OSITEMS

VOLUME 2 NO. 9 NOVEMBER '80

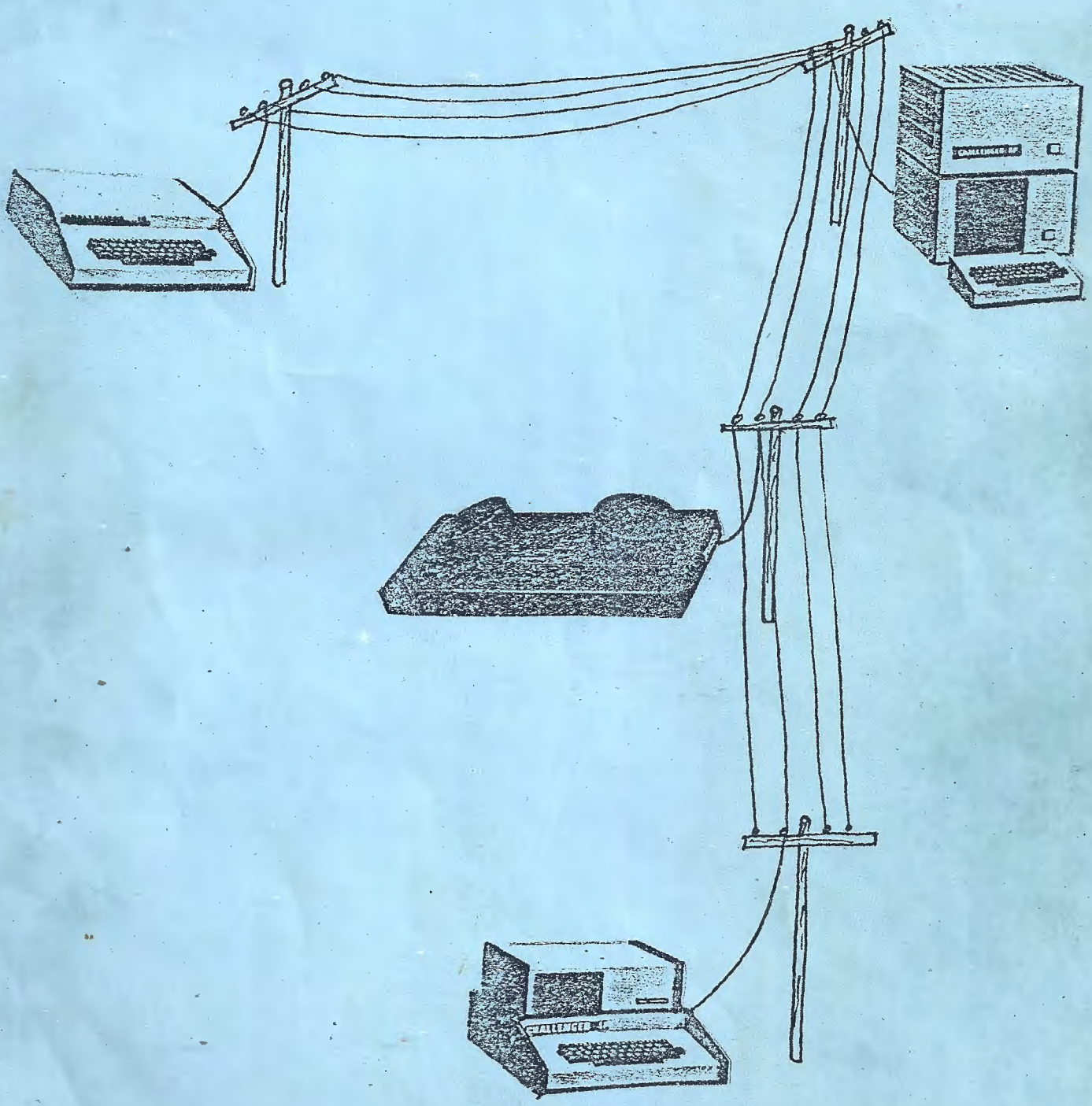


TABLE OF CONTENTS

OSI USER'S GROUP

Meets the first Thursday of every month at:

Polk's Hobbies
314 Fifth Ave.
New York, NY 10001

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

Dues: Standard Membership, $10.; Student Membership, $5.

