July '77 **$1.50**

OHIO SCIENTIFIC'S SMALL SYSTEMS JOURNAL

VOLUME 1 NO. 1

features

page

departments

The magazine for 6502 computer enthusiasts!

ANNOUNCING OS-65D FROM OHIO SCIENTIFIC
It's the best reason yet to buy a full size floppy disk!

The OS-65D floppy disk operating system from Ohio Scientific is a unique high performance-cost effective disk operating system.

Almost all the disk control functions are done in software instead of hardware resulting in a system with a high degree of flexibility at low cost.

With OS-65D it takes only a fraction of a second to load and run a program unlike time consuming paper tape or cassette operating systems that can take up to 45 minutes to load a program! OS-65D can also store Databases for business applications.

OS-65D contains the following fully integrated systems:

BASIC with disk files
Assembler/Editor
Dis-Assembler
Relocator
Disk Copy Utilities
I/O Distributors
Extended Debugger

And OS-65D can store up to 247 kilobytes of information per disk surface. That's up to 252, 928 characters easily accessible at your fingertips.

OS-65D is also flexible because multiple mixed sector length files can be opened simultaneously. This is important because the user can match sector length to his desired record length thereby minimizing mechanical movement resulting in high speed operation.

Sectors can range from 256 to 3.2K bytes in length in 256 byte increments.

Capabilities for sequential, random access, index sequential and system command files are available in OS-65D.

And finally OS-65D is fully compatible with our floppy disk bootstrap PROM so that the user can use the disk immediately on system power up.

See the other side of this sheet for OS-65D specifications.

Introduction

Ohio Scientific, Inc. has already made one attempt to get a journal into regular publication. In September 1976 our first and only issue of the OSI Systems Journal was published. Being a fairly young company, we soon found that a lot of time was necessary to assemble a staff adequate to the task of expanding our customer services beyond the fundamental business of manufacturing and selling small computers. That original aspect of our business is today in an excellent position, and we are now among the leaders in the personal computer industry. Our products are in use around the world, and we will soon be opening a new production plant, adding 20,000 square feet to our existing facilities. This growth has enabled us to add to our staff a full-time editor of our journal. It will be published monthly, and a free year's subscription will automatically accompany all Challenger orders. In each issue we will bring you features and news on the latest progress in company software and hardware. Our special section, 1K Corner, offers the beginning programmer a chance to get started with an easy-to-follow routine. Our Odds & Ends column includes miscellaneous fillers of timely interest to all Ohio Scientific users. Our section entitled Bugs & Fixes includes documentation of bugs that may have occasionally been discovered in our programs and assembly manuals. Finally we include the current catalog prices of our hardware and software. Many of our products will be highlighted in larger advertisements placed throughout the journal. We hope you will share with us your thoughts on how this publication may be expanded and improved. If you wish to contribute your own articles, we will be glad to consider them. Please send materials or comments to:

Ohio Scientific, Inc.
Small Systems Journal
Box 36
Hiram, OH 44234

Ohio Scientific's Small Systems Journal is published monthly by Ohio Scientific Inc., P.O. Box 36, Hiram, Ohio 44234. The subscription rate is six dollars for six issues. Individual copies are $1.50.

Vol. 1, No. 1	July 1977
Editor-in-chief	Gary Deckant
Production Manager	Rob Spademan
Contributing Editors	Mike Cheiky
	Eric Davis
	Marcel Meier
Production Assistant	Cindy Warrick

74,000,000!

Ohio Scientific has just received its first 74 million-byte disk drives. These new large disks represent a significant advancement in the state of the art of mass storage. Ohio Scientific is one of the first companies to be integrating these new disk drives into its computer systems. The capability provided by this large disk is completely out of the realm of what most people have considered for microprocessor-based computers. However, the relatively small size, approximately 22" long x 7" high x 19" wide (65 lb.) for the disk drive, and its unbelievable low cost, which we are projecting to be $6,000 retail, make this very large disk storage device economically feasible for any small business application, and even some advanced hobbyist or research-based applications of small computers. We are currently preparing an information package on the 74 megabyte disk and how it compares to smaller floppies and mini-floppies. Please send a letter on company stationery or a 9 x 12 envelope with 57¢ postage to receive a copy of the big disk information package as soon as it comes out. There will, of course, be a lot more information about the new large disk in upcoming issues of the journal.

Feature Article

Understanding and Using the 6502 Assembler

Most beginners in the field of computer programming start with BASIC because of its simplicity and flexibility. Moreover, since it can be run on many different computers, it is almost universal in its application. However, as programmers gain more experience and sophistication, they tend to use assemblers more frequently. Assemblers utilize special codes to give instructions to the processor and require a more detailed, step-by-step routine which very closely reflects the computer's internal functions. There are several advantages and disadvantages of the assembler:

Advantages: 1) more concise, because it allows you to write programs in the shortest possible code; 2) more direct, because you are "talking" to the processor in a language that does not need to be translated; 3) more efficient, because it enables you to make full use of the computer's speed, I/O, and other resources.

Disadvantages: 1) more error-prone, because the language is sometimes hard for humans to understand; 2) more limiting, because it cannot be used on computers other than the type for which it was designed; 3) more time-consuming, because the typical program contains more instruction than higher-level languages require.

As a help to those interested in learning more about OSI 6502 assembler language, here is a basic description of the principles behind it.

The actual form of the instructions which the user types on the 6502 terminal keyboard consists of a three-letter mnemonic. The fifty-five standard mnemonics are shown on page 7. These mnemonics have been devised for the programmer's convenience, but the processor itself is incapable of interpreting them on its own. The assembler's job is to convert these mnemonics (known technically as operational codes, or opcodes) into "machine language" or object codes. For example, the opcode BPL translates into the object code 10; PHA becomes 48, and so on. Every opcode and piece of data (operand) has an equivalent object code which directs the processor to a certain memory location known as an address. The address may be predetermined by the programmer, because it is distinguishable from the form of the object code itself. For instance, if the first instruction of an assembler program is *=$4000, this indicates that the program counter is to start at location 4000 ($ indicates hexadecimal notation). The opcode in the statement which follows will therefore be stored in location 4000, the next opcode or operand in location 4001, etc., until some other address is specified.

The fifty-five opcodes are described in the MOS Technology MCS6500 Microcomputer Family Programming Manual, which is included in all Challenger manuals. It is also available from MOS Technology, Inc., Valley Forge Corporate Center, 950 Rittenhouse Road, Norristown, Pa., 19401. The object codes corresponding to them are interpreted by the microprocessor in binary notation, which in turn are converted to hexadecimal notation whenever appearing on a terminal. This is done because hexadecimal notation corresponds more directly to binary than does decimal. Each object code consists of a two-digit hexadecimal or eight-digit binary expression called a byte.

When entering data or referring to memory locations, it is necessary to know how to indicate which notation is to be used. The Assembler is capable of interpreting binary, octal, decimal, and hexadecimal numbers. The programmer uses the following symbols to distinguish them:

%	binary
@	octal
(no sign)	decimal
$	hexadecimal

For a more detailed explanation of binary and hexadecimal notation, refer to the OSI 300 Trainer Manual, available from Ohio Scientific, Inc. for $5.00. The trainer manual also has several fundamental experiments in 6502 "machine code."

The instruction format of the assembler consists of assembler directives and machine instructions. There are five assembler directives in the 6502 system, which give operating instructions to the assembler. Probably the most common of these are the equals sign (=) and .END, which is optionally used to signal the end of a program. The other three assembler directives, .BYTE, .WORD, and .DBYTE, are described in the MOS Technology Cross Assembler Manual. Each directive must be preceded by a period.

Machine instructions contain one of the 55 opcodes [cf. p. 7]. Depending on the particular opcode, there may also be an operand, i.e., the value upon which an operation is to be performed; a label, which may at some time be useful in the execution of the program; and a comment, made at the discretion of the programmer. In addition, each line in the program begins with a statement number, which allows resequencing, instruction changes, and all editing features also available in BASIC. These five sections, or fields, of a machine instruction must be separated from one another by at least one space. There are several other restrictions on the precise form that the string of characters may take, and these are described in the MOS MCS6500 Manual.

Here is an example of a typical machine instruction showing all fields described above:

140	START	AND	#$7F	MASK DATA TO 7 BITS
statement number	label	opcode	operand	comment

When the assembled program is displayed on a terminal, the memory location and object code of each item in the instruction line also appear. It may look like this:

140	03B8	297F	START	AND	#$7F	MASK DATA TO 7 BITS
statement number	opcode location, or program counter	object code of opcode-[object code of operand-[	label	opcode	operand	comment

If the location of the opcode AND is 03B8, then the location of the next byte, in this case, the operand #$7F, is understood to be 03B9. The next line in the program would show 03BA in the program counter field, and so on.

Each opcode is said to utilize one of several established conventions for referencing data. These conventions, known as addressing modes, are determined by the type of data (i.e., operand) being used. Certain opcodes do not use operands at all, because there is no specific value involved, or because the opcode defines an operation in itself with no further data necessary. These opcodes are said to be in the implied addressing mode, as the operand is not expressed, but understood or implied. Examples of opcodes in the implied addressing mode are CLC, INX, and NOP. When an upcode is followed by an operand of a constant value, the opcode is then in the immediate addressing mode. For instance, in the command LDA #$6D, where # signifies a constant numerical value, LDA is in the immediate addressing mode. On the other hand when the operand is not a constant, but takes different values at different times, then the opcode is in the absolute addressing mode. If the above example had read, instead, LDA $4300, without any #, this would indicate that whatever value happened to be at location 4300 was being sought, and that that value may change from time to time. Therefore, LDA would in such a case be in the absolute addressing mode.

When the value of an operand is not a constant, obviously, then it is referred to by its address in memory rather than by its value. The addresses in memory are divided into groups or pages of 256 bytes each. Normally, when a location is specified in the operand field of a machine instruction, the location consists of two bytes, one signifying the page number, and one designating the particular byte on that page. If the location, however, is on the first page in memory (known as page zero 0)then the byte specifying the page number (i.e., 00) is omitted, thus saving processing time. When an operand refers to a location on page zero, then the opcode is in the page zero addressing mode. An example of such would be LDA $0040 (where 0040 is a location on page 0); this is usually written LDA $40.

