      BNE F5          #FE1B#  PU
FE26  84FF      STY-Z FF
FE28  F019      BEQ 19          #FE43#  P
FE2A  20E9FE    JSR FEE9                 I^
FE2D  C92F      CMP# 2F                 I/
FE2F  F01E      BEQ 1E          #FE4F#  P
FE31  C947      CMP# 47                 IG
FE33  F017      BEQ 17          #FE4C#  P
FE35  C94C      CMP# 4C                 IL
FE37  F043      BEQ 43          #FE7C#  PC
FE39  2093FE    JSR FE93                 ^
FE3C  30EC      BMI EC          #FE2A#  0L
FE3E  A202      LDX# 02                 "
FE40  20DAFE    JSR FEDA                 Z^
FE43  B1FE      LDA-IX FE               1^
FE45  85FC      STA-Z FC                \
FE47  20ACFE    JSR FEAC                 ,^
FE4A  D0DE      BNE DE          #FE2A#  P^
FE4C  6CFE00    JMP-IN 00FE             L^
FE4F  20E9FE    JSR FEE9                 I^
FE52  C92E      CMP# 2E                 I.
FE54  F0D4      BEQ D4          #FE2A#  PT
FE56  C90D      CMP# 0D                 I
FE58  D00F      BNE 0F          #FE69#  P
FE5A  E6FE      INC-Z FE                F^
FE5C  D002      BNE 02          #FE60#  P
FE5E  E6FF      INC-Z FF                F
FE60  A000      LDY# 00
FE62  B1FE      LDA-IX FE               1^
FE64  85FC      STA-Z FC                \
FE66  4C77FE    JMP# FE77               LW^
FE69  2093FE    JSR FE93                 ^
FE6C  30E1      BMI E1          #FE4F#  0A
FE6E  A200      LDX# 00                 "
FE70  20DAFE    JSR FEDA                 Z^
FE73  A5FC      LDA-Z FC                %\
FE75  91FE      STA-IY FE               ^
FE77  20ACFE    JSR FEAC                 ,^
FE7A  D0D3      BNE D3          #FE4F#  PS
FE7C  85FB      STA-Z FB                [
FE7E  F0CF      BEQ CF          #FE4F#  PO
FE80  AD00FC    LDA# FC00               -\
```

```
65VK MONITOR FOR 540 VIDEO WITH POLLED KEYBOARD

FE83  4A        LSR-A                   J
FE84  90FA      BCC FA        *FE80*    Z
FE86  AD01FC    LDA# FC01               -\
FE89  EA        NOP                     J
FE8A  EA        NOP                     J
FE8B  EA        NOP                     J
FE8C  297F      AND# 7F                 )
FE8E  60        RTS                     @
FE8F  00        BRK
FE90  00        BRK
FE91  00        BRK
FE92  00        BRK
FE93  C930      CMP# 30                 IO
FE95  3012      BMI 12        *FEA9*    0
FE97  C93A      CMP# 3A                 I:
FE99  300B      BMI 0B        *FEA6*    0
FE9B  C941      CMP# 41                 IA
FE9D  300A      BMI 0A        *FEA9*    0
FE9F  C947      CMP# 47                 IG
FEA1  1006      BPL 06        *FEA9*
FEA3  38        SEC                     8
FEA4  E907      SBC# 07                 I
FEA6  290F      AND# 0F                 )
FEA8  60        RTS                     @
FEA9  A980      LDA# 80                 )
FEAB  60        RTS                     @
FEAC  A203      LDX# 03                 "
FEAE  A000      LDY# 00
FEB0  B5FC      LDA-ZX FC               5\
FEB2  4A        LSR-A                   J
FEB3  4A        LSR-A                   J
FEB4  4A        LSR-A                   J
FEB5  4A        LSR-A                   J
FEB6  20CAFE    JSR FECA                 J^
FEB9  B5FC      LDA-ZX FC               5\
FEBB  20CAFE    JSR FECA                 J^
FEBE  CA        DEX                     J
FEBF  10EF      BPL EF        *FEB0*    0
FEC1  A920      LDA# 20                 )
FEC3  8DCAD0    STA# D0CA               JP
FEC6  8DCBD0    STA# D0CB               KP
FEC9  60        RTS                     @
FECA  290F      AND# 0F                 )
FECC  0930      ORA# 30                 0
FECE  C93A      CMP# 3A                 I:
FED0  3003      BMI 03        *FED5*    0
FED2  18        CLC
FED3  6907      ADC# 07                 I
FED5  99C6D0    STA-AY D0C6             FP
FED8  C8        INY                     H
FED9  60        RTS                     @
FEDA  A004      LDY# 04
FEDC  0A        ASL-A
FEDD  0A        ASL-A
FEDE  0A        ASL-A
FEDF  0A        ASL-A
FEE0  2A        ROL-A                   *
FEE1  36FC      ROL-ZX FC               6\
FEE3  36FD      ROL-ZX FD               6]
FEE5  88        DEY
FEE6  D0F8      BNE F8        *FEE0*    PX
```