OSItems Staff

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

This month's editor: Ugo V. Re'

FROM THE FRONT DESK
by Ugo V. Re'

.....

Editorial

.....

With this issue we start our second year of publication of our own journal, OSItems. This is indeed "OUR" journal as most of the members contribute something to the production of each and every issue.

It is only through each members continuing efforts that "OSItems" will provide the information that each of us needs to increase our knowledge of the equipment, software and workings of various OSI products.

Have you written an article? Why not? All articles, large or small will provide some information or a new view point for some other member of the group.

Do you have a hardware or software problem? Write a letter requesting help. Did you have a problem and were you able to fix it? Write about your problem and the solution, someone may be having the same problem.

It is only through the sharing of information that each of us will grow.

Get those articles in ! Here are the editors for the next issues of OSItems. Send the articles to them or leave them at Polk's.

December:
Dan Schwartz
75 E. 190 st.
Bronx, N.Y. 10468

January:
Salomon Lederman
40 Waterside Plz.
New York, N.Y. 10010

February:
Larry Thaler
152-30 13th Ave.
Whitestone, N.Y. 11357

Front Cover

As the personnel computer field grows the need for equipment grows and the cost of the equipment decreases.

For each of us to grow we need to acquire knowledge. As we grow, our systems grow and our methods of acquiring and exchanging information must change to keep pace with our increasing need to acquire information.

One method of exchanging information between computers is via the telephone network and the use of a modem. A modem (modulator/demodulator) converts a digital signal from the computer into an analog signal that is transmitted over the standard telephone line and it takes the incoming analog signal and converts it into a digital signal for the computer.

The modem and telephone line forms a link that ties computers together so that they can exchange information. The cost of a modem is now within our budgets.

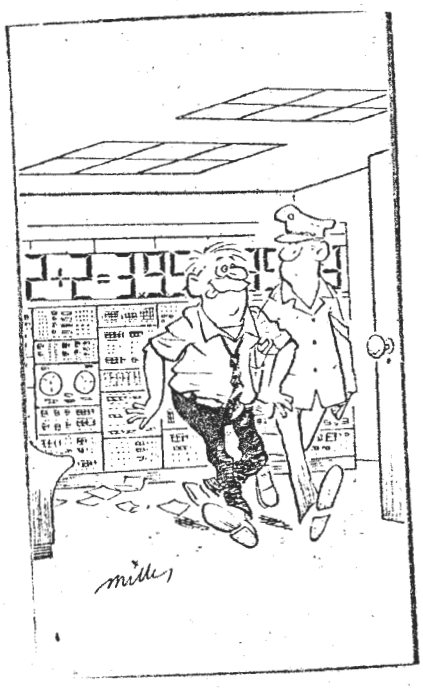
At the December meeting Mr. Michael Doliton will speak on building a modem for $50. This should be an informative meeting for anyone who is planning on using their computer to access other data bases.

Who Are We?

We are a group of people with a common interest and goal. We have gathered to exchange information and acquire knowledge about OSI systems and other related products.

But who are we? Do we have a name?

In the EDP world, groups and systems are conveniently referred to by an acronym - a form of verbal shorthand. What we need is an acronym, a name, for our group. Do you have an acronym that reflects the purpose of our group? Send all acronyms (names) for our group to me, Ugo, prior to our next meeting in December. At the meeting we will pick one acronym as a name for our group.



A Glossary of Computer Acronyms

The following acronyms and abbreviations can be pronounced as words (indicated by an asterisk) or as strings of letters. A phonetic guide appears in parentheses if the pronunciation is especially odd.

ADAPSO* (Association of DAta Processing Service Organizations) A trade association for vendors of computer systems, software and services.

AFIPS* (American Federation of Information Processing Societies) Sponsor of the National Computer Conference (NCC).

AI (Artificial Intelligence) Roughly speaking, computers doing tasks that, if humans were to do them, we would say require intelligence.

ALGOL* (ALGOrithmic Language) Developed in 1958, a landmark programming language used for solving mathematical problems.

ASCII* (American Standard Code for Information Interchange) A code used in computers and communications systems in which each character, number or special character is defined in 8 bits. (ask-ee)

BASIC* (Beginners' All-purpose Symbolic Instruction Code) A programming language with simple syntax and few commands, often used on microcomputers.

BISYNC* (BInary SYNChronous) An IBM-developed communications protocol which, during the seventies, approached becoming an industry standard.

