

In this issue...

Editorial.....	2	Salomon Lederman
Correction.....	3	Peter Schreiber
Troubleshooting Your Computer....	4	Klaus Ernst
Color Monitor Conversion.....	6	Ugo Re
CEGMON Terminal Program.....	7	Mike Bassman
Errors & Error Messages.....	8	Peter Schreiber
OSI-TEMS Index for '80.....	12	Mike Bassman
6502 Machine Language, Class 3....	18	Salomon Lederman

The Challenger Users Group of New York meets the first Thursday of the month at:

Polk's Aristocraft
314 Fifth Avenue
New York, NY 10010

President-	Ugo Re
Vice-President-	Warren Modell
Treasurer-	Daniel Schwartz
Secretary-	Thomas Cheng

Dues: \$10 per year, \$5 for students

OSI-TEMS staff:

Mike Bassman
Thomas Cheng
Mike Cohen
Salomon Lederman
Terry Terrance
Larry Thaler

This month's issue has been prepared
for publication by Salomon Lederman

All of the material in this magazine is the sole property of the writer. This includes all programs, articles, and any other material. Nothing may be copied, sold, or in any way distributed without written permission of each individual author.

IT'S BEEN A GOOD YEAR..... (Editorial)

Happy New Year. 1980 has been a great year for most of us involved in the OSI small systems, and 1981 promises to be a better year yet. In 1980, much more than in past years there has been a great surge of interest in the Challenger computers. Many people have gotten into the act, in both hardware and software. The quality of products is greatly improving.

In this past year people have begun to realize that us small system owners make up a sizable market. We want new and better hardware and software. We are finally getting the attention we deserve. This year alone there have been nearly a dozen enterprising persons competing in the software market. And there has been a great interest in hardware mods and add-ons for our micros.

All of this competition is potentially beneficial for various reasons. The more software that is available the lower the prices should be, hopefully. If everyone and his brother programs then perhaps someone will sell quality software inexpensively enough that noone should pirate copies. I don't realistically expect prices to drop but perhaps they will not go up as quickly as they've been rising. Also if there is a greater line of software available then users will be able to select the better software. Often one will buy a program because it's the only one available for his system. Wouldn't it be nice to have to choose between several chess programs?

What this all means for our club is that the New York OS Users Group will have an important role in 1981. Hopefully club members will keep us all up to date on the newest developments. We should all know what is worth getting and what isn't. So in this year more than ever we depend on all club members to let us all know through this publication what is available and its worth.

H A P P Y N E W Y E A R

Corection

Peter Schreiber

In the November issue of OSI-tams I had a program called DEPTH CHARGE which you might have tried to use, but with not much luck. Because I failed to type out the program myself, the program was out with a few errors. The following is the correct machine language part of the program:

```
0F00 A2 00 86 FE A9 D0 85 FF AD 00 A9 20 91 FE C8 D0
0F10 FB E6 FF E8 E0 08 D0 F4 A9 00 8D 44 DE 85 E0 AD
0F20 00 A9 87 99 40 D4 C8 C0 20 D0 F8 AD 00 B9 C0 0F
0F30 F0 0C 99 00 D1 B9 CA 0F 99 B1 D1 C8 D0 EF A9 01
0F40 8D 12 02 A9 80 8D 00 DF A9 0E 85 F7 A9 D4 85 FB
0F50 AD 00 A9 B5 91 F7 C8 A9 B6 91 F7 A9 63 85 08 A9
0F60 8F 85 0C AD 00 DF 85 F9 C9 80 D0 1D AD 00 D4 C9
0F70 B5 F0 16 A9 00 85 E0 A8 A9 B5 C6 F7 91 F7 A9 B6
0F80 C8 91 F7 A9 20 C8 91 F7 60 A5 F9 C9 40 D0 1E AD
0F90 1F D4 C9 B4 F0 16 A9 01 85 E0 AD 00 A9 20 91 F7
0FA0 A9 B3 C8 91 F7 A9 B4 C8 91 F7 E6 F7 60 A5 F9 C9
0FB0 02 D0 F9 A9 00 8D 12 02 A9 01 8D 44 DE 4C 00 00
0FC0 53 43 4F 52 45 3A 20 20 20 00 54 49 4D 45 3A 20
0FD0 31 32 30 24 24 24 24 24 24 24 24 24 24 24 24
```

There was also one error in the Basic part of the program and that should read as follows:

```
110 POKE11,0:POKE12,15:IFLE<10RLE>3THEN100
```

Troubleshooting Your Computer, or

A Case of Incredible Beginner's Luck

by Klaus Ernst

'This is a true story!' (David Brenner)

One day I turn on my ClP and I could not believe what I was seeing. I coldstarted my system. Everything was coming up real slow. "D/C/W/M" and "MEMORY SIZE" was coming up slowly as if I were in the save mode. I turned the machine off, then back on again. No change! I thought "oh, oh! I finally broke it! Here goes \$100. or more for a set of ROM chips.