Ordinarily the processor executes each statement in turn; but on occasion, it is required to make a decision on which statement to perform next, depending on previous data. In that case a branch instruction is used, which is only obeyed if a given condition is fulfilled. Branch instructions (e.g., BPL, BNE, etc.) are opcodes in the relative addressing mode. The operand following a branch instruction indicates which step is to be executed if the condition is met.

Another type of machine instruction which disrupts the natural order of execution is the JMP statement. This statement, however, does not depend on any test for its command to be carried out. Therefore, it is not in the relative addressing mode, but usually is in the absolute addressing mode.

There are a number of other addressing modes, all of which are discussed at length in the MOS manuals

Within the microprocessor systems are certain operational signals, commonly called status bits or flags. These flags, collectively referred to as the processor status register, affect the execution of statements, and are themselves affected by the processing of certain instructions. For example, the carry flag is set whenever two numbers are added such that the result exceeds a one-byte field. The zero flag is set whenever the result of an operation equals zero. The decimal flag is set if the programmer uses a value in decimal notation. The overflow flag is designed to be used primarily with signed addition and subtraction. The negative flag is set whenever an operation results in a negative number. When a flag is set, it is placed at a value of 1 (i.e., the flag is "on"); if the corresponding condition is not met at the time, it is reset to 0 (=off). The use of each addressing mode generally affects one or more of these flags, which may have a subsequent effect on other statements in the program. The Branch Instructions, in particular, depend on the status bits.

These are some of the basic principles of the assembler language of the OSI 6502. In order to illustrate the practical value of these fundamentals, an elementary program is very helpful.

As an example, suppose a user were to enter on a terminal the source program which is listed at the back of the OSI 65V PROM Monitor manual:

```
 10 ;65V DEMONSTRATION PROGRAM
 20 ;
 30 PNTL=0
 40 PNTH=1
 50 ;
 60 INKEY=$FEED  INPUT A CHARACTER SUBROUTINE
 70        *=2
 80 MESSAG .BYTE 'OSI 65V. ',$5F
 90 ;
100        *=$200
110 MAIN   LDA #MESSAG  ADDRESS OF MESSAGE
120        LDX #0  RESET INDEX
130        JSR STRING  OUTPUT MESSAGE
140        LDX #0  RESET INDEX AGAIN
150 LOOP   JSR INKEY  GET AN INPUT CHARACTER
160        STA $D224,X  STORE IT ON 440 SCREEN
170        INX  INCREMENT INDEX
180        JMP LOOP  GO BACK FOR ANOTHER
190 ;
200        *=$300
210 STRING STA PNTL  SET LOW ADDRESS OF MESSAGE
220        LDA #0  SET HIGH ADDRESS OF MESSAGE
230        STA PNTH
240        LDY #0  RESET INDEX
250 ANOTHR LDA (PNTL),Y  GET A CHARACTER
260        CMP #$5F  SEE IF IT IS THE LAST
270        BEQ EXIT
280        STA $D1E4,X  STORE IT ON 440 SCREEN
290        INX  INCREMENT INDEX
300        INC PNTL  INCREMENT MESSAGE POINTER
310        BNE ANOTHR  LOOP BACK OR
320        INC PNTH  EXIT IF PAST PAGE 0
330 EXIT   RTS  RETURN
340         END
```

Depressing the A key would then cause the terminal to display the following assembled listing:

```
10 0000          ;65V DEMONSTRATION PROGRAM
20 0000          ;
30 0000          PNTL=0
40 0000          PNTH=1
50 0000          ;
60 0000          INKEY=$FEED  INPUT A CHARACTER SUBROUTINE
70 0002          *=2
80 0002 4F       MESSAG  .BYTE 'OSI 65V.',$5F
80 0003 53
80 0004 49
80 0005 20
80 0006 36
80 0007 35
80 0008 56
80 0009 2E
80 000A 5F
90 000B          ;
100 0200         *=$200
110 0200 A902    MAIN    LDA #MESSAG   ADDRESS OF MESSAGE
120 0202 A200            LDX #0  RESET INDEX
130 0204 200003          JSR STRING   OUTPUT MESSAGE
140 0207 A200            LDX #0  RESET INDEX AGAIN
150 0209 20EDFE  LOOP    JSR INKEY   GET AN INPUT CHARACTER
160 020C 9D24D2          STA $D224,X   STORE IT ON 440 SCREEN
170 020F E8              INX  INCREMENT INDEX
180 0210 4C0902          JMP LOOP  GO BACK FOR ANOTHER
190 0213         ;
200 0300         *=$300
210 0300 8500    STRING  STA PNTL  SET LOW ADDRESS OF MESSAGE
220 0302 A900            LDA #0  SET HIGH ADDRESS OF MESSAGE
230 0304 8501            STA PNTH
240 0306 A000            LDY #0  RESET INDEX
250 0308 B100    ANOTHR  LDA (PNTL),Y  GET A CHARACTER
260 030A C95F            CMP #$5F  SEE IF IT IS THE LAST
270 030C F00A            BEQ EXIT
280 030E 9DE4D1          STA $D1E4,X  STORE IT ON 440 SCREEN
290 0311 E8              INX  INCREMENT INDEX
300 0312 E600            INC PNTL  INCREMENT MESSAGE POINTER
310 0314 D0F2            BNE ANOTHR  LOOP BACK OR
320 0316 E601            INC PNTH  EXIT IF PAST PAGE 0
330 0318 60      EXIT    RTS  RETURN
340 0319                 .END
```

Each statement begins with a statement number followed by an opcode location (or program counter). None of the first seven statements contains any object code because statements 10, 20, and 50 are comments (indicated by the semicolon) and statements 30, 40, 60, and 70 are assembler directives (indicated by the equals sign). In statements 30 and 40, the variables PNTL and PNTH are stored in locations 0 and 1, respectively. These locations could also be written 0000 and 0001, which shows that they are on page zero (by the initial pair of zeros in each case). In line 60, a special subroutine already included in the computer's 65V PROM Monitor ($FEED) is given the label INKEY. Line 70 sets the program counter at location 2, from which point the program proceeds.

In line 80, the .BYTE directive "freezes" the line number until the last byte in its field is placed in memory. In line 90, a semicolon indicates another comment (a blank line to separate the actual program being considered).

Line 100 contains another directive, this one setting the program counter at 200. Therefore, the main program starts at location 200 (cf. line 110). The instruction LDA loads the number 2 (=the location of the .BYTE directive labeled MESSAG) into the accumulator. The X register is reset to 0 (line 120), because it will be used later as a counter. In line 130, we are immediately directed to the instruction labeled STRING (line 210) for the next step. This causes the 2 that has been loaded into the accumulator to be stored in the location to which PNTL has been assigned (i.e., location 0, cf. line 30). In other words, the value of 2 is now stored in memory location 0. Similarly, lines 220 and 230 load a 0 in the accumulator and store it in location 1 (i.e., the address to which PNTH has been assigned in statement 40). This insures a page zero address later on.

Line 240 resets the Y register at 0. Lines 250 to 330 are a subroutine which prints on the 440 screen those characters that have been stored in this particular case. The first step in the subroutine, line 250, loads the accumulator with the ASCII character at the location PNTL+Y. As we pass through the subroutine the first time, the value at PNTL is 2 (recall that we had stored this 2 at the location to which we had assigned the label PNTL), namely location 0. To this 2, the processor must add the value of the Y index (=0) or 2+0=2. Next, the ASCII character occupying location 2 is called from memory (=4F, cf. line 80, location 0002). Line 260 gives an instruction to compare 4F with 5F. If these were equal, according to line 270, we would proceed to line 330, where we find the EXIT label. Since they are not equal, we merely proceed to the next line, no. 280. This line causes the character corresponding to ASCII code 4F to be printed at the location D1E4+X on the 440 screen. We arbitrarily chose D1E4 because it occupies a suitable location on the screen. Recall that X=0 during our first pass, according to line 120. Therefore, screen location D1E4+0=D1E4 will be occupied by the letter O. (=ASCII code 4F).

Line 290 increases the X value by 1 to X=1. Line 300 increases the value at PNTL (location 0) by 1 to a value of 3. In line 310, the processor is directed back to the start of the loop (line 250, labeled ANOTHR), unless the zero flag is set. Since on our first pass, the ASCII value was not zero, the zero flag has not been set. Therefore the subroutine is executed again on a second pass. On each pass, obviously the value in location 0 (labeled PNTL) has increased by one. Therefore an ASCII character for a consecutive address is called during each pass. Furthermore, the X value also increases by one each time the subroutine is executed, which results in those ASCII characters being stored in consecutive locations on the 440 screen in the same order in which they were called from memory.

This procedure continues until the last pass, when statement 260 compares 5F (the termination character) to 5F. Since two equal entities are now being compared, thus fulfilling the prescribed condition, the program moves on to line 330, labeled EXIT. This instructs the processor to return from the subroutine (RTS). Recall that we first entered the subroutine after executing line 130. The last object code in that line was 0206. The processor has automatically stored the next location (0207) as a return address, until whenever we have an exit from the subroutine. That time is now, and so we return to location 0207 (line 140).

The remainder of this program (lines 140-180) is really a separate program in itself. It allows the programmer to enter any characters desired and store (i.e., display) them on the 440 screen. The storage location will be slightly different (i.e., beginning at location D224). Any group of characters entered in line 80 must end with a 5F, because that has been chosen as the termination character. The routine is exited by pushing the reset button.