Bit* (BInary digiT) The smallest unit of information the computer recognizes, represented by the absence or presence of an electrical impulse.

CAD/CAM* (Computer-Aided Design/Computer-Aided Manufacturing) A term applied to efforts to automate design operations and manufacturing operations.

CAI (Computer-Assisted Instruction) Using computers to help teach students. Usually the computer "talks back" to the student, telling him or her when a mistake is made.

CICS* (Customer Information Control System) A widely used IBM teleprocessing monitor. (kicks)

COBOL* (COmmon Business-Oriented Language) A high-level programming language used in business applications.

CPU (Central Processing Unit) The part of the computer that controls the interpretation and execution of the processing instructions.

CRT (Cathode-Ray Tube) A televisionlike screen that can be used for entering data into or retrieving it from the computer.

DASD* (Direct-Access Storage Device) A unit of computer equipment that allows direct, quick access to storage for entry or retrieval of information. (daz-dee)

DBMS (Data-Base Management System) The software that makes a company's data base—integrated, computerized files of information—accessible in multiple ways to many users.

DOS (Disk Operating System) An operating system that uses disks to assemble, edit and execute programs.

DP (Data Processing) The transformation of raw data into useful information by electronic equipment; sometimes referred to as ADP (automatic data processing) or EDP (electronic data processing).

ENIAC* (Electronic Numerical Integrator And Calculator) Generally recognized as the first stored-program computer ever built. Completed by John Mauchly and J. Presper Eckert at the Moore School of the University of Pennsylvania in 1946.

EPO (Emergency Power Off) The circuit, and the buttons activating it, that can turn an entire computer off in an emergency. There may be as many as 20 EPO buttons in a large installation.

Acronyms

FIFO* (First In, First Out) An accounting concept, also used in programs where the oldest data is used first. (fie-foe)

FORTRAN* (*FOR*mula *TRAN*slator) A programming language widely used to solve scientific and engineering problems.

GIGO* (Garbage In, Garbage Out) Or what happens when you put sloppy data into a computer; you get sloppy answers. (gig-oh)

IC (Integrated Circuit) An electronic circuit or combination of circuits contained on semiconductor material; the basis of a computer's intelligence.

IMS (Information Management System) A very large IBM-supported data-base management system.

IPL (Initial Program Load) The procedure that causes an operating system to begin.

ISAM* (Indexed Sequential Access Method) A method of gaining access to data by referencing a key word. (I-Sam)

JCL (Job Control Language) A command language for issuing requests for action to the operating system.

LED (Light-Emitting Diode) A solid-state electrical valve that produces light; used in low-cost, low-powered digital displays.

LIFO* (Last In, First Out) An accounting technique in which the last data entered is the first data operated on. (lie-foe)

LSI (Large-Scale Integration) The process of integrating a large number of circuits on a single chip of semiconductor material.

MCP (Master Control Program) The name for all operating systems produced by Burroughs.

MICR (Magnetic Link Character Recognition) The technique used in banking to encode account numbers and check values so they can be automatically debited from accounts. The characters are only visible for convenience; the real encoding is magnetic and invisible.

MIS (Management Information System) A system designed to provide information to management as an aid to decision making.

Modem* (MOdulator-DEModulator) An electronic device that can be used to connnect computers and terminals over communications lines.

OCR (Optical Character Recognition) The machine reading of graphic characters—both letters and numbers—using light-sensitive devices.

OEM (Original Equipment Manufacturer) An organization that purchases computer equipment from the manufacturer and resells it to users, after adding value in the form of software or by integrating whole systems.

POS (Point-Of-Sale) Systems used to capture data at the point of sale or transaction in retail operations.

PROM* (Programmable Read-Only Memory) A memory that is not programmed during manufacturing and requires a physical or electrical process to program it.

RAM* (Random Access Memory) A memory in which each element of information has an address (location) and from which any element can be easily and conveniently retrieved by using that address.

ROM* (Read-Only Memory) A memory whose data cannot be changed by programming.

SDLC (Synchronous Data Link Control) The current IBM-supported communications protocol, designed to replace BISYNC.

UNIVAC* (*UNIV*ersal Automatic Computer) Built by John Mauchly and J. Presper Eckert, the first commercially available general-purpose computer. The UNIVAC I was accepted for use by the Bureau of the Census on March 30, 1951. Today's Sperry Univac computer operation is a direct descendant of the original UNIVAC project.