```
65VK MONITOR FOR 540 VIDEO WITH POLLED KEYBOARD

FEE8  60        RTS                          @
FEE9  A5FB      LDA-Z FB                     %[
FEEB  D091      BNE 91        *FE7E*         P
FEED  4C00FD    JMP# FD00                    L]
FEF0  A9FF      LDA# FF                      )
FEF2  8D00DF    STA# DF00                    ←
FEF5  AD00DF    LDA# DF00                    -←
FEF8  60        RTS                          @
FEF9  EA        NOP                          J
FEFA  3001      BMI 01        *FEFD*         0
FEFC  00        BRK
FEFD  FEC001    INC-AX 01C0                  ^@
```

```
65K POLLED KEYBOARD POLLING ROUTINE

FD00  8A        TXA
FD01  48        PHA                          H
FD02  98        TYA
FD03  48        PHA                          H
FD04  A901      LDA# 01                      )
FD06  8D00DF    STA# DF00                    +
FD09  AE00DF    LDX# DF00                     +
FD0C  D005      BNE 05        *FD13*         P
FD0E  0A        ASL-A
FD0F  D0F5      BNE F5        *FD06*         PU
FD11  F053      BEQ 53        *FD66*         PS
FD13  4A        LSR-A                        J
FD14  9009      BCC 09        *FD1F*
FD16  2A        ROL-A                        *
FD17  E021      CPX# 21                      @!
FD19  D0F3      BNE F3        *FD0E*         PS
FD1B  A91B      LDA# 1B                      )
FD1D  D021      BNE 21        *FD40*         P!
FD1F  20C8FD    JSR FDC8                      HJ
FD22  98        TYA
FD23  8D1302    STA# 0213
FD26  0A        ASL-A
FD27  0A        ASL-A
FD28  0A        ASL-A
FD29  38        SEC                          8
FD2A  ED1302    SBC# 0213                    M
FD2D  8D1302    STA# 0213
FD30  8A        TXA
FD31  4A        LSR-A                        J
FD32  20C8FD    JSR FDC8                      HJ
FD35  D02F      BNE 2F        *FD66*         P/
FD37  18        CLC
FD38  98        TYA
FD39  6D1302    ADC# 0213                    M
FD3C  A8        TAY                          (
FD3D  B9CFFD    LDA-AY FDCF                  90J
FD40  CD1302    CMP# 0213                    M
FD43  D026      BNE 26        *FD6B*         P&
FD45  CE1402    DEC# 0214                    N
FD48  F02B      BEQ 2B        *FD75*         P+
FD4A  A005      LDY# 05
FD4C  A2C8      LDX# C8                      "H
FD4E  CA        DEX                          J
FD4F  D0FD      BNE FD        *FD4E*         PJ
FD51  88        DEY
FD52  D0F8      BNE F8        *FD4C*         PX
FD54  F0AE      BEQ AE        *FD04*         P.
FD56  C901      CMP# 01                      I
FD58  F035      BEQ 35        *FD8F*         P5
FD5A  A000      LDY# 00
FD5C  C902      CMP# 02                      I
FD5E  F047      BEQ 47        *FDA7*         PG
FD60  A0C0      LDY# C0                       @
FD62  C920      CMP# 20                      I
FD64  F041      BEQ 41        *FDA7*         PA
FD66  A900      LDA# 00                      )
FD68  8D1602    STA# 0216
FD6B  8D1502    STA# 0215
FD6E  A902      LDA# 02                      )
FD70  8D1402    STA# 0214
FD73  D08F      BNE 8F        *FD04*         P
```