This program can also be assembled in a form which shows only the object code, if the programmer issues the A2 command. The following lines are then displayed:

```
;0900024F534920363556 2E5F0264
;130200A902A200200003A20020EDFE9D24D2E84C09020704
;1803008500A9008501A000B100C95FF00A9DE4D1E8E600D0F2E6010C0B
;01031860007C
```

These lines are interpreted as follows: Each section beginning with a semicolon is called a record. The records mainly consist of the two-digit bytes which formed the object code of the program. However, the first byte always specifies the number of bytes in the record; the next two bytes give the memory location of the initial byte; and the last two bytes give the sum total of all the bytes in the record. As usual, all bytes are in hexadecimal notation. This assembled version of the program is called the "checksum format," because it is used to check against errors during loading.

There is also an A1 command which outputs those lines containing errors and error messages only, and an A3 command which places the object code in memory, producing no display.

Below is an interpretation of selected lines in the 65V demonstration program, with a discussion of how the principles discussed earlier apply to it.

line	byte	explanation
110	A9	object code for LDA in immediate addressing mode
	02	location of #MESSAG, as assigned in lines 70-80 (the program counter 0200 indicates that A9 is in location 0200, and 02 is in location 0201; this pattern applies throughout)
130	20	object code for JSR (always in absolute addressing mode)
	0003	must be inverted to 0300 to reveal location of the label STRING; this indicates that the label identifies location 0300, which, of course, it does)
150	20	object code for JSR
	EDFE	must be inverted to FEED, which is a subroutine to take the hexadecimal equivalent of AN ASCII character, put it in the accumulator and execute a program. The FEED subroutine is in this case labeled INKEY.
210	85	object code for STA in zero page addressing mode
	00	location of PNTL, as assigned in line 30; that is, a page zero location
270	F0	object code for BEQ (always in relative addressing mode)
	0A	number of object codes further down, where the next step to be followed can be found, provided the given condition is met. This line tells us that that step is labeled EXIT. To demonstrate, consider the next object code (9D) as step 0, and count up to object code #A: 9D,E4,D1,E8,E6,00,D0,F2,E6,01,60 / 0 1 2 3 4 5 6 7 8 9 A
290	E8	object code for INX--always in implied addressing mode, therefore no operand, and only one byte in object code.

Opcodes used in the 6502 Assembler

ADC	Add with Carry to Accumulator
AND	"AND" to Accumulator
ASL	Shift Left One Bit (Memory or Accumulator)
BCC	Branch on Carry Clear
BCS	Branch on Carry Set
BEQ	Branch on Zero Result
BIT	Test Bits in Memory with Accumulator
BMI	Branch on Result Minus
BNE	Branch on Result not Zero
BPL	Branch on Result Plus
BRK	Software Interrupt
BVC	Branch on Overflow Clear
BVS	Branch on Overflow Set
CLC	Clear Carry Flag
CLD	Clear Decimal Mode
CLI	Clear Interrupt Disable Bit
CLV	Clear Overflow Flag

CMP	Compare Memory and Accumulator
CPX	Compare Memory and Index X
CPY	Compare Memory and Index Y
DEC	Decrement Memory by One
DEX	Decrement Index X by One
DEY	Decrement Index Y by One
EØR	Exclusive-or Memory with Accumulator
INC	Increment Memory by One
INX	Increment X by One
INY	Increment Y by One
JMP	Jump to New Location
JSR	Jump to New Location Saving Return Address
LDA	Transfer Memory to Accumulator
LDX	Transfer Memory to Index X
LDY	Transfer Memory to Index Y
LSR	Shift One Bit Right (Memory or Accumulator)
NØP	Do Nothing - No Operation
ØRA	"OR" Memory with Accumulator
PHA	Push Accumulator on Stack
PHP	Push Processor Status on Stack
PLA	Pull Accumulator from Stack
PLP	Pull Processor Status from Stack
RØL	Rotate One Bit Left (Memory or Accumulator)
RTI	Return From Interrupt
RTS	Return From Subroutine
SBC	Subtract Memory and Carry from Accumulator
SEC	Set Carry Flag
SED	Set Decimal Mode
SEI	Set Interrupt Disable Status
STA	Store Accumulator in Memory
STX	Store Index X in Memory
STY	Store Index Y in Memory
TAX	Transfer Accumulator to Index X
TAY	Transfer Accumulator to Index Y
TSX	Transfer Stack Register to Index X
TXA	Transfer Index X to Accumulator
TXS	Transfer Index X to Stack Register
TYA	Transfer Index Y to Accumulator

1K Corner

In the game of NIMB, two players start with a set of N objects. The players then, in turn, each remove 1, 2, or 3 objects from the set. The player who is forced to take the last object is the loser.

23 NIMB, for OSI 65V systems, is played in the same manner. At the start of the game, the computer will tell you that N=23 and ask how many you would like to take. After keying in your move, followed by a carriage return, the computer will tell you its move, the new value of N, and then ask you how many you would like to take. The game continues in this fashion until you or the computer is forced to lose. A new game may be instituted by "↑" (shift N) and illegal moves are not accepted.

23 NIMB resides at 0200$_{16}$ to 0333$_{16}$ and is entered at 0200$_{16}$. The initial value of N may be changed from 23 to any decimal number from 01 to 99 by changing program step 0204$_{16}$ from 23 to the desired value.

At the right is the dump for the NIMB program illustrating the memory locations of the object code from the program.

Bugs & Fixes

Expanded and simplified assembly instructions for the audio cassette portion of the old 430A PC board are available free of charge from OSI. To get this information, please send a self-addressed, stamped, legal-size envelope with 24¢ postage, and specify on the envelope "430A cassette assembly instructions."

* * *

Early audio cassette and paper tape versions of the 6502 assembler do not properly assemble the PLP instruction. To correct this, simply change location 0FA6 from 48 to 28. Disk system users should always utilize the assembler in OS-65D instead of older versions of the assembler. It has a much faster editor package as well as additional features.

* * *

A 1K current limiting resistor should be inserted in the foil run to pin 14 of each of the 1408L8s that you are using on old 430A boards. The 430B board has this in the parts layout and documentation.

* * *

Early cassette and paper tape versions of 8K BASIC have a bug which causes the DEFINE function to hang in a loop when executed. The following patch should correct the problem and allow the DEFINE function to work normally. Change locations 0228 (hex) onward to 50 10 A9 11, all in hex.

* * *

When constructing 430B-based Audio Cassette Interfaces, omit the procedure on page B5 of the manual, part 5, which refers to the keyboard echo program. This procedure may not function properly even if the board is properly set up.

* * *

The cassette I/O in cassette versions of 8K BASIC is subject to the same null requirements as the paper tape versions. We recommend that the user set the null function equal to 10 before storing programs on cassette. Before reading programs from cassette into the computer in BASIC, be sure the null function is set at 0. The null function is normally set at 0, unless modified by the user.

```
U0200,0332
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0200  20 20 03 A9 23 85 FD A5 FD 20 F7 02 C9 30 D0 02
0210  A9 20 8D 0A 03 A5 FD 20 FB 02 8D 0B 03 A0 20 AD
0220  08 03 91 FE EE 20 02 C8 C0 2E D0 F3 A9 08 8D 20
0230  02 20 E0 02 C9 5E F0 C8 C9 31 30 F5 C9 34 10 F1
0240  C8 C8 91 FE 38 29 03 85 FC A5 FD E5 FC F0 5D 30
0250  5B 85 FD 20 E0 02 C9 0D D0 F9 A9 20 91 FE 18 A9
0260  01 C5 FD F0 2F 10 08 85 FA A9 04 65 FA 10 F2 38
0270  A5 FD E5 FA 09 30 8D 07 03 A5 FA 85 FD A0 01 AD
0280  00 03 91 FE EE 80 02 C8 C0 09 D0 F3 A9 00 8D 80
0290  02 4C 07 02 A9 01 C5 FD F0 33 38 A5 FD E5 FB 85
02A0  FD A5 FB 09 30 8D 07 03 A5 FD 10 D1 20 20 03 A0
02B0  20 AD 16 03 91 FE EE B2 02 C8 C0 28 D0 F3 A9 16
02C0  8D B2 02 20 E0 02 C9 5E D0 F9 4C 00 02 20 20 03
02D0  A0 22 A9 49 8D E5 D0 A9 1A 8D B2 02 10 D3 FB 60
02E0  A9 03 C6 FB F0 0D AD 01 DF 30 F5 48 AD 01 DF 10
02F0  FB 68 60 85 FB 10 EF 4A 4A 4A 4A 29 0F 09 30 60
0300  49 20 54 4F 4F 4B 20 32 4E 3D 32 33 2C 48 4F 57
0310  20 4D 41 4E 59 3F 59 4F 55 20 4C 4F 53 45 20 20
0320  F8 A9 C5 85 FE A9 D0 85 FF A0 FF A9 20 91 FE 88
0330  D0 FB 60 FF FF FF FF FF FF FF FF FF FF FF FF FF
:
```

An Introduction

The Auto-Load Cassette System

Ohio Scientific's computers equipped with a 440 video display board, usually have a 65V PROM Monitor. This 256-word PROM provides memory load, examine, and program execute functions with a simple ASCII parallel keyboard and a video display connected to the 440 interface. The 65V PROM also has a cassette bootstrap program which allows input to be from an audio cassette interface located on a 430 board, instead of from the keyboard. The 65V accepts a starting address for a program with data in sequential form, and finally allows the user to input the execution starting point of the program by typing a G on the keyboard. This simple format, known as the OSI 65V format, is also used by the audio cassette input if the user types an L. Thus the audio cassette has to provide input just as the user would type it on the keyboard

When a computer is first turned on, the only program it has in its memory is its PROM Monitor. In the case of the 65V, it simply has load, examine, and execute functions as mentioned above. It does not have a CRT routine, which allows a 440 screen to act as a conventional computer terminal, because PROM is quite expensive and less versatile than RAM. The "CRT Routine" is one of the functions that one is expected to put in RAM. So in order to have any large, useful program in the computer, we will need at least the CRT routine and that program. It is also convenient to have other routines in the computer when running a large program. These routines, in conjunction with the program of interest, are called an operating system.