Poking 517,0 is supposed to turn off the save mode. I tried that. No good. PRINT PEEK(517) came up with 64. Could it have been a bad memory chip? But which one?

64 is 40 in hex or 0100 0000 in binary. For each 1k of memory two 2114 chips are requitred, one for the low order bits (D0-D3) and one for the high order bits (D4-D7). Since location 517 is under 1k and the troublemaking bit 6 is the second highest, the bad chip had to be U45. (See note 1 for an explanation.) A memory test was published in The Small Systems Journal (see listing) that came through the mail. (Since this issue is neither dated nor does it have an issue number, I can only refer you to the page number, which is 54.)

This test proved useless in my case because it starts testing at 1k, or 1025 (a bit over 1k). I worked around this by swapping U45 and U52. This was an arbitrary decision; I could have chosen any other chip, except U31 which is the other 1k chip.

To my delight swapping the chips licked the problem. Normal speeds were now attainable at coldstart. Now all I had to do to reassure myself was to run the memory test with the bad chip in the high order spot of the 8k pair. I got the message:

LOCATION 7685 WAS 64, NOT 0

The chip was proven bad. 7685 is 7x1024+517 or 7k+517.

What does this all mean to you? If your machine does strange things then you should run a memory test(note 2). Of course, things may be so bad that your machine can't even do that anymore, but it's worth a try. You can also try to switch memory chips U31 and U45. Your first 1k of memory, where most of your flags and vectors are located should thus be checked.

Happy New Year. I hope you'll never need this.

(See notes on next page.)

Note 1: How to Find RAM Chip Locations on your Superboard

The RAM chips are shown on sheet 3 (of 13) of the schematics in the back of your users manual after page E4, also in the SAMS service manual pages 10,11,12. (ed. note: in newer C1 models there are no included schematics, you must purchase them seperately.)

$\overline{RS0}$ s the chip enable for the first 1k. This guides you to U31 and U45. The physical location is shown on the last page of the schematics(before the memory map). If you want to understand the address decoding then look at sheet 2 (of 13). $\overline{RS0}$ is connected to pin 7 of U22 (74LS138). Pin 7 is labelled Y7. Pin 15 is labeled Y0 and selectline $\overline{RS7}$ is connected to it. The \overline{RS} lines are numbered backwards (in respect to the Y outputs of U22), because the inputs (A,B,C) are inverted datalines ($\overline{A10}, \overline{A11}, \overline{A12}$).

Chip U22 is enabled by $\overline{\phi 2}$ (pin 5; G2B) and Y0 (low) of U23 (pin 15) connected to G2A of U22 (pin 4). This means that for the first 8k of memory address lines A12,A11,A10 are always low also.

	A→15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0000 _H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
03FF _H	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

0000-lowest address (0000 in decimal)
03FF-highest address (1023 in decimal)

Note 2:

To run the memory test per listing you have to add:

ϕ Y=xxxx

for 4k...Y is set to 4095

for 8k...Y is set to 8191

for 16k..Y is set to 16383

There are two machine language memory tests in this month's Small Systems Journal. (MICRO 32: 82-84, Jan. 81)

```
2 P=1025
5 Q=255
10 FORX=P TO Y
12 POKE X,Q
13 NEXTX
20 FORX=P TO Y
22 Z=PEEK(X)
23 IF Z<>Q THEN GOSUB 100
25 NEXTX:PRINT"PASSUSING ";Q
26 IF Q=0 THEN 5
30 Q=0:GOTO 10
100 PRINT"LOCATION";X;"WAS";Z
NOT":Q:RETURN
OK
```


Converting Your Color TV to a Color Monitor

by Ugo Re

I purchased and installed the monitor conversion kit in a Boshie TC700 color TV. The new monitor provides better resolution than the RF modulator that I was previously using.

The monitor conversion kit comes completely assembled and includes adequate instructions on how to install it in a TV. When I ordered the kit I had to supply a schematic of my set.

Installation involves cutting some traces and replacing some resistors in the TV, and hooking up about ten wires from the monitor board.

While this is a good kit that I found very easy to install, I would not recommend it to anyone who is not familiar with TV repairing and the use of schematics.

The total cost of the kit with the audio input option was \$155.