```
FD75  A296      LDX# 96                 "
FD77  CD1602    CMP# 0216               M
FD7A  D002      BNE 02        *FD7E*    P
FD7C  A214      LDX# 14                 "
FD7E  8E1402    STX# 0214
FD81  8D1602    STA# 0216
FD84  A901      LDA# 01                 )
FD86  8D00DF    STA# DF00               ←
FD89  AD00DF    LDA# DF00               -←
FD8C  4A        LSR-A                   J
FD8D  9033      BCC 33        *FDC2*    3
FD8F  AA        TAX                     *
FD90  2903      AND# 03                 )
FD92  F00B      BEQ 0B        *FD9F*    P
FD94  A010      LDY# 10
FD96  AD1502    LDA# 0215               -
FD99  100C      BPL 0C        *FDA7*
FD9B  A0F0      LDY# F0                  P
FD9D  D008      BNE 08        *FDA7*    P
FD9F  A000      LDY# 00
FDA1  E020      CPX# 20                 @
FDA3  D002      BNE 02        *FDA7*    P
FDA5  A0C0      LDY# C0                  @
FDA7  AD1502    LDA# 0215               -
FDAA  297F      AND# 7F                 )
FDAC  C920      CMP# 20                 I
FDAE  F007      BEQ 07        *FDB7*    P
FDB0  8C1302    STY# 0213
FDB3  18        CLC
FDB4  6D1302    ADC# 0213               M
FDB7  8D1302    STA# 0213
FDBA  68        PLA                     H
FDBB  A8        TAY                     (
FDBC  68        PLA                     H
FDBD  AA        TAX                     *
FDBE  AD1302    LDA# 0213               -
FDC1  60        RTS                     @
FDC2  D092      BNE 92        *FD56*    P
FDC4  A020      LDY# 20
FDC6  D0DF      BNE DF        *FDA7*    P←
FDC8  A008      LDY# 08
FDCA  88        DEY
FDCB  0A        ASL-A
FDCC  90FC      BCC FC        *FDCA*    \
FDCE  60        RTS                     @
FDCF  D0BB      BNE BB        *FD8C*    P;
FDD1  2F        ?                       /
FDD2  205A41    JSR 415A                 ZA
FDD5  512C      EOR-IY 2C               Q,
FDD7  4D4E42    EOR# 424E               MNB
FDDA  5643      LSR-ZX 43               VC
FDDC  58        CLI                     X
FDDD  4B        ?                       K
FDDE  4A        LSR-A                   J
FDDF  48        PHA                     H
FDE0  47        ?                       G
FDE1  4644      LSR-Z 44                FD
FDE3  53        ?                       S
FDE4  4955      EOR# 55                 IU
FDE6  595452    EOR-AY 5254             YTR
FDE9  4357      EOR-Z 57                EW
```

```
65K POLLED KEYBOARD POLLING ROUTINE

FDEB  00        BRK
FDEC  00        BRK
FDED  0D0A4F    ORA# 4F0A           O
FDF0  4C2E00    JMP# 002E           L.
FDF3  FF        ?
FDF4  2DBA30    AND# 30BA           -:0
FDF7  B9B8B7    LDA-AY B7B8         987
FDFA  B6B5      LDX-ZY B5           65
FDFC  B4B3      LDY-ZX B3           43
FDFE  B2        ?                   2
FDFF  B1AD      LDA-IX AD           1-
```

```
FF00  D8        CLD                          X
FF01  A228      LDX# 28                      "(
FF03  9A        TXS
FF04  2022BF    JSR BF22                      "?
FF07  20FEBE    JSR BEFE                      ^>
FF0A  A000      LDY# 00
FF0C  98        TYA
FF0D  A20E      LDX# 0E                      "
FF0F  9D0302    STA-AX 0203
FF12  CA        DEX                          J
FF13  10FA      BPL FA          *FF0F*       Z
FF15  B9B7FF    LDA-AY FFB7                  97
FF18  3006      BMI 06          *FF20*       0
FF1A  2015BF    JSR BF15                      ?
FF1D  C8        INY                          H
FF1E  D0F5      BNE F5          *FF15*       PU
FF20  2007BF    JSR BF07                      ?
FF23  C94D      CMP# 4D                      IM
FF25  D003      BNE 03          *FF2A*       P
FF27  4C40FE    JMP# FE40                    L@^
FF2A  C957      CMP# 57                      IW
FF2C  D003      BNE 03          *FF31*       P
FF2E  4C0000    JMP# 0000                    L
FF31  C943      CMP# 43                      IC
FF33  D0CB      BNE CB          *FF00*       PK
FF35  A900      LDA# 00                      )
FF37  AA        TAX                          *
FF38  A8        TAY                          (
FF39  4C11BD    JMP# BD11                    L=
FF3C  0A        ASL-A
FF3D  4F        ?                            O
FF3E  4B        ?                            K
FF3F  0DF0BE    ORA# BEF0                    P>
FF42  48        PHA                          H
FF43  8E0402    STX# 0204
FF46  AD1002    LDA# 0210                    -
FF49  D022      BNE 22          *FF6D*       P"
FF4B  68        PLA                          H
FF4C  2015BF    JSR BF15                      ?
FF4F  48        PHA                          H
FF50  AD0502    LDA# 0205                    -
FF53  F013      BEQ 13          *FF68*       P
FF55  68        PLA                          H
FF56  20F3BE    JSR BEF3                      S>
FF59  C90D      CMP# 0D                      I
FF5B  D00C      BNE 0C          *FF69*       P
FF5D  48        PHA                          H
FF5E  A20A      LDX# 0A                      "
FF60  A900      LDA# 00                      )
FF62  20F3BE    JSR BEF3                      S>
FF65  CA        DEX                          J
FF66  D0FA      BNE FA          *FF62*       PZ
FF68  68        PLA                          H
FF69  AE0402    LDX# 0204
FF6C  60        RTS                          @
FF6D  68        PLA                          H
FF6E  48        PHA                          H
FF6F  AE1102    LDX# 0211
FF72  DD3CFF    CMP-AX FF3C                  ]<
FF75  D008      BNE 08          *FF7F*       P
FF77  E8        INX                          H
```