The auto-load cassettes offered by Ohio Scientific construct an entire operating system, including the program of interest. The user simply has to turn the computer and the cassette recorder on, press the reset button, and type L on the keyboard. The rest is automatic. This is accomplished by placing on the cassette a CRT routine and a checksum loader in the 65V format. When the user types an L, the machine accepts a CRT subroutine in the simple address and data format. It then accepts a checksum loader, a more sophisticated program loader which utilizes the CRT routine for output. The entire program up to this point is now executed.

Then the program of interest is loaded in a checksum format, because it will tell the user if a loading

```
0000          ;
0000          ;  -AUDIO CASSETTE TAPE GEN.-
0000            ;
0D50            *=$0D50
0D50          ;
0D50          ;CRT SIMULATOR ROUTINE
0D50          ;
0D50 20EE0E   CRAUD JSR UOUT   OUTPUT TO TTY
0D53          ; THEN TO TV
0D53 8D160E   CROUT STA TMP   SAVE CHAR.
0D56 8A             TXA  SAVE REGISTERS
0D57 48             PHA
0D58 98             TYA
0D59 48             PHA
0D5A AD160E         LDA TMP
0D5D 297F           AND #$7F  MASK TO 7 BITS
0D5F C90A           CMP #$A  CR OR LF?
0D61 F031           BEQ LF
0D63 C90D           CMP #$D
0D65 F024           BEQ CR
0D67 C920           CMP #$20   $20 TO $5F?
0D69 3018           BMI EXIT
0D6B C960           CMP #$60
0D6D 1014           BPL EXIT
0D6F 8D170E         STA SAVER  PUT CHAR. @ CURSOR
0D72 20F50D         JSR UNS
0D75 EE1A0E         INC POINT  MOVE CURSOR OVER
0D78 AD190E         LDA LEN   NEED AUTO CRLF?
0D7B CD1A0E         CMP POINT
0D7E 3011           BMI AUTCR
0D80 20070E         JSR SAV
0D83 68       EXIT  PLA  RESTORE REGISTERS
0D84 A8             TAY
0D85 68             PLA
0D86 AA             TAX
0D87 AD160E         LDA TMP
0D8A 60             RTS
0D8B          ;
0D8B 20FF0D   CR    JSR RT
0D8E 4C830D         JMP EXIT
0D91          ;
0D91 20020E   AUTCR JSR RTA  AUTO CRLF
0D94          ;
0D94 20F50D   LF    JSR UNS  UNSAVE CURSOR
0D97 A9D0           LDA #$D0  SCROLL TO TOP
0D99 8DAD0D         STA MOV+2
0D9C 8DB00D         STA MOV+5
0D9F A900           LDA #0
0DA1 8DAC0D         STA MOV+1
0DA4 8DAF0D         STA MOV+4
0DA7 A020     LP4   LDY #$20  RESET OFFSETS
0DA9 A200           LDX #0
0DAB B9FFFF   MOV   LDA $FFFF,Y  MOVE LINE ELE. UP
0DAE 9DFFFF         STA $FFFF,X
0DB1 E8             INX  INC. OFFSETS
0DB2 C8             INY
0DB3 E020           CPX #$20  DONE?
0DB5 D0F4           BNE MOV
0DB7 8A             TXA  ADD $20 TO POINTERS
0DB8 18             CLC
0DB9 6DAC0D         ADC MOV+1
0DBC 8DAC0D         STA MOV+1
0DBF 8DAF0D         STA MOV+4
0DC2 9006           BCC NC4
```

error has occurred, and allow him to recover from the error without reloading the tape from the start. This is especially helpful in the case of programs which take some time to load. Again the checksum loader is not part of the initial PROM because the checksum loader is too lengthy for the PROM.

There is a final line on the tape which is the starting address of the program of interest, so that when the L is typed, a CRT routine is entered, followed by the checksum loader; after the checksum loader is executed, the program of interest is displayed on the screen while being loaded, so that the user may detect an error, if any has occurred, in loading. Then the program of interest is automatically executed by a command on the tape, and the user can then proceed with the program. In other words, the last thing the user has to do is type an L on the keyboard, which results in the loading of the entire operating system and the program of interest.

The entire software package which allows the user to generate his own auto-load cassettes is called the auto-load cassette tape generator. This package, listed here in assembler form, also includes, as a subset of this program the routines which are actually placed on the cassette, e.g., the CRT routine and the checksum loader. These are present on every auto-load cassette manufactured by OSI. Some are located at the actual addresses of the listing; others are located offset in memory. The user should be able to locate these routines and modify them if desired.

The auto-load cassette tape generator contains the following routines (line nos. refer to accompanying printout): CRT simulator (lines 20-1040), which is used both by the tape generator program itself, and later by the auto-load package; checksum loader (lines 1180 to 1945); cassette I/O subroutines (1955 to 1983), which make the program, to a great extent, PROM independent; checksum generator (2014 to 2660); 65V format generator (2710 to 3165); and mainline program (3180 to 3430).

In operation, the user must place the auto-load cassette package in memory at the address shown in the listing or at some other origin suitable for his application. He then must place the program which he wishes to store on cassette in memory. Then the parameters in lines 1090, 1100, 1110, and 1120 must be set, in order to specify the start and end of the checksum record (referred to in the comment field as a KIM-1 record). These define the beginning and end of the program of interest which the user wishes to store on cassette. The user must then place assembler lines 3420 and 3430 at the starting address of his program. He then turns on the cassette recorder, places it in record

```
 590 0DC4 EEAD0D        INC MOV+2
 600 0DC7 EEB00D        INC MOV+5
 610 0DCA A9D3   NC4    LDA #$D3   DONE?
 620 0DCC CDAD0D        CMP MOV+2
 630 0DCF D0D6          BNE LP4
 640 0DD1 AD180E        LDA HOME
 650 0DD4 29E0          AND #$E0   FIND START OF
 660 0DD6 AA            TAX        HOME LINE
 670 0DD7 CDAC0D        CMP MOV+1
 680 0DDA D0CB          BNE LP4
 690 0DDC A920          LDA #$20   FILL HOME LINE
 700 0DDE A8            TAY        WITH BLANKS
 710 0DDF 9D00D3 LP5    STA $D300,X
 720 0DE2 E8            INX
 730 0DE3 88            DEY
 740 0DE4 D0F9          BNE LP5
 750 0DE6 20070E        JSR SAV  SAVE CURSOR
 760 0DE9 4C830D        JMP EXIT  DONE!
 770 0DEC               ;
 780 0DEC AD1A0E CALC   LDA POINT  CALC. CURSOR
 790 0DEF 18            CLC        OFFSET FROM
 800 0DF0 6D180E        ADC HOME   $D300
 810 0DF3 AA            TAX
 820 0DF4 60            RTS
 830 0DF5               ;
 840 0DF5 20EC0D UNS    JSR CALC   UNSAVE OLD CHAR.
 850 0DF8 AD170E        LDA SAVER  (RESTOR CURSOR)
 860 0DFB 9D00D3        STA $D300,X
 870 0DFE 60            RTS
 880 0DFF               ;
 890 0DFF 20F50D RT     JSR UNS  DO A RETURN &
 900 0E02 A900   RTA    LDA #0
 910 0E04 8D1A0E        STA POINT
 920 0E07               ;
 930 0E07 20EC0D SAV    JSR CALC  PUT DOWN CURSOR
 940 0E0A BD00D3        LDA $D300,X  (SAVE OLD CHAR.)
 950 0E0D 8D170E        STA SAVER
 960 0E10 A95F          LDA #$5F  ASCII FOR CURSOR
 970 0E12 9D00D3        STA $D300,X
 980 0E15 60            RTS
 990 0E16               ;
1000 0E16 00     TMP    .BYTE 0
1010 0E17 20     SAVER  .BYTE $20
1020 0E18 64     HOME   .BYTE $64
1030 0E19 18     LEN    .BYTE $18 — Line Length
1040 0E1A 00     POINT  .BYTE 0
1050 0E1B               ;
1060 0E1B 0E24   RESET  .DBYTE READ
1065 0E1D               ;
1070 0E1D 00     CKL    .BYTE 0   ;CHECKSUM Loader
1080 0E1E 00     CKH    .BYTE 0
1090 0E1F 00     ENDL   .BYTE 0 ;END OF KIM-1 DUMP
1100 0E20 00     ENDH   .BYTE 0
1110 0E21 00     PNTL   .BYTE $00  ;START OF KIM-1 DUMP
1120 0E22 02     PNTH   .BYTE $02  ;SHOULD BE > $1FF
1130 0E23 00     TEMP   .BYTE 0 ;TEMPORARY
1150 0E24               ;
1160 0E24               ;   LOAD CHECKSUM FORMAT MAG TAPE
1170 0E24               ;
1180 0E24 20DF0E READ   JSR UIN
1181 0E27 C924          CMP #'$   AUTO START?
1182 0E29 F06C          BEQ AUTOS
1190 0E2B C93B          CMP #';   IS THIS A NEW REC.?
1200 0E2D D0F5          BNE READ
1210 0E2F A900          LDA #0  ZERO CHECKSUM
1220 0E31 8D1D0E        STA CKL
1230 0E34 8D1E0E        STA CKH
1240 0E37 20B90E        JSR INBYT  READ REC. LEN.
1250 0E3A AA            TAX
1260 0E3B 20B60E        JSR CKIN  READ SA
1270 0E3E 8D520E        STA LOCA+2
1280 0E41 20B60E        JSR CKIN
1290 0E44 8D510E        STA LOCA+1
1300 0E47 20A90E        JSR CKSM
1310 0E4A 8A            TXA  ZERO LEN. REC.?
```

mode, and executes the auto-load package at line 3200, which in this assembly would be at 0FCE. The auto-load generator first calls up the 65V format generator, which then loads the CRT routine and the checksum loader onto the tape. We then invoke the checksum generator, which loads the program of interest onto the tape. Finally, the restart (auto-start) vector, which specifies the starting location of the program of interest, is placed on the tape. The program we are running to generate the tape then exits to the 65V monitor. The actual data being outputed to the cassette is also being outputed to the screen. The user will see the CRT routine and checksum loader routine outputed in 65V format. Then he will see the program of interest outputed in checksum format. The cassette is now capable of being used in a system which has no previous program storage--it is an auto-load, and can be placed in a machine immediately after power-up.