The kit is being offered by Video Marketing, Inc.

780 Lorraine Drive

PO Box 339

Warrington, Pa. 18976

(215)343-3000

The ad for it states (and I quote);

"Color Video Monitor Conversion Kit -MCK100...converts a color TV receiver into a combination professional quality color video monitor and a TV receiver. A video monitor provides a significant improvement in picture resolution, color quality and stability as well as a substantial reduction in the susceptibility to interference and noise, as compared to R.F. modulator techniques for video displays.

Specifications...use with SONY and other TV sets. Supply us with the SONY model number. Other sets, supply the schematic diagram. Uses latest technology to provide electrical isolation of video input from hot chassis or AC line of TV set. 1500 volt isolation. Accepts standard composite video input: 0.5 to 2.5 volts p-p. 100% solid state circuitry. Bandwidth: Converted Video Monitor-approx. 4 Mhz, horizontal resolution-300 lines. MCK Circuitry: 8 Mhz. Input impedance: 75 Ohms. Audio Input Option: For use with video tape recorders or computers, signal input: 0.5 v. p-p, high impedance \$15."

CEGMON Terminal Program

by Mike Bassman

If you have both CEGMON and a modem then you have trouble since the two are not completely compatible. SYMON and CEGMON differ in vector and keyboard locations. Your present terminal program will probably not work at all with CEGMON. Here is a terminal program that does work with CEGMON. Following is a brief explanation of the program.

line(s)	explanation
10-40	CEGMON keyboard routine that returns a character value even if no key is detected
45	Check if modem detects an incoming character; if so then gosub 100
50-60	Gosub keyboard routine. If character is a rubout then change ASCII 95 to a standard backspace (ASCII 8, control H)
100-110	Print incoming character

*****NOTE*****

This program is set up for half duplex, even parity, 7 data bits, 1 stop bit.

```
5 POKE530,1
10 FORX=1TO80:M=PEEK(64767+X):POKE575+X,M:NEXT:FORX=1TO3:READM
20 POKE633+X,M:NEXT:FORX=1TO15:READM:POKE655+X,M:NEXT:DATA76,203,253
30 DATA141,19,2,76,110,253,32,64,2,163,169,0,76,193,175
40 POKE11,150:POKE12,2:A=61440:POKEA,3:POKEA,9
41 FORK=1TO32:PRINT:NEXT:PRINT"Miketerminal ready":PRINT
45 IFPEEK(A)AND1THENGOSUB100:GOTO45
50 P=USR(X):IFP=0THENF=1:GOTO45,
52 IFF=0THEN45
53 IFP=27THENPOKEA,96:POKEA,3:POKEA,3:GOTO45
55 IFP=95THENP=8
60 F=0:POKEA+1,P:GOTO45
100 C=PEEK(A+1):IFC=8THENC=95
101 IFC=7THENRETURN
110 PRINTCHR$(C);:RETURN
OK
```

A Look at Errors and Error Messages

Peter Schreiber
1609 Washington Avenue
Seaford, N.Y. 11783

Programming errors are a problem that every programmer encounters while solving problems on the computer. Every computer system will make checks during execution of code to prevent error because a program either tells the computer something that it simply does not understand, or something that it is incapable of doing with present data. The OSI computer, as well as other micro-computer systems, give only a simple code as error messages from which the programmer must look up what he has done wrong. The OSI computer has a simple two character code for its 18 different error messages.

When the Basic interpreter finds an error during the execution of a program, it breaks from its present routine to the error message printer routine located at \$A24E. Before this break is made in the present routine, an error code corresponding to the appropriate error is stored in the X-register. The X-register contains an offset value which corresponds to the appropriate error in the Basic error table, located at \$A164 to A185. The characters located at \$A164 + the contents of the X-register and at \$A164 + the contents of the X-register + 1, are printed on the screen as the error code of the error just made. As you can see, the error printed directly depends on the offset value that is put into the X-register. The error message routine then prints the word "ERROR" after the code, and then depending on whether you were executing a program or not, the appropriate line number. This means, if you were executing code in immediate mode, you will not get a line number in the error message.

Now that the computer has printed the error message on the screen, you say "WHAT?", and break out your OSI manual to find out what just happened. This is not fun! You have to spend time looking up the error in the error table, while you could be repairing the program and running it again.

In an article by Dan Schwartz (see OSI-tems, V.II, No.9, p.14), the problem was cut down somewhat by eliminating that awful graphics character and replacing it with the original second letter. That was great; but when I found that I still did not know what a "TM" error message was, I had to break out my OSI manual again, and find out what had just happened.