```
FF78  E004      CPX# 04              @
FF7A  D005      BNE 05       *FF81*  P
FF7C  20AEFF    JSR FFAE
FF7F  A200      LDX# 00              "
FF81  8E1102    STX# 0211
FF84  4C50FF    JMP# FF50            LP
FF87  AD00FC    LDA# FC00            -\
FF8A  4A        LSR-A                J
FF8B  9013      BCC 13       *FFA0*
FF8D  A900      LDA# 00              )
FF8F  8D0302    STA# 0203
FF92  AD01FC    LDA# FC01            -\
FF95  F0F0      BEQ F0       *FF87*  PP
FF97  297F      AND# 7F              )
FF99  C905      CMP# 05              I
FF9B  D0CF      BNE CF       *FF6C*  PO
FF9D  20AEFF    JSR FFAE
FFA0  AD0302    LDA# 0203            -
FFA3  F0E2      BEQ E2       *FF87*  PB
FFA5  AD05FB    LDA# FB05            -[
FFA8  4A        LSR-A                J
FFA9  90DC      BCC DC       *FF87*  \
FFAB  4CEABE    JMP# BEEA            LJ>
FFAE  AD1002    LDA# 0210            -
FFB1  49FF      EOR# FF              I
FFB3  8D1002    STA# 0210
FFB6  60        RTS                  @
FFB7  43        ?                    C
FFB8  2F        ?                    /
FFB9  57        ?                    W
FFBA  2F        ?                    /
FFBB  4D3FAD    EOR# AD3F            M?-
FFBE  00        BRK
FFBF  FC        ?                    \
FFC0  4A        LSR-A                J
FFC1  9003      BCC 03       *FFC6*
FFC3  4C33A6    JMP# A633            L3&
FFC6  4C28A6    JMP# A628            L(&
FFC9  48        PHA                  H
FFCA  A901      LDA# 01              )
FFCC  D008      BNE 08       *FFD6*  P
FFCE  48        PHA                  H
FFCF  A901      LDA# 01              )
FFD1  8D0302    STA# 0203
FFD4  A900      LDA# 00              )
FFD6  8D0502    STA# 0205
FFD9  68        PLA                  H
FFDA  20FEBE    JSR BEFE               ^>
FFDD  4C19A3    JMP# A319            L#
FFE0  64        ?                    D
FFE1  18        CLC
FFE2  00        BRK
FFE3  00        BRK
FFE4  03        ?
FFE5  FF        ?
FFE6  3F        ?                    ?
FFE7  00        BRK
FFE8  03        ?
FFE9  FF        ?
FFEA  3F        ?                    ?
FFEB  4C87FF    JMP# FF87            L
```

```
FFEE  4C42FF      JMP# FF42              LB
FFF1  4CBDFF      JMP# FFBD              L=
FFF4  4CCEFF      JMP# FFCE              LN
FFF7  4CC9FF      JMP# FFC9              LI
FFFA  3001        BMI 01        *FFFD*   0
FFFC  00          BRK
FFFD  FF          ?
FFFE  C001        CPY# 01                @
```