The assembler listing of the auto-load cassette package contains other useful information for users of auto-load cassettes. As stated earlier, the CRT routine and the checksum loader are part of every auto-load cassette OSI sells. On cassettes designed for 4K systems or smaller, the CRT simulator routine resides in memory exactly as it is shown here in the assembler listing. By use of this listing, the programmer can customize the CRT routine in programs such as Tiny BASIC, Black Jack, etc., to suit his individual display. Specifically, he can change the variable HOME at line 1020, and the variable LEN at line 1030. HOME is the starting position of the cursor in the CRT routine and can be offset either to the left or to the right of its normal position by changing the 64 to a 63 or a 65, and so on. More importantly, the line length can be changed by changing LEN from 18 downward for a narrower screen or upward for a wider screen. By use of the auto-load package here, the user could re-record his auto-load cassette with these new parameters in it, or he could simply use the 65V monitor each time he loads the cassette to change these locations to suit his particular video display.

Ohio Scientific is now offering the following Auto-Load Cassettes:

1. OSI 6502 4K Tiny BASIC by Tom Pittman, free with 65V Challengers having 4K or more memory and audio cassette interfaces. $12.00 postpaid.
2. OSI 6502 8K BASIC by Microsoft, free with 65V Challengers having 12K or more memory and audio cassette. $52.00 postpaid.
3. OSI Extended Monitor for 8K or larger systems. A "must" for machine language development. $17.00 postpaid.
4. OSI 6502 Assembler by Jim Halverson for 8K or larger systems.

```
1320 0E4B F014          BEQ DONE
1330 0E4D 20B90E  LLOP  JSR INBYT   READ DATA
1340 0E50 8DFFFF  LOCA  STA $FFFF
1350 0E53 20A90E        JSR CKSM
1360 0E56 EE510E        INC LOCA+1
1362 0E59 D003          BNE RT7
1364 0E5B EE520E        INC LOCA+2
1370 0E5E CA      RT7   DEX  DONE?
1380 0E5F D0EC          BNE LLOP
1400 0E61 20B90E  DONE  JSR INBYT   READ CHECKSUM
1410 0E64 CD1E0E        CMP CKH  CORRECT?
1420 0E67 D00B          BNE ERROR
1430 0E69 20B90E        JSR INBYT
1440 0E6C CD1D0E        CMP CKL
1450 0E6F D006          BNE ERROR+3
1470 0E71 4C240E        JMP READ  ALWAYS LOOP
1490 0E74               ;
1500 0E74 20B90E  ERROR JSR INBYT   DUMMY READ
1510 0E77 A945          LDA #'E  TYPE "ERR"
1520 0E79 20530D        JSR CROUT
1530 0E7C A952          LDA #'R
1540 0E7E 20530D        JSR CROUT
1550 0E81 A952          LDA #'R
1560 0E83 20530D        JSR CROUT
1572 0E86 AD1B0E        LDA RESET   SET 65V TO
1573 0E89 85FF          STA $FF  POINT TO
1574 0E8B AD1C0E        LDA RESET+1   READ
1575 0E8E 85FE          STA $FE
1576 0E90 A900          LDA #0  SET KEYBOARD
1577 0E92 85FB          STA $FB  INPUT
1579 0E94 4C43FE        JMP $FE43
1580 0E97               ;
1581 0E97 20B90E  AUTOS JSR INBYT   PICK UP
1582 0E9A 8D220E        STA PNTH    FOR START
1583 0E9D 20B90E        JSR INBYT
1584 0EA0 8D210E        STA PNTL
1585 0EA3 6C210E        JMP (PNTL)  GO!
1595 0EA6               ;
1605 0EA6 205E0F  OUTCK JSR OUTBYT  PRINT BYTE &
1615 0EA9               ;
1625 0EA9 18      CKSM  CLC  CALC. CHECKSUM
1635 0EAA 6D1D0E        ADC CKL
1645 0EAD 8D1D0E        STA CKL
1655 0EB0 9003          BCC NC3
1665 0EB2 EE1E0E        INC CKH
1675 0EB5 60      NC3   RTS
1685 0EB6               ;
1695 0EB6 20A90E  CKIN  JSR CKSM  CALC. CHECKSUM &
1705 0EB9               ;
1715 0EB9 20BC0E  INBYT JSR PACK  INPUT A BYTE
1725 0EBC               ;
1735 0EBC 20DF0E  PACK  JSR UIN  INPUT A DIGIT
1745 0EBF C930          CMP #'0
1755 0EC1 301B          BMI XHIT
1765 0EC3 C947          CMP #'G
1775 0EC5 1017          BPL XHIT
1785 0EC7 C940          CMP #'@
1795 0EC9 3003          BMI LBL
1805 0ECB 18            CLC
1815 0ECC 6909          ADC #9
1825 0ECE 2A      LBL   ROL A
1835 0ECF 2A            ROL A
1845 0ED0 2A            ROL A
1855 0ED1 2A            ROL A
1865 0ED2 A004          LDY #4
1875 0ED4 2A      LOPP  ROL A
1885 0ED5 2E230E        ROL TEMP
1905 0ED8 88            DEY
1915 0ED9 D0F9          BNE LOPP
1925 0EDB AD230E        LDA TEMP
1935 0EDE 60      XHIT  RTS
1945 0EDF               ;
1955 0EDF AD05FB  UIN   LDA $FB05  INPUT FROM UART
1956 0EE2 4A            LSR A
```

$37.00 postpaid.

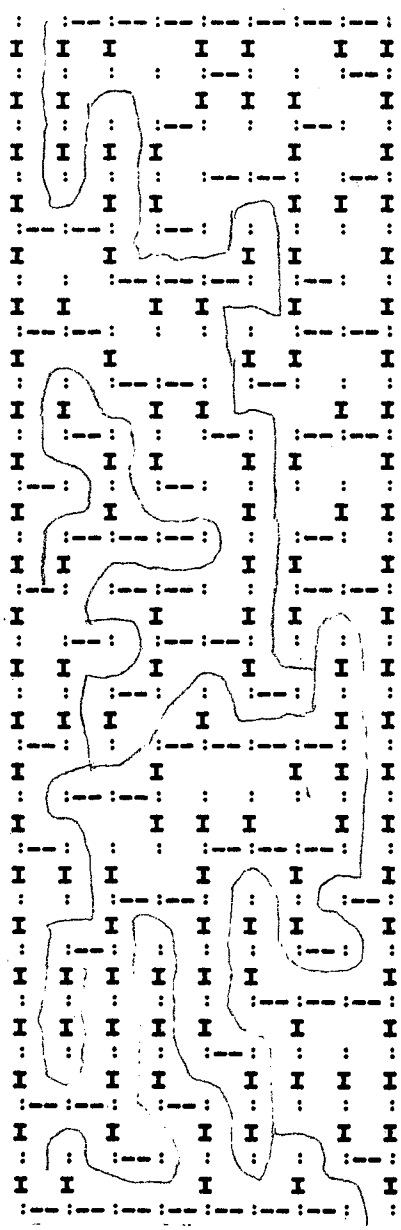
5. OSI "Life" for 4K or larger systems. An exciting real time program. $12.00 postpaid.

6. OSI "Graphics Editor" for 4K or larger systems with the graphics option. Also includes an important plot package for other uses. $10.00 postpaid.

7. OSI Black Jack--a simple game program. $4.00 postpaid.

We are very interested in offering other useful programs for audio cassette systems via a users group exchange in the auto-load format. If you have a program and are interested in distributing it, please write to The Journal.

RUN AMAZIN



```
1957 0EE3 90FA          BCC UIN
1958 0EE5 AD03FB        LDA $FB03
1959 0EE8 8D07FB        STA $FB07
1975 0EEB 4C530D        JMP CROUT   ECHO IT
1976 0EEE          ;
1977 0EEE 48       UOUT PHA UART TO CASSETTE
1978 0EEF AD05FB   WAIT LDA $FB05
1979 0EF2 10FB          BPL WAIT
1980 0EF4 68            PLA
1981 0EF5 8D04FB        STA $FB04
1982 0EF8 4C530D        JMP CROUT TO TV
1983 0EFB          ;
2010 0EFB          ;
2014 0EFB          ;    GENERATE CHECKSUM MAG TAPE
2016 0EFB          ;
2040 0EFB A000     PUNCH LDY #0
2041 0EFD AD210E        LDA PNTL   SET POINTER
2042 0F00 8D3F0F        STA LOOP+1
2043 0F03 AD220E        LDA PNTH
2044 0F06 8D400F        STA LOOP+2
2050 0F09 8C1D0E   AGAIN STY CKL   CLEAR CHECKSUM
2060 0F0C 8C1E0E        STY CKH
2070 0F0F A90D          LDA #$D   OUTPUT CRLF
2080 0F11 20EE0E        JSR UOUT
2090 0F14 A90A          LDA #$A
2100 0F16 20EE0E        JSR UOUT
2110 0F19 AD3F0F        LDA LOOP+1   PAST END
2120 0F1C CD1F0E        CMP ENDL
2130 0F1F AD400F        LDA LOOP+2
2140 0F22 ED200E       SBC ENDH
2150 0F25 B045          BCS OUT   EXIT IF DONE
2160 0F27 A93B          LDA #';   OUTPUT RECORD MARK
2170 0F29 20EE0E        JSR UOUT
2270 0F2C A918     CONT LDA #$18   REC. LENGTH
2280 0F2E AA            TAX
2290 0F2F 20A60E        JSR OUTCK
2300 0F32 AD400F        LDA LOOP+2
2310 0F35 20A60E        JSR OUTCK
2320 0F38 AD3F0F        LDA LOOP+1
2330 0F3B 20A60E        JSR OUTCK
2340 0F3E ADFFFF   LOOP LDA $FFFF   GET DATA
2350 0F41 20A60E        JSR OUTCK
2360 0F44 EE3F0F        INC LOOP+1
2362 0F47 D003          BNE RT6
2364 0F49 EE400F        INC LOOP+2
2370 0F4C CA       RT6  DEX   DONE?
2380 0F4D D0EF          BNE LOOP
2390 0F4F 20550F        JSR PCKSM   PRINT CHECKSUM
2430 0F52 4C090F        JMP AGAIN   LOOP BACK
2436 0F55          ;
2440 0F55 AD1E0E   PCKSM LDA CKH   PRINT CHECKSUM
2450 0F58 205E0F        JSR OUTBYT
2460 0F5B AD1D0E        LDA CKL   LET IT DROP
2470 0F5E          ;
2480 0F5E 48       OUTBYT PHA   PRINT BYTE
2490 0F5F 4A            LSR A
2500 0F60 4A            LSR A
2510 0F61 4A            LSR A
2520 0F62 4A            LSR A
2530 0F63 206D0F        JSR RID
2540 0F66 68            PLA
2550 0F67 48            PHA
2560 0F68 206D0F        JSR RID
2570 0F6B 68            PLA
2580 0F6C 60       OUT  RTS
2590 0F6D          ;
2600 0F6D 290F     RID  AND #$F   PRINT LSD
2610 0F6F 0930          ORA #$30
2620 0F71 C93A          CMP #':
2630 0F73 9002          BCC ELSE
2640 0F75 6906          ADC #6
2650 0F77 4CEE0E   ELSE JMP UOUT
2660 0F7A          ;
```