Listing 1 is a machine language program which produces the ultimate error messages (for OSI anyway). It is located in the free memory at page 2, of course. Now when I try to execute:

```
10 LET TM$ = 36           , I get the error message:  
TYPE MISMATCH ERROR IN 10  
Ready
```

and I respond to this message with a simple "OOPS!", which is a great improvement from trying to remember where I put that manual.

For this luxury to our system we, of course, have to pay; not with dollars, but with bytes. The reason that Basic does not have such informative error messages is probably because it costs too many bytes.

The messages that I used consume 1 page of your available RAM. That is expensive, but it is a luxury you can afford if you don't happen to be using that extra RAM. I have located the messages at the very top of available memory, which is \$1F00 to \$1FFF of my 8K system. For any other size system, the messages can be moved with a few modifications to the program in Listing 1.

<u>Location</u>	<u>Machine Language</u>	<u>Mnemonics</u>
0222	A2 FE	LDX #\$FE
0224	9A	TXS
0225	4C 74 A2	JMP \$A274
0228	AE 40 D7	LDX \$D740
022B	E0 3F	CPX #\$3F
022D	D0 3A	BNE \$0269
022F	AD 42 D7	LDA \$D742
0232	48	PHA
0233	6D 41 D7	ADC \$D741
0236	85 F0	STA \$F0
0238	68	PLA
0239	29 01	AND #\$01
023B	65 F0	ADC \$F0
023D	A2 00	LDX #\$00
023F	E8	INX
0240	DD FF 1E	CMP \$1EFF,X
0243	D0 FA	BNE \$023F
0245	A0 00	LDY #\$00
0247	B9 43 D7	LDA \$D743,Y
024A	99 AE 02	STA \$02AE,Y
024D	C8	INY
024E	C9 5F	CMP #\$5F
0250	D0 F5	BNE \$0246
0252	A9 00	LDA #\$00
0254	99 AD 02	STA \$02AD,Y
0257	A9 42	LDA #\$42
0259	8D 00 02	STA \$0200
025C	A0 1F	LDY #\$1F
025E	8A	TXA
025F	20 C3 A8	JSR \$A8C3
0262	A0 02	LDY #\$02
0264	A9 AE	LDA #\$AE
0266	20 C3 A8	JSR \$A8C3
0269	A0 1F	LDY #\$1F
026B	A9 F6	LDA #\$F6
026D	4C C3 A8	JMP \$A8C3

Listing 1.

To use the program, you also need to type in all of the error messages. It is also not fun to type in ASCII characters using the OSI graphics manual. Listing 2 is a Basic program which will poke all of the characters into memory. The set of messages are placed into string variable form, converted into ASCII characters by the Basic ASC function, and poked directly into the last page of memory.

LIST

```

10 A=7936:REM Dec 7936 = Hex 1F00
20 FOR Y=1 TO 19:READ B,A$:POKE A,B:A=A+1:FOR X=1 TO LEN(A$)
30 P=ASC(MID$(A$,X,1)):POKE A,P:A=A+1
40 NEXT X:POKE A,A:A=A+1:NEXT Y
100 DATA 10,DOUBLE DIMENSION
110 DATA 12,UNCTION CALL
120 DATA 15,LEGAL DIRECT
130 DATA 22,EXT-FOR
140 DATA 21,UT OF DATA
150 DATA 31,UT OF MEMORY
160 DATA 39,VERFLOW
170 DATA 35,YNTAX
180 DATA 28,ETURN-GOSUB
190 DATA 43,DEFINED STATEMENT
200 DATA 224,DIVISION BY ZERO
210 DATA 19,ONTINUE
220 DATA 34,ONG STRING
230 DATA 40,UT OF STRING SPACE
240 DATA 41,TRING TEMPORARY
250 DATA 36,YPE MISMATCH
260 DATA 29,DEFINED FUNCTION
270 DATA 24,UFFER
280 DATA 13,$Ready$$
290 POKE A-9,10:POKE A-3,13:POKE A-2,10
300 END

```

OK

Listing 2.