```
68A2 SERIAL MONITOR FOR 6800 MICRO PROCESSOR

FE00  7E FF D3 47 24 FA B6 FC 01 84 7F 81 7F 27 F1 7E
FE10  FF 88 8D EC 81 52 27 13 81 30 2B F6 81 39 2F 0A
FE20  81 41 2B EE 81 46 2E EA 80 07 39 7E FF A8 8D 07
FE30  8D 13 A7 00 08 20 F9 8D 0C B7 1F 34 8D 07 B7 1F
FE40  35 FE 1F 34 39 8D CB 48 48 48 48 16 8D C4 84 0F
FE50  1B 39 00 00 8D E1 86 0D 8D 2E 86 0A 8D 2A 8D 17
FE60  8D 15 8D 13 8D 11 8D 0F 8D 0D 8D 0B 8D 09 B6 FC
FE70  00 47 24 E2 7E FF B2 8D 26 39 44 44 44 44 84 0F
FE80  8B 30 81 39 23 02 8B 07 37 F6 FC 00 57 57 24 F9
FE90  B7 FC 01 33 39 A6 00 8D E1 A6 00 8D E1 08 39 8D
FEA0  F4 86 20 20 E3 00 00 00 86 03 B7 FC 00 86 B1 B7
FEB0  FC 00 8E 1F 28 86 0D 8D CF 86 0A 8D CB BD FF 00
FEC0  16 8D DE C1 4C 26 03 7E FF 2E C1 50 27 86 C1 47
FED0  26 E0 3B B6 1F DF 27 03 7E 1F B0 B6 FC 00 47 24
FEE0  FA 7E FF 06 00 00 00 7F 1F DF C6 00 F7 F7 01 F7
FEF0  F7 00 C6 04 F7 F7 01 7E FF A8 FF A8 1F E0 FF E7
```

Conventional Typewriter

This program provides a means of using the OSI-65V when interfaced to a printer to be used as a conventional typewriter and also modify the text for a data file.

Consider the program as having two parts or functions. The first part allows you to type text in much the same way you would on a typewriter, that is, without formatting PRINT instructions in BASIC. You need type only what you want to be printed.

After clearing the video screen, the program starts by asking the user "CHARACTER WIDTH?". This tells the program how many characters per line to limit. The maximum width would be determined by your printer or video display format. Next the computer displays the line number of the text which you are typing. An up-arrow appears on the screen (via a POKE instruction in the program) indicating where your line must terminate. This becomes a very useful feature. In the event that the line-width is exceeded, an OVER-WIDTH error message comes up and you are free to rewrite the line. When the text input has been completed, input a right-arrow, which signals the program to jump to its next section.The correct format is then printed out on your display, following which the screen is cleared. Next the program lists three options. The first asks "PRINT DATA" -- inputing a 1 causes the printer to turn on via a POKE instruction. The correct text is then printed in correct format and the printer is turned back off via another POKE instruction. The program now displays the option list again.

The second option asks "FILE DATA" -- inputing a 2 displays the message "SET RECORDER".

At this point a delay loop in the program allows you enough time to turn on the recorder in the record mode. When the loop times out, each line of the text is preceded by "100X DATA". Output to tape and display on screen are as in the following condensed example:

```
1000 DATA Now is the
1001 DATA time for all
1002 DATA good men
```

The line numbers are incremented, the word "DATA" is inserted, and the text follows.

When inputing back from the tape, any program could be written to use the data. The usefulness of the program should be readily apparent.

The third option asks "ADD DATA" -- inputing a 3 allows you to continue with the text from the point where you left off.

Note that some of the added features provided in this program via the POKE commands (e.g., screen erase and printer on-off at programmed points) cannot be appreciated in the hard copy printout.

General notes: a space inputed instead of characters yields a blank line. The text is limited to 256 lines of memory capability. The program will not recognize commas. If you type a line containing a comma, an error message "EXTRA IGNORED" will appear in the succeeding line.

This same program can also be used on OS-65A (serial-based systems) with the following restrictions:

No up-arrow appears on the screen as a prompter to indicate where a line should end.

The screen is not cleared at the start of each operation (on video-based system the clearing of the screen occurs as a result of lines 60 and 300; on serial-based systems these lines cause merely a triple-carriage return-linefeed).