++

OS-65D CORNER - WHAT TO DO IF YOU HIT BREAK
by Mike Cohen,
with contributions by Hal and Fred

In all OSI video systems except the C1 MF, hitting BREAK can be a real disaster, since it wipes out the disk drivers. Here is a way you can usually recover.

First, re-enter the kernal at $2A51 (in the monitor, type .2A51G). You will get the usual A* prompter, then type "GO FF00". You will get an ERR#7, but the disk drivers will be "alive" so you can save your program, but DO NOT try to return to BASIC or the assembler, since it's pointers have been screwed up.

What happens when you hit BREAK is that the ROM tries to initialize the ports at $C000, but that makes it unusable by OS65D. The routine at $FF00 re-initializes the port, homes the drive and prepares to reboot. After this, OS65D will beable to access the drives again.

ON the C1P MF, you have a completely different problem. The ROM doesn't initialize the ports, but in an attempt to reset ROM BASIC's I/O pointers, it wipes out a portion of disk BASIC or assembler in page 2. The fix in this case is to simply restore that page. If you were using BASIC, enter the kernel and type "CA 0200=02,1" and then re-enter BASIC. You will usually preserve the program in this case. With assembler, there is no safe way to preserve the program, so the best way is to save the program and reboot.

A QUICK WAY TO unNEW YOUR PROGRAMs
by Salomon Lederman

Have you ever accidentally NEWed a program and then wished you hadn't? Here's a quick way to fix some of the damage.

If you have begun to type in another program then this method will not work for you. But let's say that you just typed NEW. You try a List and there's nothing there. Luckily for you BASIC hasen't erast your program. It has just moved around a few pointers.

Here's what you do.

BREAK and go into the monitor. Look at $007C. It should be an $03. Change it to some number that is just a bit less than the number of pages of memory that you have. For example, I have 8K which is 32 pages so I'll change $007C to $30, which is

$1E. Now go on to look at $0302. It should be a $00. Change it to an $03.

Now type a .A31C G. This runs a routine that relinks your line number pointers.

Now you are back in BASIC so just LIST and your program is back. Do not attempt to RUN it. Using any variables may very well cause your program to explode. Just SAVE it and thank god that you pulled through this crisis.

Now that your program is safely stored on tape run it and you may be in for a big surprise.

Thanks to Mike Cohen for this useful routine information.

.....

MORE ON COPYRIGHT
by Walter Godsoe
233 E. 3st.
N.Y., N.Y.

.....

In a previous issue of OSItems Terry Terrance discussed the need to copyright material.

I would like to suggest the following method to have proof of a date of creation which will hold up in court. This method is quick, offical, and cheap.

Simply place a program listing in an envelope and mail it to yourself. If you want you could send it by registered mail but this is not called for. Upon receiving the envelope DO NOT open it. The postmark is the date of creation and it is a goverment seal. Put the envelope in a safe place, and should you ever have to go into court, only allow a court officer to open the envelope.

So for the cost of a postage stamp your protected. This method is frequently used by authors and it does meet all legal requirements.

BUILD YOUR OWN ROBOT FROM AN OFF-THE-SHELF KIT?

You'll have the opportunity to build a Heathkit robot as early as 1982. Designed around a Motorola 6802 μP and featuring voice synthesis, 4k of RAM, 32k of ROM and a detachable joy stick, the approximately 3-ft-high electromechanical robot was demonstrated to Heath retail-store managers in August. The less-than-$1000 robot employs one multipurpose arm powered by six stepper motors.

The Benton Harbor, MI firm explains that the actual introduction date hinges on how fast the difficult software-development task progresses.—CW

GENERAL INTEREST (I Hope!!!)

Warren Modell
3133 Rochambeau Avenue
Bx. N. Y. 10467 (212) 654-3773

1. IBM is working on a computer that has to work in a Cryogenic environment (Close to absolute zero). It's presently called the Josephson-Junction or Josephson-Technology Computer. It can perform between one billion to 10 billion operations per second (30 to 50 times faster than our present technology). A prototype will take about 5 years to build and close to another five years before the first operational model will be available.