Two last details must be completed before everything is set to go. First, you must save the top of memory so that Basic will not write over it with string storage. This is because Basic uses this space to store strings and we want to protect that from happening. To save the top most page of memory for these messages, do the following: (1) break to the monitor, and type in ".0086/", (2) examine the value stored in that address, and (3) decrease the value of the address. For example, if the contents of \$0086 is \$20, replace it with \$1F. This move will change the upper limit of memory down 1 page saving the messages. Second, you must change the warmstart and message jumps to point to the program in listing 1. Do the following:

<u>Location</u>	<u>Old</u>	<u>New</u>
0000	4C 74 A2	4C 22 02
0003	4C C3 A8	4C 28 02

If you have other than an 8K system you are going to have to make a few changes. For a 4K system do the following:

- (1) Change variable A in line 10 of listing 2 to 3840.
- (2) Break to the monitor, type ".0086/00"
- (3) Change these locations of listing 1:

0242	1E	0E
0258	42	02 ; on C1P only?
025D	1F	0F
026A	1F	0F

Once you have put everything into its place in memory, I suggest that you save it on tape. It takes long enough to put everything in the first time, and nobody wants to do it again. To save the code see "Cassette Save/Hex Memory Dump" article in the November issue of OSI-tems (V.II, No.9, p.28). After you have done this, you can get rid of the program in Listing 2 since all the messages have been saved on tape.

After this is done you are ready to try out the program by making a few errors. In doing this, you will also find two extra features. The first is the elimination of the "OUT OF MEMORY" error you get when you warmstart the system. I was glad to see that go! The second is the "OK" message being replaced with "Ready". I don't know which you would like, so with a few modifications you can get the old message back.

Unfortunately the story does not end here. Now that you know what the error is that you have made, you have to fix it. If there is one thing you will hate more than anything else about programming, it is debugging. You might be able to correct your error right after you get your error message and say "OOPS!", but then again it might be 85 "OOPS!"s from the first. This is the case with logic errors. A syntax error can usually be corrected by retyping the line, but what do you do when you find no errors? This is where the Basic "STOP" instruction comes into use.

The STOP instruction is used by the programmer to set break points in a program. When Basic encounters the STOP, all pointers and variables are saved and program execution is halted by the "BREAK IN XXXX" message, where XXXX is the line number Basic last encountered. Now that the program has halted, the programmer is free to examine any variable value at that point in the program. This is of great importance in locating logic errors in your Basic program, because you begin to cry if everything does not go as you have planned. Try the following routine when locating errors in your program.

- (1) Set break points in your program at key locations. If you have an idea of the location of the error, put some STOP instructions there first, although this does not mean your right.
- (2) Execute the program using simple data. This means use data so you will be able to calculate equations on paper and check them against the computer, as the computer encounters the break points. You have now become the computer running the program.
- (3) If nothing has come up at that point, then use the CONT instruction to continue execution. From here on in, it is a matter of repeating the process again and again. You might pass over the error several times before you see any light, and even then there is still others you may have missed.
- (4) If you can not find the error after some time, don't just sit there because paralysis will set in after the first five minutes. Make changes in the program that might not seem to produce much worth. You can always save the original on tape, even if it does not work. The idea is not to let the error get the best of you, and make those changes. You will learn something from them even if it is not the answer.

OSI-tems Index for Volume 1 and 2.

Mike Bassmen
39-65 52 st.
Woodside, NY 11377

During a great deal of our meetings, the same questions get asked over and over again. Usually, many of these answers have already been published somewhere in the first 13 issues of OSI-tems. So here for your convenience are the articles in these issues. They are arranged according to author.

William J. Balaban

1. I Upped My Memory - Up Yours. Volume 2, Number 7. 4pages.
total: 4 pages.

Michael Rolles

1. Unlocking OSI 'End-User' Program Disks. Volume 1, Number 1. 1 page.
total: 1 page.

George Brown

1. Adjustable Print Routine. Volume 2, Number 1. 1 page.
total: 1 page.

Thomas Cheng

1. Basic/Machine Language Variable Passing. Volume 2, Number 2. 1 page.
2. Turning USR(X) Routines into Data Statements. V2,N4. 1p.
3. Waiting on Basic. V2,N7. 2p.
4. Xerox Model OSI. V2,N8. 1p.
total: 5 pages.

Mike Cohen

1. Hardware Reviews. Volume 2, Number 2. 1 page.
2. Lissajous. V2,N2. 1p.
3. Poker Routine. V2,N3. 1p.
4. Number Formatter. V2,N6. 3/4 p.
5. I-Ching. V2,N6. 3/4 p.
6. Bar Graph. V2,N6. 1p.
7. Trig Quiz. V2,N6. 1 3/4 p.
8. 23 Androids. V2,N6. 2p.
9. Machine Language Call. V2,N7. 3/4 p.
10. Useful Magazine Article Index. V2,N7. 1 1/2 p.
11. Block Letters. V2,N8. 2 1/2 p.
12. Software Reviews. V2,N8. 1p.
13. Software Review. V2,N9. 3/4 p.
14. OS65D Corner. V2,N10. 1p.
total: 15 pages.