```
 10 PRINT
 20 PRINT "PROGRAM BY
 25 PRINT "GARY SMITH"
 30 PRINT "4322 WATTERSON ST.
 35 PRINT "CINCINNATI OH 45227"
 50 GOSUB 220
 60 PRINT:PRINT:PRINT
 70 INPUT "CHARACTERS PER LINE";D
 80 I=256
 90 DIMA$(I)
 100 PRINT
 110 FOR I=1TO256
 120 PRINTI
 130 IFD=>25ANDD=<49THENPOKE54149+(D-25),94
 140 IFD=>49ANDD=<70THENPOKE54181+(D-50),94
 150 IFD=<22THENPOKE54149+D,94
 160 IFD=24THENPOKE54148,94
 170 IFD=23THENPOKE54172,94
 180 INPUTA$(I)
 190 IFLEN ((A$(Y)))>DTHENPRINT"OVERWIDTH":I=I-1
 200 IFA$(I)=">"THEN260
 210 NEXTI
 220 FORC=53348TO54268
 230 POKEC,32:NEXTC
 240 RETURN
 250 POKE 64258,1
 260 GOSUB 220
 270 FORJ=1TOI-1
 280 PRINTA$(J)
 290 NEXTJ
 300 PRINT:PRINT:PRINT
 310 POKE 64258,0
 320 PRINTI-1;"DATA LINES"
 330 PRINT
 340 PRINT "INPUT OPTION..."
 350 PRINT
 360 PRINT"  1=PRINT DATA"
 370 PRINT"  2=FILE DATA"
 380 PRINT"  3=ADD MORE DATA"
 390 INPUTE
 400 PRINT
 410 IFE=3THENI=I-1:GOTO210
 420 IFE=1THENPOKE64258,1:GOTO260
 430 IFE<>1ANDE<>2THENPRINT"BAD INPUT ?":PRINT:GOTO340
 440 PRINT"SET RECORDER"
 450 NULL10
 460 FORL=1TO1000:NEXT
 470 GOSUB 220
 480 PRINT "1000 DATA ";I-1
 490 FORK=1TOI-1
 500 A=1000+K
 510 PRINTA;"DATA ";A$(K)
 520 NEXTK
 530 NULL0
 540 END
OK
```

Hex Conversion Table

The following tables contain direct conversions between decimal and hexadecimal integers. The tables cover a range between $0–4095_{10}$. To convert hexadecimal integers outside of this range to decimal integers, add a displacement from the following list of displacements.

Hexadecimal	Decimal	Hexadecimal	Decimal
01 000	4 096	1F 000	126 976
02 000	8 192	20 000	131 072
03 000	12 288	30 000	196 608
04 000	16 384	40 000	262 144
05 000	20 480	50 000	327 680
06 000	24 576	60 000	393 216
07 000	28 672	70 000	458 752
08 000	32 768	80 000	524 288
09 000	36 864	90 000	589 824
0A 000	40 960	A0 000	655 360
0B 000	45 056	B0 000	720 896
0C 000	49 152	C0 000	786 432
0D 000	53 248	D0 000	851 968
0E 000	57 344	E0 000	917 504
0F 000	61 440	F0 000	983 040
10 000	65 536	100 000	1 048 576
11 000	69 632	200 000	2 097 152
12 000	73 728	300 000	3 145 728
13 000	77 824	400 000	4 194 304
14 000	81 920	500 000	5 242 880
15 000	86 016	600 000	6 291 456
16 000	90 112	700 000	7 340 032
17 000	94 208	800 000	8 388 608
18 000	98 304	900 000	9 437 184
19 000	102 400	A00 000	10 485 760
1A 000	106 496	B00 000	11 534 336
1B 000	110 592	C00 000	12 582 912
1C 000	114 688	D00 000	13 631 488
1D 000	118 784	E00 000	14 680 064
1E 000	122 880	F00 000	15 728 640

For example: To convert $23A7_{16}$ to decimal,

1) Find $3A7_{16}$ = 935_{10} from table
2) Add 2000_{16} = 8192_{10} from list

$$\begin{array}{r} 8192 \\ +\ 935 \\ \hline 9127 \end{array}$$

thus $23A7_{16}$ = 9127_{10}

To convert decimal integers outside of this range to hexadecimal integers, use the following list of displacements.

Decimal	Hexidecimal	Decimal	Hexidecimal
1 000	3E8	2 000 000	1E8 480
2 000	7D0	3 000 000	2DC 6C0
3 000	BB8	4 000 000	3D0 900
4 000	FA0	5 000 000	4C4 B40
5 000	1 388	6 000 000	5B8 D80
6 000	1 770	7 000 000	6AC E60
7 000	1 B58	8 000 000	7A1 200
8 000	1 F40	9 000 000	895 440
9 000	2 328	10 000 000	989 680
10 000	2 710	20 000 000	1 312 D00
20 000	4 E20	30 000 000	1 C9C 380
30 000	7 530	40 000 000	2 625 A00
40 000	9 C40	50 000 000	2 FAF 080
50 000	C 350	60 000 000	3 938 700
60 000	E A60	70 000 000	4 2C1 D80
70 000	11 170	80 000 000	4 C4B 400
80 000	13 880	90 000 000	5 5D4 A80
90 000	15 F90	100 000 000	5 F5E 100
100 000	18 6A0	200 000 000	B EBC 200
200 000	30 D40	300 000 000	11 E1A 300
300 000	49 3E0	400 000 000	17 D78 400
400 000	61 A80	500 000 000	1D CD6 500
500 000	7A 120	600 000 000	23 C34 600
600 000	92 7C0	700 000 000	29 B92 700
700 000	AA E60	800 000 000	2F AF0 800
800 000	C3 500	900 000 000	35 A4E 900
900 000	DB 8A0	1 000 000 000	3B 9AC A00
1 000 000	F4 240		

For example: To convert $41,236_{10}$ to hexadecimal

1) Find 1236_{10} = $4D4_{16}$ from table

2) Add $40,000_{10}$ = $9C40_{16}$ from above list

$$\begin{array}{r} 9C40 \\ +\ 4D4 \\ \hline A114 \end{array}$$

thus $41,236_{10}$ = $A114_{16}$

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
010	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
020	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
030	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
040	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
050	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
060	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
070	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
080	0126	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
090	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A0	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B0	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C0	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D0	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E0	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F0	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
100	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
110	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
120	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
130	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
140	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
150	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
160	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
170	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	038.	0383
180	0384	0385	0366	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
190	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A0	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B0	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C0	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D0	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E0	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F0	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
200	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
210	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
220	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
230	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
240	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
250	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
260	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
270	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
280	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
290	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A0	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B0	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C0	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D0	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E0	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F0	0752	0753	0754	0755	0756	0757	0758	0759	0760	0781	0782	0763	0764	0765	0766	0767

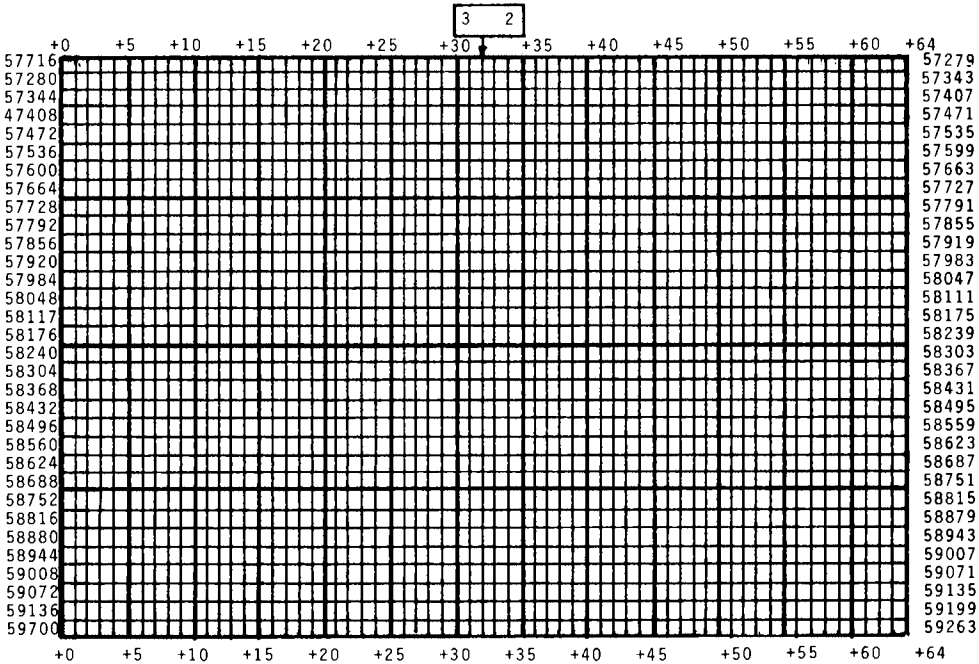
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
300	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
310	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
320	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
330	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
340	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
350	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
360	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
370	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
380	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