2. Compatible Diskette Cross-Reference guide (wall chart)

   Available free from:

   Carl L. Holder, V. P. Product Mkg. Div.
   Wabash Tape Corporation
   2700 Des Plaines Avenue
   DesPlaines, Il. 60018

3. If you are interested in languages other than basic - here are a few addresses of "User" groups.

   a) APL Market Newsletter
      2348 Whitney Avenue
      Mt. Carmel, Conn. 06518
      $6.00/yr. - includes a column on personal computer applications.

   b) Pascal Users Group (PUG)
      % Rick Shaw
      Digital Equipment Corp.
      5795 Peachtree Dunwoody Road
      Atlanta, Ga. 30342

4. a) Is there anyone interested in buying computer furniture?

   b) Would you be interested in customized furniture for your OSI System? or another System?

   Let me know - either at a meeting or phone me.

****

Hardware Review
by Mike Bassman
39-65 52 st.
Woodside, N.Y. 11377

.....

C1P Series 2
Ohio Scientific

The new C1P Series 2 has just arrived. A very large number of people have asked about the differences between it and the old C1, so I'll relay some of my impressions.

First and most obviously, there is the new case. It is a white and shiny plastic on metel. It strongly resembles the material of Imperial trooper armor in "Star Wars". The old C1P was drab, but the Series 2 is garish. I'm really not sure which I prefer.

Next is the much advertised new video display. When you turn on the Series 2, it comes up with the good old 24X24 display. Next, you must load in a small machine program (coded in BASIC data statements) which activates the 48X12 mode. This program, incidentally, fits on the top of memory, rather than in the free portion of page 2. This is both good and bad. The good part is that it frees page 2 for all those neat additions to BASIC that have been published. The bad part of it is that you don't have all of your memory left, so those programs which require 7.999K won't work any more.

When you hit BREAK, the new display automatically goes back to the old one. To get back the new one you have to do two POKEs, which, of course were not in the manual (I deduced them from the display initialization program).

Now about the display itself. There are 48 characters across by 12 down. Note that this display needs no extra video RAM, since 48X12= 24X24= 576. The characters are small, crisp and clear. Another feature is that it double spaces text, making it very good in the text readability section. This is awful for graphics as nothing contiguous can be drawn vertically, but this display mode is not meant for graphics. For graphics, select the 24X24 mode with one POKE. Since it does retain the 24X24 mode, all existing C1P programs will work on it.

The new display mode does several other things as well. It gives a new cursor, a block (CHR$(161)) rather then the underline. It will now print "SN" errors, rather than "S<bip>" errors. Unfortunately, this disables the printing of graphic characters since bit 7 is now masked off. (Note: for more details on this see D. Schwartz's article in this issue.)

One important difference is the addition of a RS232C serial port. On the old C1P, traces were provided for the port, but no parts. On the Series 2, the RS232 port is populated. Two RS232 connectors are mounted on the back of the case. One is for the printer and the other is for a modem. The reason for two separate ports is that the RS232 requirements for printer and modem are not the same. Specifically, lines 2

& 3 are reversed on one port (Printer). Note that only one port can be used, a switch is installed for this purpose.

Another matter of interest is that the Series 2 will be compatible with the soon to be released 630 board. The 630 board will have a complex DAC on board for music, as well as a new and improved RGB color. It will also have a high speed serial port.