Index...continued.

Klaus Ernst

1. The CIP is Alive and Well...in Germany. Volume 2, Number 10. $2\frac{1}{2}$ pages.
2. Cassette Motor Control. V2,N11. $3\frac{1}{2}$ p.

total: 6 pages.

David Gillett

1. Two Software Reviews, or, The President Speaks! Volume 2, Number 6. 1 page.

Walter Godsoe

1. More on Copyright. Volume 2, Number 10. $\frac{1}{2}$ page.
2. Book Review: Programming and Interfacing the 6502. V2,N10. $\frac{1}{2}$ p.

total: 1 page.

Peter Halverson.

1. 600 Board to OSI Bus Schematic. Volume 2, Number 7. 2 pages.

total: 2 pages.

Wally Kendall.

1. 65U Memory Locations. Volume 2, Number 9. 1 page.

total: 1 page.

Warren Modell

1. Letter. Volume 2, Number 6. 2 pages.
2. Book Review: The First Book of Ohio Scientific. V2,N6. $1\frac{1}{2}$ p.
3. General Items. V2,N7. $3/4$ p.
4. General Interest. V2,N10. 1 p.
5. Program Reviews. V2,N10. $1\frac{1}{2}$ p.

total: $6\frac{1}{2}$ pages.

Milagros Montalvo

1. Peek Meter. Volume 2, Number 8. $1\frac{1}{2}$ pages.
2. Mystery Program. V2,N11. $\frac{1}{2}$ p.

total: $1\frac{3}{4}$ pages.

Ohio Scientific

1. Variable Storage. Volume 2, Number 2. 6 pages.
2. OS65U Patch. V2,N6. 1p.

total: 7 pages.

Ugo Re

1. Etch-A-Sketch. Volume 2, Number 6. 2 pages.
2. Serial Interface. V2,N6. 3p.
3. Single Drive Copy. V2,N7. $1\frac{1}{2}$ p.
4. Editorial. V2,N10. $\frac{1}{2}$ p.
5. Software Review: DO Secretary. V2,N10. $3/4$ p.
6. Book Review. V2,N10. $3/4$ p.

total: $11\frac{1}{2}$ pages.

Index...continued.

Peter Schreiber

1. Extending OSI Basic-In-Rom. Volume 2, Number 9. $3\frac{1}{2}$ pages.
 2. Depth Charge. V2,N10. $1\frac{1}{2}$ p.
 3. Cassette Save/Hex Dump. V2,N10. $2\frac{1}{2}$ p.
 4. Base Conversions. V2,N10. 1p.
 5. Extending OSI Basic-In-Rom, Version 2. V2,N11. $4\frac{1}{2}$ pages.
- total: 13 pages.

Danny Schwartz

1. Fine Points of OSI Basic. Volume 1, Number 1. $3\frac{1}{2}$ pages.
 2. CIP Memory Locations. V2,N1. $1\frac{1}{2}$ p.
 3. Drawing Program. V2,N1. $1\frac{1}{2}$ p.
 4. Invisible Programs. V2,N3. 1p.
 5. Hex Fifteen Puzzle. V2,N3. $1\frac{1}{2}$ p.
 6. Beat The String Bug. V2,N4. $2\frac{1}{2}$ p.
 7. Dumb Terminal. V2,N5. 1 p.
 8. July 4th Special. V2,N6. $\frac{1}{2}$ p.
 9. Updates and Corrections. V2,N6. $1\frac{1}{4}$ p.
 10. Goto X. V2,N6. 1p.
 11. A Quick & Dirty Way to Put Machine Language On Tape. V2,N8. $2\frac{3}{4}$ p.
 12. Sorting Arrays. V2,N8. $\frac{1}{2}$ p.
 13. Printing Normal Error Messages. V2,N10. $1\frac{1}{2}$ p.
 14. Toy Piano. V2,N10. 1p.
 15. 65U Utility Correction. V2,N11. 2p.
- total: 22 $\frac{3}{4}$ pages.

Terry Terrence

1. Find for OSI. Volume 2, Number 2. 1p.
 2. Resequencing. V2,N3. 3p.
 3. Editorial. V2,N4. $\frac{1}{2}$ p.
 4. Software Copyright. V2,N4. $2\frac{1}{2}$ p.
 5. Thoughts on Cassette Recording. V2,N6. 5p.
 6. Review: TI Model 733 'Silent Writer'. V2,N7. 3p.
 7. Editorial. V2,N9. $\frac{3}{4}$ p.
 8. Miscellany. V2,N9. $1\frac{1}{2}$ p.
 9. Inside 65D's Kernel. V2,N9. 1p.
- total: 18 pages.