390	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A0	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B0	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C0	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D0	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E0	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F0	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023
400	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
410	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
420	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
430	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
440	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
450	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
460	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
470	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
480	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
490	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A0	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
480	1200	1201	1202	1203	1204	1205	1208	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C0	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D0	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E0	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F0	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
500	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
510	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
520	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
530	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
540	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
550	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
560	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
570	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
580	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
590	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A0	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B0	1456	1467	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C0	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D0	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E0	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F0	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
600	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
610	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
620	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
630	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
640	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
650	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
660	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
670	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
680	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
690	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A0	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B0	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C0	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D0	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E0	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F0	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
700	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
710	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
720	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
730	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
740	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
750	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
760	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
770	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
780	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
790	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A0	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B0	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C0	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D0	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E0	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F0	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
800	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
810	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
820	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
830	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
840	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
850	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
860	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
870	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
880	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
890	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A0	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B0	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C0	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D0	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E0	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F0	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
900	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
910	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
920	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
930	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
940	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
950	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
960	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
970	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
980	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
990	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A0	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B0	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	3494	2495
9C0	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D0	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E0	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F0	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559
A00	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A10	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A20	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A30	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A40	2824	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A50	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A60	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A70	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A80	2688	2689	2690	2691	2692	2693	2894	2695	2696	2697	2698	2699	2700	2701	2702	2703
A90	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA0	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB0	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD0	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B00	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B10	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B20	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B30	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B40	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B50	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B60	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B70	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B80	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B90	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA0	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB0	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC0	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD0	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE0	3040	3041	3042	3043	3044	3045	3046	4047	3048	3049	3050	3051	3052	3053	3054	3055