CONTINUED, at right,
Auto-Load cassette tape generator

Odds & Ends

The OS-65D version 2.0 Disk Operating System is now available and is being shipped as standard with OSI Challenger systems. The 2.0 Operating System is available to current owners of OS-65D 1.5 Version Operating System for $15 per disk. When ordering, please also specify that you need the new manual. Version 2.0 allows data files on drive B in BASIC. It also has special additional I/O commands, a relocator, a tab function in the assembler, and other improvements which make the system easier to use.

* * *

A little-known feature of the editor portion of our standard assembler is that additional lines can be inserted between consecutively numbered lines of test in the assembly. The procedure is to list the source via the PRINT command up to the line just prior to the desired insertion. The user then labels all consecutive lines as line no. 0. Then via a RESEQ (resequence) command these additional lines will be inserted immediately after the last line that was listed. By use of additional software (key poll with echo to 430 Board), or "off line" punching, frequently used subroutines may be listed as all lines zero. They may then be brought into the editor and inserted as required.

* * *

A new improved Tiny BASIC cassette is now available. It features an improved driver package, which yields very reliable cassette I/O (the original Tiny BASIC did not have any screening of illegal characters on cassette input, which made cassette input operations rather touchy under certain equipment conditions). The new improved version also has its drivers immediately above the Tiny BASIC program itself, so that all available memory can be utilized without any adjustment of internal pointers. The driver package is also significantly shorter, yielding more user workspace in a small computer. The improved Tiny BASIC will be shipped automatically when Tiny BASIC cassette is ordered.

* * *

We have heard of several modifications made by users to the model 440 Video board, particularly to get more lines on a screen. We would be

```
2670 0F7A            ;DUMP DATA BLOCK IN 65V LOADER
2671 0F7A            ;FORMAT TO AUDIO CASSETTE.
2680 0F7A            ;LP+1,LP+2=START OF DUMP
2690 0F7A            ;EAL,EAH=END OF DUMP (NOT INCLUSIVE)
2700 0F7A            ;
2710 0F7A A92E       DUMP   LDA #'.   ADDR. MODE CMD.
2720 0F7C 20EE0E            JSR UOUT
2730 0F7F AD950F            LDA LP+2  LOAD START ADDR.
2740 0F82 20B40F            JSR BYTE
2750 0F85 AD940F            LDA LP+1
2760 0F88 20B40F            JSR BYTE
2770 0F8B A92F              LDA #'/   DATA MODE CMD.
2780 0F8D 20EE0E            JSR UOUT
2790 0F90 AECA0F            LDX EAL
2800 0F93 ADFFFF     LP     LDA $FFFF  FETCH DATA
2810 0F96 20B40F            JSR BYTE
2820 0F99 EE940F            INC LP+1
2830 0F9C D003              BNE NCAR
2840 0F9E EE950F            INC LP+2
2850 0FA1 A90D       NCAR   LDA #$D   OUTPUT CR
2860 0FA3 20EE0E            JSR UOUT
2870 0FA6 EC940F            CPX LP+1   END?
2880 0FA9 D0E8              BNE LP
2890 0FAB ADCB0F            LDA EAH
2900 0FAE CD950F            CMP LP+2
2910 0FB1 D0E0              BNE LP
2920 0FB3 60                RTS
2930 0FB4            ;
2940 0FB4 48         BYTE   PHA  OUTPUT A BYTE
2950 0FB5 4A                LSR A
2960 0FB6 4A                LSR A
2970 0FB7 4A                LSR A
2980 0FB8 4A                LSR A
2990 0FB9 20BD0F            JSR DIGIT
3000 0FBC 68                PLA
3010 0FBD            ;
3020 0FBD 290F       DIGIT  AND #$F   OUTPUT A DIGIT
3030 0FBF 0930              ORA #$30
3040 0FC1 C93A              CMP #':
3050 0FC3 9002              BCC UOUTJ
3060 0FC5 6906              ADC #6
3070 0FC7 4CEE0E     UOUTJ  JMP UOUT
3140 0FCA            ;
3150 0FCA FB0E       EAL    .WORD PUNCH   ;END OF 65V DUMP
3160 0FCC            EAH=EAL+1
3165 0FCC 500D        STRT .WORD CRAUD ; START ADDRESS OF DUMP
3170 0FCE            ;
3180 0FCE            ; ---MAINLINE---
3181 0FCE            ;GEN. A TAPE WITH CROUT AND
3182 0FCE            ;KIM-1 LOADER IN 65V FORMAT
3183 0FCE            ;AND THEN DUMPS A SECTION
3184 0FCE             ;     OF MEMORY IN CHECKSUM FORMAT
3185 0FCE            ;AND AN AUTO-START VECTOR.
3186 0FCE            ;THUS THE USER TYPES AN "L"
3187 0FCE            ;AND SOON THE PROGRAM IS
3188 0FCE            ;LOADED AND RUNNING.
3190 0FCE            ;
3200 0FCE ADCC0F     GENT   LDA STRT   LOAD RANGE
3210 0FD1 8D940F            STA LP+1   FOR 65V
3220 0FD4 ADCD0F            LDA STRT+1
3230 0FD7 8D950F            STA LP+2
3240 0FDA 207A0F            JSR DUMP   DUMP UTILITIES
3250 0FDD A92E              LDA #'.    & START LOADER
3260 0FDF 20EE0E            JSR UOUT
3270 0FE2 AD1B0E            LDA RESET
3280 0FE5 20B40F            JSR BYTE
3290 0FE8 AD1C0E            LDA RESET+1
3300 0FEB 20B40F            JSR BYTE
3310 0FEE A947              LDA #'G   GO COMMAND
3320 0FF0 20EE0E            JSR UOUT
3330 0FF3 20FB0E            JSR PUNCH   DUMP BLOCK
3340 0FF6 A924              LDA #'$   LOAD START VECTOR
3350 0FF8 20EE0E            JSR UOUT
```

very interested in hearing of any other modifications that you have made to upgrade the 440 board. If there is interest, we would then take the best combination of modifications and make up a small piggyback board for the 440 to upgrade the board to 64-character display width, etc.

```
0FFB AD0B10            LDA STH
0FFE 20B40F            JSR BYTE
1001 AD0A10            LDA STL
1004 20B40F            JSR BYTE
1007 4C43FE            JMP $FE43   RETURN TO 65V
100A               ;
100A 00      STL       .BYTE $00   ;AUTO START VECTOR
100B 02      STH       .BYTE $02
100C               ;
100C                   .END
```

From object to source code

The 6502 Disassembler

When dealing with machine code programs, it is always more convenient to work with mnemonics than with the actual code. This is, of course, because it is very difficult to remember all the hexadecimal codes and what they stand for, whereas the mnemonics have an intuitive meaning. This is one of the reasons that we use assemblers to assemble machine code. It is always desirable to have an assembled source listing available when working with a particular machine code program. However, this is not always possible or available for various reasons.

The next best thing to an assembled source listing is to have a disassembler available. A disassembler is a program which attempts to convert machine code back into assembler source. It can only do an approximate job of this, because many features of the assembler cannot be derived. For instance, the comment field of the assembler is in no way retrievable from the object code, since it simply represents the programmer's thoughts when he programmed the code. It is also impossible to derive the labels from the program once it has been assembled. Some disassemblers attempt to put pseudo-labels back in programs, such as L1, L2, L3, but these really do little to improve the understanding of the disassembled listing. Therefore most disassemblers simply place absolute hexadecimal numbers in the disassembled source listing where there were originally labels.