Index...Continued.

Larry Thaler

1. Joysticks. Volume 2, Number 3. $\frac{1}{2}$ page.
 2. Super Superboard. V^o,N3. $1\frac{1}{2}$ p.
 3. Editor's Letter. V2,N5. 1p.
 4. Add an RS-232 Port to Your C1P. V2,N5. $4\frac{1}{2}$ p.
 5. Submarine. V2,N5. 2p.
 6. Stupid Trek. V2,N5. 2p.
- total: $11\frac{1}{2}$ pages.

Salomon Lederman

1. Bits, Volume 2, Number 1. $\frac{1}{2}$ page.
 2. Fire Game. V2,N1. 1p.
 3. Machine Language Clock. V2,N2. $1\frac{1}{2}$ p.
 4. Probability Machine. V2,N2. 1p.
 5. Multiplication. V2,N2. $\frac{1}{2}$ p.
 6. Kaleidoscope. V2,N2. $\frac{1}{2}$ p.
 7. Factorial. V2,N3. $\frac{1}{2}$ p.
 8. Press Release. V2,N3. $1\frac{1}{2}$ p.
 9. C1P Trivia Quiz. V2,N3. 2p.
 10. Life. V2,N4. 1p.
 11. Dodge 'em. V2,N5. $1\frac{1}{2}$ p.
 12. Type Test. V2,N5. $1\frac{1}{2}$ p.
 13. Gopher. V2,N5. $1\frac{1}{2}$ p.
 14. Print At. V2,N6. 1p.
 15. Memory Swapping. V2,N6. 1p.
 16. Random Numbers in Machine Language. V2,N6. $3/4$ p.
 17. USSR Dispatch. V2,N6. $1\frac{1}{2}$ p.
 18. Software Reviews. V2,N6. 2p.
 19. Trap. V2,N8. $1\frac{1}{2}$ p.
 20. Another Print At. V2,N9. 1p.
 21. UnNew Your Programs. V2,N10. $3/4$ p.
 22. Shape Plotter. V2,N10. 2p.
 23. 6502 Machine Language Class 1. V2,N10. 3p.
 24. Renumberer. V2,N11. $1\frac{3}{4}$ p.
 25. Software Review. V2,N11. $3/4$ p.
 26. 6502 Machine Language Class 2. V2,N11. 3p.
- total: $32\frac{3}{4}$ pages.

Index...Continued.

Mike Bassman

1. Great Moments in OSI History. Volume 1, Number 2. 2 $\frac{1}{2}$ pages.
2. Skeet Shoot. Volume 1, Number 2. 1 $\frac{1}{2}$ p.
3. Software Reviews. V2,N1. 1 $\frac{1}{2}$ p.
4. "Mike". V2,N1. $\frac{1}{2}$ p.
5. Racer. V2,N1. $\frac{1}{2}$ p.
6. Comprint 912-S Printer Review. V2,N2. $\frac{1}{2}$ p.
7. Extended Cassette Manipulation. V2,N2. 2 $\frac{1}{2}$ p.
8. Hangman. V2,N2. 1 $\frac{1}{2}$ p.
9. Software Sources. V2,N2. $\frac{1}{2}$ p.
10. Program Security. V2,N3. 1p.
11. OSI. V2,N3. 3 $\frac{1}{4}$ p.
12. Reflex Test. V2,N3. 1 $\frac{1}{2}$ p.
13. Source Update. V2,N3. 1p.
14. Working Around Strings. V2,N3. 2 $\frac{1}{4}$ p.
15. Elevator Demo. V2,N3. 2p.
16. Potato Chip Invasion. V2,N4. 2p.
17. Vectors and the Challenger 1P. V2,N5. 2p.
18. Source Update. V2,N5. $\frac{1}{2}$ p.
19. New Instructions for the 6502. $\frac{1}{2}$ p.
20. Quiz System. V2,N6. 1 $\frac{3}{4}$ p.
21. Understanding Basic - The Routine at 00BC. V2,N7. 1 $\frac{3}{4}$ p.
22. Source Update. V2,N7. 1 $\frac{1}{2}$ p.
23. News From OSI. V2,N8. 1 $\frac{1}{2}$ p.
24. Software Reviews. V2,N8. 2p.
25. Source Update. V2,N9. 1p.
26. CIP Series 2 Review. V2,N10. 1 $\frac{1}{4}$ p.
27. Source Update. V2,N11. 2p.
- total: 40 $\frac{1}{4}$ pages.