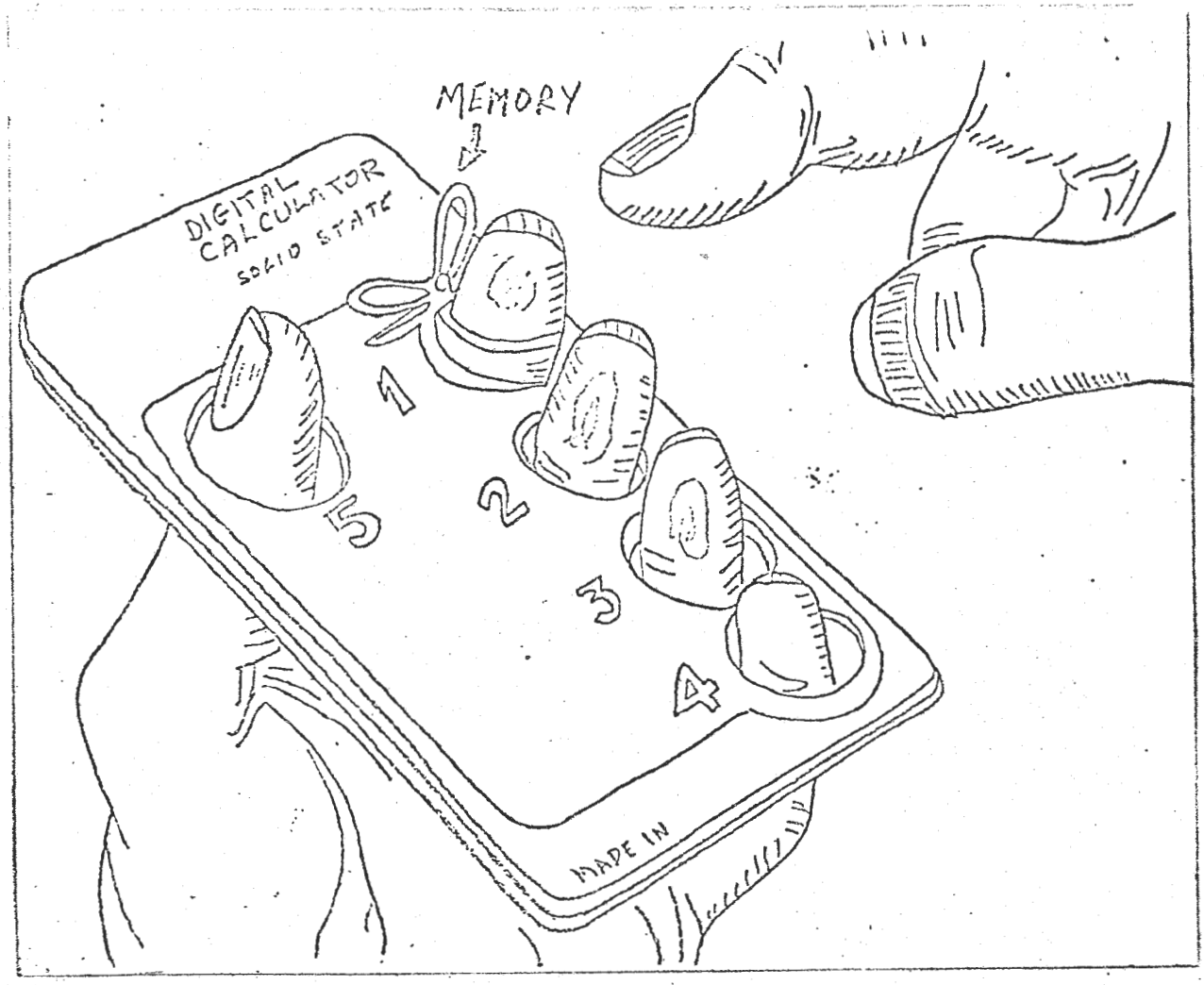
The Series 2 also comes with a ** NEW ** manual in a binder. Is it better than the old one? Probably, but this has to be proven.

Lastly, but not least is a new price. The C1P Series 2 is $479 as apposed to the old C1P which was $399 (8K).

Hand Calculator
by Anonymous

.....

This is the latest in hand calculators, it doesn't require batteries.



PROGRAM REVIEWS

by: Warren Modell
3133 Rochambeau Avenue
Bronx, N. Y.10467 (212) 654-3773

1. WP6502 Word Processing by Dwo Quong Fok Lok Sow
23 East 20th St.
N. Y., N. Y. 10003

The packaging is really impressive. The documentation is not only thorough but easily understandable and with a sense of humor. The program follows the packaging and documentation - it is great!! It is worth much more than what it is selling for.

2. Adventure: "Marooned in Space" by Orion Softwave Assoc., Inc.
Ossining, N. Y.

This is the best packaging I've seen in OSI games. Orion has the best quality plastic Zip Loc bag to the best quality cassette tape. The documentation is excellent. Everything is explained and clearly printed. I only have one other Adventure game, "Escape from Mars" which I thought was very good - This one is better!!

3. "Video Trek" Rev. 15 by Robert J. Retelle
2005 Whittaker Rd.
Ypsilanti, Mi. 48197

The packaging is standard. The action is absolutely terrific!! The graphics are one of the best for this type game (Star Trek). You have the Enterprise, Klingon Cruisers, Planets & Stars, Black Holes, a couple of Doomsday Machines, Phasers and Photons. I'm definitely buying more games from this guy.