The disassembler listing for the 6502 was first published in Interface Age, Vol. 1, No. 10, Sept. 1976, pp. 14-23. The assembler as published can be quickly modified to run on an OSI computer. It is possible to disassemble portions of machine code via the Q command in the Extended Monitor of the OS-65D Disk Operating System. The actual disassembly, listed at the right, turns out to be the NIMB program shown in the 1K Corner [on page 8]. This assembled output does not have line numbers, since no editing is possible or necessary. It does not have a label field or a comment field. Also, all numbers are specified to be

```
Q0200
0200 202003 JSR $0320
0203 A923   LDA #$23
0205 85FD   STA $FD
0207 A5FD   LDA $FD
0209 20F702 JSR $02F7
020C C930   CMP #$30
020E D002   BNE $0212
0210 A920   LDA #$20
0212 8D0A03 STA $030A
0215 A5FD   LDA $FD
0217 20FB02 JSR $02FB
021A 8D0B03 STA $030B
021D A020   LDY #$20
021F AD0803 LDA $0308
0222 91FE   STA ($FE),Y
0224 EE2002 INC $0220
0227 C8     INY
0228 C02E   CPY #$2E
022A D0F3   BNE $021F
022C A908   LDA #$08
022E 8D2002 STA $0220
0231 20E002 JSR $02E0
0234 C95E   CMP #$5E
0236 F0C8   BEQ $0200
0238 C931   CMP #$31
023A 30F5   BMI $0231
023C C934   CMP #$34
023E 10F1   BPL $0231
0240 C8     INY
0241 C8     INY
0242 91FE   STA ($FE),Y
0244 38     SEC
0245 2903   AND #$03
0247 85FC   STA $FC
0249 A5FD   LDA $FD
024B E5FC   SBC $FC
024D F05D   BEQ $02AC
024F 305B   BMI $02AC
0251 85FD   STA $FD
0253 20E002 JSR $02E0
0256 C90D   CMP #$0D
0258 D0F9   BNE $0253
025A A920   LDA #$20
025C 91FE   STA ($FE),Y
025E 18     CLC
025F A901   LDA #$01
0261 C5FD   CMP $FD
0263 F02F   BEQ $0294
0265 1008   BPL $026F
0267 85FA   STA $FA
0269 A904   LDA #$04
026B 65FA   ADC $FA
026D 10F2   BPL $0261
026F 38     SEC
0270 A5FD   LDA $FD
0272 E5FA   SBC $FA
0274 0930   ORA #$30
0276 8D0703 STA $0307
0279 A5FA   LDA $FA
027B 85FD   STA $FD
027D A001   LDY #$01
027F AD0003 LDA $0300
0282 91FE   STA ($FE),Y
0284 EE8002 INC $0280
0287 C8     INY
0288 C009   CPY #$09
028A D0F3   BNE $027F
028C A900   LDA #$00
028E 8D8002 STA $0280
0291 4C0702 JMP $0207
0294 A901   LDA #$01
0296 C5FD   CMP $FD
0298 F033   BEQ $02CD
029A 38     SEC
029B A5FD   LDA $FD
029D E5FB   SBC $FB
029F 85FD   STA $FD
02A1 A5FB   LDA $FB
02A3 0930   ORA #$30
02A5 8D0703 STA $0307
02A8 A5FD   LDA $FD
02AA 10D1   BPL $027D
02AC 202003 JSR $0320
02AF A020   LDY #$20
02B1 AD1603 LDA $0316
02B4 91FE   STA ($FE),Y
02B6 EEB202 INC $02B2
02B9 C8     INY
02BA C028   CPY #$28
02BC D0F3   BNE $02B1
02BE A916   LDA #$16
02C0 8DB202 STA $02B2
02C3 20E002 JSR $02E0
02C6 C95E   CMP #$5E
02C8 D0F9   BNE $02C3
02CA 4C0002 JMP $0200
02CD 202003 JSR $0320
02D0 A022   LDY #$22
02D2 A949   LDA #$49
02D4 8DE5D0 STA $D0E5
02D7 A91A   LDA #$1A
02D9 8DB202 STA $02B2
02DC 10D3   BPL $02B1
02DE FB     ???
02DF 60     RTS
02E0 A903   LDA #$03
02E2 C6FB   DEC $FB
02E4 F00D   BEQ $02F3
02E6 AD01DF LDA $DF01
```

in hexadecimal notation, that is, they are preceded by $. However, the rest of the listing is very similar to the assembler. That is, the first column on the page is equivalent to the program counter location, and is equivalent to column two of an assembler listing. The next field is the actual machine code as it appeared. There is, of course, a missing label field. The next field is the mnemonic for the opcode and the following field is the operand, always specified in hexadecimal.

Another shortcoming of the disassembler is the fact that the disassembler cannot discern between data fields and executable machine code. An example of this occurs in locations 0300 to 0305. Here the program actually has ASCII text in binary form, i.e., 49, 20, 54, etc., which correspond to ASCII characters to be outputed on the screen. The disassembler has no way of knowing this, and so it attempts to disassemble these ASCII characters into mnemonics and operands. It successfully finds a mnemonic for a 49, even though it is supposed to be an ASCII character, but it cannot find a mnemonic or opcode for 54, 4F, etc., and so flags these with question marks.

Another problem may also occur when the disassembler gets back to executable code from a data table. It can fall out of sync by interpreting a data field as a two or three-byte opcode and operand set where actually the second or third byte is the beginning of other legitimate code. The programmer must keep in mind that the disassembler cannot accurately locate data tables, and that when the disassembler leaves a data table and passes back into executable code, it may yield erroneous results for the first few bytes until it falls back into sync with the executable code.

```
02E9 30F5   BMI $02E0
02EB 48     PHA
02EC AD01DF LDA $DF01
02EF 10FB   BPL $02EC
02F1 68     PLA
02F2 60     RTS
02F3 85FB   STA $FB
02F5 10EF   BPL $02E6
02F7 4A     LSR A
02F8 4A     LSR A
02F9 4A     LSR A
02FA 4A     LSR A
02FB 290F   AND #$0F
02FD 0930   ORA #$30
02FF 60     RTS
0300 4920   EOR #$20
0302 54     ???
0303 4F     ???
0304 4F     ???
0305 4B     ???
0306 20324E JSR $4E32
0309 3D3233 AND $3332,X
030C 2C484F BIT $4F48
030F 57     ???
0310 204D41 JSR $414D
0313 4E593F LSR $3F59
0316 594F55 EOR $554F,Y
0319 204C4F JSR $4F4C
031C 53     ???
031D 4520   EOR $20
031F 20F8A9 JSR $A9F8
0322 C585   CMP $85
0324 FEA9D0 INC $D0A9,X
0327 85FF   STA $FF
0329 A0FF   LDY #$FF
032B A920   LDA #$20
032D 91FE   STA ($FE),Y
032F 88     DEY
0330 D0FB   BNE $032D
0332 60     RTS
```

HARDWARE

A Preview of our new CPU Boards

Ohio Scientific is coming out with two brand new CPU boards as part of a general upgrade of our systems product line to the 500 series. The earliest 500 series board was quietly introduced a few months ago as the model 520 16K RAM board. Because of continuing shortages of supplies of 4K RAM chips for the 520 board, no major announcement was made of this new board. We are currently working on several other 16K and 64K RAM boards to insure a large volume supply of large memory boards independent of manufacturer shortages. The main board of the 500 line was introduced at the MACC show in Cleveland and at the National Computer Conference in Dallas in June. That same month, the model 500 board was announced in EDN. It completely replaces the old model 400, which is now available only by special order. The 500 is now in production and being shipped on a regular basis.

Designed to be the general systems workhorse board for the OSI line for the next few years, the 500 represents the state of the art in single-board computers. It is fully compatible with all 400 series boards and the OSI 48-line bus, thus no older products now become obsolete. The model 500 can accept eight 2K x 8 2616-type mask ROMs, which contain our super-fast 8K BASIC by Microsoft. The model 500 also has provisions for 4K of 2102-type memories, an ACIA-based serial interface, which can be populated for RS-232 or 20-m.A. current loop at five different baud rates, which are also jumper selectable.

The 500 can optionally have a PIA-based 16-line parallel I/O port, part of which is used for a 256K memory management unit controlling two additional address lines on the bus (A16 and A17). The 500 can also accept up to three 1702-type PROMs and can be populated for one, two, or three of these PROMs by partially or fully decoding the address base at FD, FE, and FFXX. The model also has full bus pullup resistors on board. The 500 can be used with our existing 65A, 65V, and floppy disk bootstrap PROMs. It is effective as a stand-alone computer which uses 8K BASIC with 4K of workspace, and is thus functionally equivalent to a 12K computer. It is offered fully assembled and tested, minus the PIA port, for $298, including the BASIC and ROM.

The board has many subtle points that will make it even more useful than is apparent so far. For example, there are provisions on the board for additional data direction and wait state diodes and address decoding, so that the 500 board may be used as an accessory board in OSI computers. That is, the 4K of RAM memory can be addressed for some location other than location 0, and other ports can be placed at other addresses, so that the 500 board can be used as a combination ROM, RAM, EPROM, PIA, and ACIA board, along with another 500 or other board that has a processor on it. This allows for extremely high system density. The four sockets used for the 2616 mask ROM are programmable in function by two 16-bit dip locations, which are normally set up for 8K BASIC. By changing jumpers in these dip locations, these four sockets can be changed to accept up to thirty different devices including the popular 2704, 2708, and 2716 UV erasable EPROMS. Thus the board can be populated with 2K, 4K or 8K of user-programmed EPROM, instead of for BASIC. This in conjunction with the PIA makes this board extremely useful for dedicated industrial control applications.

Other features of the board include 256K addressing, or two additional address bit pager. This feature, along with an Expanded Monitor PROM Set which will soon be available, will allow up to four independent users in a 500-based system by flipping upper memory lines. The user programs can be switched in and out, thus allowing ultra-fast interrupt service, and memory partitioning of users. The 8K BASIC ROMs also include a complete CRT routine and audio cassette drivers, so that by changing to a different support PROM, the board can support BASIC in conjunction with a 440 video board. The system can support a 430 based cassette I/O board in either serial or video modes. Extra control characters in BASIC allow storage and retrieval of BASIC programs, even when the serial baud rate is lower than the cassette baud rate, by invoking printing or non-printing operations.

The 500 board can be selectively populated to emulate any 400 board, and is completely compatible with systems using the 400 board. Therefore, the 500s are now being used to fill existing 400 orders in bare boards, kits, and fully assembled products. The model 500 is available as a bare board for $39.00; as a 504A and 504V kit with 1K RAM (equivalent to 414A and 414V, at the same price), fully assembled with 8K BASIC in ROM and a serial port.