Because there has been a bit of confusion over which OSI-tems issue was when, because of mislabeling issues and some month jumping, here is the correct list.

list on next page...

Index...continued.

- Volume 1, Number 1. November, 1979. Danny Schwartz, Editor.
- Volume 1, Number 2. December, 1979. Mike Bassman, Editor.
- Volume 2, Number 1. January/February, 1980. Thomas Cheng, Editor.
- Volume 2, Number 2. March, 1980. Mike Bassman, Editor.
- Volume 2, Number 3. April, 1980. Salomon Lederman, Editor.
- Volume 2, Number 4. May, 1980. Terry Terrence, Editor.
- Volume 2, Number 5. June, 1980. Larry Thaler, Editor.
- Volume 2, Number 6. July, 1980. Mike Bassman, Editor.
- Volume 2, Number 7. August, 1980. Thomas Cheng, Editor.
- Volume 2, Number 8. September, 1980. Mike Cohen, Editor.
- Volume 2, Number 9. October, 1980. Terry Terrence, Editor.
- Volume 2, Number 10. November, 1980. Ugo Re, Editor.
- Volume 2, Number 11. December, 1980. Danny Schwartz, Editor.

January 8, 1981

Once you are able to write a simple routine in machine language, then what? Well, you can try to write the whole rest of the program in machine or assembly language. Unfortunately it is not a simple task to write a large machine language program. Also many machine language programs could have been satisfactorily written in BASIC. So what do you do? Well, the obvious answer is to write parts of your program in BASIC, and other parts in machine language. The parts that don't need to run fast or very efficiently can be left in BASIC. Once you find out which sections run slow then you can begin to something about it.

There are many programs that could be greatly improved without great effort. A screen clear will run instantaneously in machine and it can be written easily in less than fifty bytes. Isn't it worth a little effort to get that screen clear to run fast? If you have a C2 or a C4 then you could also use a color setting routine. If you have games where scores have to be constantly updated then BASIC may be too slow. Or if you want to put messages up on the screen, then POKING letters on may be slow. There are so many examples of simple routines that can speed up your program greatly if done in machine language. If you look back at old issues of OSI-TEMS then you'll see many of the routines that I've published. Try to see how they work. Disassemble them. Most are not too difficult.

OK. Let's say that you've found a part of your BASIC program that is slow. Let's say that you're writing some fighting game and you need real time missiles, and that you'd like to allow each player to have up to four missiles in the air at the same time. The idea is good but the program is slow. Then one day you look into your favorite computer magazine and you find that somebody has written a routine that handles the screen plotting and movement of realtime missiles. (I have no idea why anybody would ever write such an article but let's pretend.) You envision a super game, but how are you going to interface that program to your main BASIC program?

The first thing that you have to do is figure out what information is needed by the routine. You'll also want to know what the program will output. You can work with the missile routine as a black box; there is no need to know how the thing works, you just want to know what it does. So you read the documentation that comes with the program

and you figure out what information you have to provide. Suppose that the routine is set up for a one player game. Also suppose that the routine is totally selfsufficient; it needs no BASIC help to run. Then all you'll have to do is call the routine every time that you want to fire or move a missile. BASIC-poke statements can be used to set flags for the program, and the USR function is used to call the routine.

The USR function is not difficult at all to use. First you determine the starting address of the routine. Make sure that the routine does not interfere with your BASIC program in any way. Always save BASIC programs before you add machine language routines. Once you have determined the starting address you must convert it to hex if it is not already in hex. From hex you break the number up into two bytes. The low byte gets poked into 11, the hi byte goes into 12. In BASIC of course you must poke decimal numbers into 11 and 12.

Here's a simple example: There is a ROM routine at FD00. If you want to call it then you break up FD00 into FD and 00. FD is 253 in decimal and 0 is just 0. So you poke 11 with 0 and 12 with 253. Make sure that your routine ends with a RTS opcode (its code is 60). The RTS is necessary to return control to BASIC. With this simple knowledge you can treat machine language routines just like BASIC sub-routines.

At this point you must find some way to save your machine language routine with your BASIC program. This problem has been gone over various times in OS-ITEMS; I will not repeat it here. Once all this is done you should be able to use machine language routines with ease. I will provide one example to illustrate these techniques.

A machine language screen clear is very useful to have. Let's say that you have the following program in memory:

1500	A9 00	151C	D0 EC
1502	8D 0D 15	151E	60
1505	A9 D0		
1507	8D 0E 15		
150A	A9 20		
150C	8D 00 00		
150F	EE 0D 15		
1512	D0 03		
1514	EE 0E 15		
1517	AD 0E 15		
151A	C9 D8		