4. Starship I #004C04-GG by Procom Software
8 Hampton South
Southampton, Mass. 01073

The packaging is standard. As a "Starship" goes across the screen you fire at a wall below, which is protecting a city. Eventually you shoot through the wall and if you have enough fire power left you may beable to destroy the city below.

5. Road Race w/Track I #002C08 - GG Procom Software
Track 2 For 8K Road Race #003C08-GG

The packaging is standard (Two Cassettes) This is a good game! There are two cars, you can compete against the computer or a Human Type!! There's a nice track and when you bump against

the wall or the other car, you lose control and lose time. The game offers you a choice of speeds and the number of times you wish to go around the track.

I think the number of programs are increasing quite a bit for OSI Systems. I believe they're getting more sophisticated. I hope this trend continues and I think many members of our club are playing a big part in this rapid development.

SOFTWARE REVIEW
by Ugo V. Re'

DQ Secretary
(an enhanced OS65D)
Dwo Quong Fok Lok Sow

"DQ SECRETARY allows you to check the directory, create, delete, and rename files, without disturbing data being held in memory." "When you're ready to save data, DQ SECRETARY creates the file and puts the data in it in one swift operation and makes the file the right length and finds a space for it on disk, automatically."

"If you want to store data in an existing file which isn't big enough, DQ SECRETARY automatically extends that file (with your permission)." "If there isn't room to extend it, DQ SECRETARY automatically shifts the file somewhere else on disk." "If the disk is almost full it repacks it for you, to group spare tracks together and at the same time, it updates the disk directory for you."

"DQ SECERTARY is written in machine language, so its fast." "It can be incorporated into any standard OS-65D systems disk or into WP6502." "It loads into your workspace before you start writing text or programs, just type EXIT (or select "FILE" from WP6502's Menu) and the Secretary's services are immediately available when you need them."

The above Quotes comes from the advertisement for the DQ SECRETARY. I used the Secretary along with WP6502 to file many of the articles that I had to type for this issue of OSItems and I can assure you that the Secretary does everything the ad says it does.

The DQ SECRETARY replaces OSI's CREATE, DELETE, RENAME and DIR utilities and also takes over the PUT and LOAD functions. The program comes with a well written manual that throughly explains how to use the various Menue options and how to incorporate the program on an OS-65D or Wp6502 disk.

I would recommend this program along with WP6502 for anyone who will be doing word processing.

BOOK REVIEW
by Walter E. Godsoe

Title: Programming and Interfacing the 6502
Author: Marvin L. De Jong
Publisher: Howard Sams & Co. Inc.

Mr. DeJong's book is one of many books which deals with machine and assembly language.

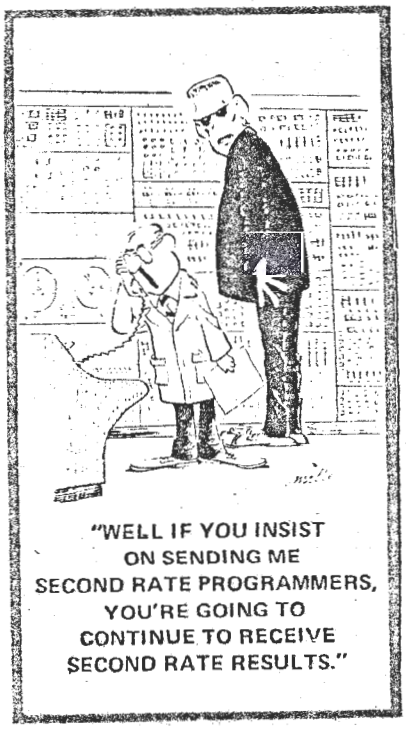
Those of us who are new to computing and its complexities can easily be overwelmed by the mass of information which is available by trying to understand and assimilate this information without hands on experence.

Mr. DeJoug recognizes the value of getting ones hands dirty by doing some work. Unfortunately for OSI users he is primarily speaking to SYM, KIM, and AIM owners. THE information and experements in general can be implemented on any 6502, but he is specific about addresses and devices not found on the C1P. This need not prevent us from taking part in the experements. Rather than toggling a 6522, a screen location could simulate the device. For all practical purposes a device is an address, so the experence need not escape us.