When the model 500 is placed in a Challenger case, the computer is called a Challenger II, so that all Challenger II's are based on 500 CPU board. The Challenger II's are available in a serial form as a Challenger IIS, in video form as a Challenger IIV, and are also to be available in a four-slot version with a captive keyboard as a Challenger IIP. This special small computer is similar in appearance to the Processor Tech Sol-20, and in cost and performance to the Commodore PET.

500 boards and Challenger II's can also be used with disk drives, of course. However, the 8K BASIC in ROM is not suitable for use with disk, as it does not have the disk I/O commands. Furthermore, BASIC can be quickly pulled in from disk instead of ROMs when the disk drive is present. We offer Challenger II's without the BASIC ROMs and with the floppy disk instead. Check the price list for details. We will be covering 8K BASIC ROMs in more detail in future issues of the journal.

The other exciting new CPU board which Ohio Scientific will be introducing to both Computermania and Personal Computing 77 is the model 510. This CPU board is the most technically advanced central processor available today for small computers. It comes standard with a 6502A, 6800, and Z-80 microprocessor. The board also includes a serial port which can be configured for RS-232 and 20-m.A. current loop with crystal controlled baud rate generation from 110 to 19,200 baud, and provisions for three monitor PROMs for the 6502 and one for the 6800. These PROMs are 1702-type devices.

The three processors are actually separate entities on the board. Only one processor can be active at a time, and all control lines and address and data lines of the other two processors are in a tristate condition when they are not selected. In its most basic form, a user switches processors by holding in the reset switch and rotating a switch at the back of the computer. Memory is preserved during switching operations, so that for instance, the 6502 Disk Operating System can be used to bring a Z-80 program into memory, and then the user can switch to the Z-80, and run the Z-80 program.

We are also offering an optional software processor switch and 1-megabyte pager. The software processor switch allows programmed processor switches, so that one can utilize 6800 and Z-80 programs in conjunction with the 6502 DOS in a totally automatic form. The 1-megabyte pager provides an additional four address lines--A16, A17, A18, and A19--thus allowing up to 16 users on a system, or other uses for partitioned memory.

The 510 CPU board is available only in two forms: as a fully assembled PC board with or without the software switch option, or as the processor in the Challenger II system. The Challenger III system is available only with floppy disk. Challenger III or model 510 software will be offered only on floppy disk; however, the 6502 portion on the 510 board is fully compatible with existing 500 and 400 series software.

Ohio Scientific will be offering a program for present Challenger I owners to trade in their current 400 CPU board for a new 510 CPU board. This trade-in program applies only to OSI-assembled 400 CPU boards, however. Details of this plan will be in the next Journal.

The 510 system also allows full DMA, or complete processor shutdown, so that other 500 series processor boards may be added as an extension of the 510 board. We are currently planning a Micronova-type CPU board, based on the new Fairchild chip, which will be introduced this winter. To complement the 510 philosophy, the older 460Z has been upgraded to allow tristate operations on both its input and output bus, so that the 460Z, now called the 560Z, can be used in conjunction with the 510 boards on both its input and output bus. The 560Z, of course, will be able to use a 500 or 400 CPU board on its input side, as originally specified. The 510's functions are not to be confused with the 460Z, now 560Z. The 510 allows only one processor to be operational at a time, whereas the 460Z allows true multiprocessing, with one host CPU running in conjunction with one processor on the 460Z. Furthermore, the 460Z is the only practical approach to implementing the PDP8-equivalent 6100 economically. 510 boards are scheduled for delivery in early September 1977. We will be covering in greater detail some of the aspects of the 500, 510, and 560Z in the new products section of future issues of the journal.

Current Price List

400/500 Boards and Kits

Model 500 CPU Board	39.00
Model 504A Serial CPU Kit (replaces 414A)	149.00
Model 504V Video CPU Kit (replaces 414V)	134.00
Model 420C Memory Board	35.00
Model 422 Memory Board and Parts (continuation of special) 2MHx low power	99.00
Model 427 Memory Board and Parts 1MHz Medium Power	79.00
Model 430B I/O Board	35.00
Model 440B Video Graphics Board	35.00
Model 446 Video Graphics Board and Parts	129.00
Model 450 PROM Board	35.00
Model 455 PROM Board	35.00
Model 460Z, Z-80, & 6100	125.00
Model 475 Kit	749.00
Model 480 Backplane Board	39.00
Model 495 Prototyping Board	29.00
Model 498 Card Edge Extender Board	29.00

The 500 board and 504 kits replace the popular 400 series. The popular 422 and 475 kits have been added permanently.

Monitor PROMs

65A for Serial 6502	29.00
65V for Video 6502	29.00
68A for Serial 6800	29.00
68V for Video 6800	29.00
65-500F2 Floppy Disk Bootstrap PROM	29.00

Parts

2513 Character Generator	12.00
8T26 Buffers	3.00
6850 ACIA	15.00
6520 PIA	10.00
S1883 UART	10.00
K-1 Connector Kit	3.00
1408L8 D/A Convertor	10.00

Challengers

Old Products:	
C-SIT Challenger* System with Microterm Terminal and Monitor	2,599.00
C-S1 Challenger* System without Terminal	2,099.00
C-S2 Challenger* Video System	2,499.00
* Delivered as Partially Populated Challenger II	
C-D1 Single Drive Floppy Disk	990.00
C-D2 Dual Drive Floppy Disk	1,590.00
New Products (or New Prices):	
C2-0 500 Board Fully Populated	298.00
C2-1 500-1 (4K RAM & Small Cabinet)	429.00
C2-8S 500-8 or Challenger II (4K RAM)	629.00
A1 Add Memory Management & Parallel Port	50.00

Challengers (Cont.)

A2 2MHz Operation	50.00
A3 Configure for Disk Use (must have 16K RAM minimum) Add Disk Bootstrap and Delete 8K BASIC ROMs	-30.00
C2-8V Challenger IIV (4K RAM)	750.00
A4 Add Serial Port	50.00
A1 Add Memory Manangment & Parallel Port	50.00
A3 Configure for Disk Use (16K Minimum) Add Disk Bootstrap and Delete 8K BASIC ROMS (subtract from above)	-30.00
A5 Add 128 x 128 Graphics	150.00
C2-4P Challenger IIP--Direct competition to Commodore PET includes 4-slow backplane, 8K BASIC in ROM, 4K RAM, Special 32 x 64 Display, Audio Cassette Interface, and Keyboard in Special Enclsoure	598.00

INTRODUCTORY PRICE ONLY

Add Graphics to C2-4P	50.00
Add 4K RAM Plus Fan to C2-4P	159.00

Challenger IIV replaces Challenger 65V and has 440 Video Board. Challenger IIP is a four-slot computer similar in appearance to the Sol-20 which is designed for direct competition with the Commodore PET.

Challenger Accessories

CM-1 4K 1MHz Challenger Memory for use with ROM BASIC	125.00
CM-2 4K 2MHz Low Power Memory	149.00
CM-3 16K 1.5MHz Ultra Low Power Memory	596.00
CA-6 Challenger Audio Cassette	99.00
CA-7 Fully Populated 430B Board	399.00
CA-8 460Z Subsystem	1,190.00

The CM-1 Memory Boards are for use with ROM BASIC machines, since BASIC in ROM can run only at 1MHz.
CM-2 and CM-3 Memories should be used in large systems for low power dissipation.

C3-8 Challenger III--6502A, 6800, Z-80, 16K RAM Crystal Control, Serial Port, and Floppy Disk Bootstrap, ALL Standard	1,295.00
A-100 1 Megabyte Memory Management Software Processor Switch Swappable RAM, and Additional Parallel Port	150.00
A-101 Video Board and Video PROM	189.00
C3-0 510 CPU Board (for System Upgrades)	359.00

(A100 applies) Factory Direct Trade-ins are available for Challenger owners.

The Challenger III comes fully equipped for disk, but the price does not include the disk drive(s) C-D1 or C-D2. Challenger IIIs with the A-100 option will take 15 to 30 days longer to deliver.

Challenger Accessories

Keyboard, Enclosure & Cable	149.00
Cassette Recorder & Cables	59.00
Video Monitor	159.00
OKI Data Model 110 with Cable & Interface	1,900.00
OKI Data Model 22 with Cable & Interface	2,900.00

Software: Paper Tape, Cassette, or Diskette

S-1 Introductory Software Package 12S-1 (paper tape only)	20.00
S-2 Assembler/Editor	35.00

Software (Cont.)

S-3 8K BASIC	50.00
S-4 Extended Monitor	15.00
S-5 Life	10.00
S-6 Graphics Editor (no Paper Tape)	8.00
S-7 Tiny BASIC	10.00

Add $15.00 for OS-65D Diskette. Multiple diskette copies can be purchased directly by end user at $15.00 each.

Sample orders

If you wish, for example, to order a Challenger with Dual Floppy. a Serial Port (RS-232), and 20K or RAM Memory, your order should look like this:

C2-8S (Challenger II Serial)	629.00
A3 (Configure for Disk)	-30.00
CM-3 (16K RAM Board)	596.00
Serial Port (Standard with "S" Series)	N/C
C-D2 (Dual Floppy)	1,590.00
8K BASIC and OS-65D Disk Operating System (Standard with Disk)	N/C
TOTAL (retail)	$2,785.00

If your order consists of an 8-slot computer similar to the PET, but, expandable, and you have your own TV set-tape recorder, your order would look like this:

C2-8V Challenger IIV (includes BASIC in ROM and Video Display Board)	750.00
CA-6 Cassette Interface	99.00
Challenger Keyboard	149.00
TOTAL (retail)	$998.00

If you are ordering a top-of-the-line system for business software development, i.e., Challenger III, 48K RAM, Dual Floppy, and Line Printer, your order would look like this:

C3-8 Challenger III (includes 16K RAM, Serial Port)	1,295.00
Two CM-3 16K RAM Memory Boards @596.00	1,192.00
C-D2 Dual Floppy	1,590.00
OKI Data Model o-22 Line Printer	2,900.00
TOTAL (retail)	$6,977.00

Ohio Scientific 11679 Hayden Hiram, OH 44234
SMALL SYSTEMS JOURNAL