BF0	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C00	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C10	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C20	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C30	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C40	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C50	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C60	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C70	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C80	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C90	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA0	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB0	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC0	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD0	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE0	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF0	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D00	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D10	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D20	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D30	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D40	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D50	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D60	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D70	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D80	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D90	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA0	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB0	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC0	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
CC0	3536	3537	3538	3539	3540	3541	3542	3643	3544	3545	3546	3547	3548	3549	3550	3551
DE0	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF0	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E00	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E10	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E20	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E30	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E40	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E50	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E60	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E70	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E80	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E90	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	4743
EA0	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB0	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC0	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED0	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE0	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF0	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
F00	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F10	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F20	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F30	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F40	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F50	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F60	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F70	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F80	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F90	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA0	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB0	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC0	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD0	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE0	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF0	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095



POKE 56832,0	CHARACTERS=32	SOUND OFF	B + W
POKE 56832,1	CHARACTERS 64	SOUND OFF	B + W
POKE 56832,2	CHARACTERS 32	SOUND ON	B + W
POKE 56832,3	CHARACTERS 64	SOUND ON	B + W
POKE 56832,4	CHARACTERS 32	SOUND OFF	Color
POKE 56832,5	CHARACTERS 64	SOUND OFF	Color
POKE 56832,6	CHARACTERS 32	SOUND ON	Color
POKE 56832,7	CHARACTERS 64	SOUND ON	Color

*NOTE
COLOR GRAPHICS LOCATIONS ARE 3968 LOCATIONS BEYOND THE CHARACTER GRAPHICS LOCATIONS. EXAMPLE: CHARACTER MEMORY LOCATION 54302 AND COLOR GRAPHIC MEMORY LOCATION 58270 BOTH ADDRESS THE SAME SCREEN LOCATION

0	INVERTED	YELLOW	10	INVERTED	PURPLE
1		YELLOW	11		PURPLE
2	INVERTED	RED	12	INVERTED	SKY BLUE
3		RED	13		SKY BLUE
4	INVERTED	GREEN	14	INVERTED	BLACK
5		GREEN	15		BLACK
6	INVERTED	OLIVE GREEN			
7		OLIVE GREEN			
8	INVERTED	BLUE			
9		BLUE			

EXAMPLE POKE 58270,13

IMPORTANT NEW BOOKS FROM ELCOMP

Order No.	Price	Title	Review
150	$11.00	Care and Feeding of The Commodore PET	Eight chapters exploring PET hardware. Includes repair and interfacing information. Programming tricks and schematics.
151	$ 9.95	8K Microsoft BASIC Reference Manual	Authoritative reference manual for the original Microsoft 4K and 8K BASIC developed for Altair and later computers including PET, TRS-80, and OSI. OSI owners please take note!
152	$ 9.95	Expansion Handbook for 6502 and 6802 (S-44 Card Manual)	This is a revised, formal printing of our unique S-44 card manual. Describes all of the 4.5 x 6.5 44-pin S-44 cards including RAM, ROM, digital I/O, and MUX/A to D.
153	$14.90	Microcomputer Application Notes	Reprint of Intel literature.
154	$ 6.95	Complex Sound Generation	New, revised applications manual for the Texas Instruments SN 76477 Complex Sound Generator. Circuit board available.
155	$14.90	The First Book of TRS-80	Programs and applications ideas for the TRS-80 TRS-80 is a trademark of Tandy Corp.
156	$14.90	Small Business Programs	Business Programs in BASIC for use on most microcomputers.

Available direct from us NOW. Dealers please contact us for starter stock. Plans for either large or small dealerships.

NEW: MONJANA/1 – THE USER-GUIDED MONITOR ROM FOR COMMODORE CBM

A well-documented, powerful new monitor ROM that any Commodore CBM user can plug into one of the free ROM sockets. At a price of only $98, including an extensive manual, MONJANA/1 offers more user guidance and debugging aids than any other monitor available today. It is indispensable for anyone intending to program his CBM in 6502 machine language. (Trace, link, disassembler, dump, relocate printer option, line assemble, and much more.) Order No. 2001

Commodore Business Machines Inc. registered trade name of

Order No.	Price	Title	Review
2001	$98.00	MONJANA/1 CBM Monitor	Installs in free ROM socket.

NEW: OUR REDYSOFT CASSETTE SOFTWARE

3475	$49.00	Assembler for CBM (BASIC and machine language)	Complete assembler including disassembly and link. Cassette and manual.
3476	$69.00	Editor/Assembler in machine language.	Fully screen-oriented, scrolling, fail-safe operation.
3999	$34.50	BASIC with I/O for TRS-80	Extended level 1 with I/O and string handling extension. Cassette and manual.
8094	$ 1.10	Blank Cassette (Quantity 1)	Highest quality C-10 cassettes.
8095	$ 7.99	Blank Cassettes (Quantity 10)	
8096	$69.00	Blank Cassettes (Quantity 100)	

ELCOMP
PUBLISHING, INC.

3873L Schaefer Avenue
Chino, California 91710
(714) 591-3130

Payment: Check, Visa, Mastercharge
POSTPAID in U.S.A.
California add 6% tax


ELCOMP ELCOMP
ELCOMP ELCOMP
ELCOMP ELCOMP
ELCOMP ELCOMP
ELCOMP ELCOMP
ELCOMP ELCOMP
ELCOMP ELCOMP
ELCOMP ELCOMP
ELCOMP ELCOMP
ELCOMP ELCOMP
ELCOMP ELCOMP
ELCOMP ELCOMP
ELCOMP ELCOMP