The first half of the book is dedicated to programming. In each of the ten chapters, methods of addressing and instructions are dealt with by giving examples, explanation and most important experements. His intergration of concept and application make this book a real tool.

The second half of the book defines interface problems, decoding, buffering, etc. They are discussed individually and as a whole system. Experements are emphasized which cover the hardware that most of us could have.

I strongly recommend this book to anyone interested in 6502 machine language and interfacing. This book is recently published and should be widely available.



BOOK REVIEW
by Ugo V. Re'

.....

6502 Assembly Language Programming
Lance A. Leventhal
Osborne/Mc Graw-Hill

This book contains a broad coverage of 6502 assembly language programming techniques along with system software development. It assumes that the reader is familiar with the general features of computers, microprocessors, addressing methods, and instruction sets.

The first two chapters provide a brief but through introduction to assembly language programming, numbering, instruction mnemonics, and assemblers. Chapter three covers the 6502 addressing modes and the instruction set.

The next six chapters present various examples of simple microprocessor tasks. Each program example contains a title that describes the general problem and a statement of purpose that describes the specific task. The example contains data, a flowchart, and a source and object listing. Explanatory notes completes the example. At the end of each chapter there are problems that the reader must solve. The problems follow the material and examples presented in the chapter. All of the examples and problems represent individual tasks that any microprocessor can do.

In chapter 10, Subroutines, the author discusses how one or more of the simple tasks can be used as part of a larger program.

The next two chapters cover the programming need to communicate with and respond to the outside world. The author discusses the 6520 (PIA) and 6850 (ACIA) chips and how to control them via software. More examples and problems are given on solving real problems.

After reviewing the examples and programming techniques a person should be familiar with the 6502 assembly language and the coding of short programs. The remaining chapters cover program design, flowcharting, structured programming, debugging, testing, documentation, maintenance and redesign of programs.

For the 6502 user this book packs a great amount of programming information and useful examples and problems. I would recommend this book to all users interested in learning assembly language.

Note: this is the book that Salomon Lederman will be using for the programming classes at the club.

PRINTING NORMAL ERROR MESSAGES

by Dan Schwartz

.....

One of the peculiarities of OSI ROM BASIC computers is the rather odd error messages they give. A syntax error is "S<corner of a square> ERROR", a function call error is "F<big slash> ERROR" etc. The reason for this, of course, is that the second character of the error message is really a letter with bit 7 set in their ASCII codes, and are interpreted as a graphic character by the OSI character generator. When Microsoft first wrote the BASIC interpreter they didn't know that OSI was going to implement this graphic system, so they didn't think that the state of bit 7 in the output character mattered. It was convenient for other reasons to have bit 7 set on the second character, so they just let them output that way.

OSI's "solution" to the graphic-character-instead-of-letter problem on disk systems is simply to mask off bit 7 on every character output to the screen. That provides normal error messages, but it also makes it impossible to print graphic characters to the screen with a PRINT CHR$(X) statement.

Is there a way to get normal error messages without sacrificing the ability to PRINT graphics to the screen? As someone else once said, would I be writing this if there weren't?

What needs to be done is to find some way of distinguishing the error message characters from characters output by a normal PRINT statement. The key to doing this is the 6502's X register.

Whenever a string is output by BASIC (a "string" here includes the ASCII representation of a number, as well as an actual BASIC "string"), the X register is used as a counter of how many characters are left to print. For example, in printing the string "THIS IS A STRING", the X register will have a value of 16 (decimal) when the initial "T" is printed, since there are 16 characters in the string. After outputting the "T", the X register is decremented and, since it is still non-zero, BASIC knows to go ahead and print the next character "H". This process will continue until the final character, the "G", is printed, at which time the X register will have a value of one. When the X register is decremented again and reaches zero, BASIC knows that there are no more characters to output in this string.

How does this help us in regard to printing error messages? Well, the thing to notice here is that since the value of X is continually decremented while a string is being printed, X will never have the same value for two successive characters, except in the case when a single character string is being printed following another string, in which case X will twice in a row have the value of one.

When error messages are printed, though, X will twice-in-a-row have the same even value, namely the value of the offset into the table of error messages of the error message being printed. For example, for an "NF" (NEXT without FOR), X has a value of zero, while for a "SN" (Syntax) error, X has a value of two. Since these offsets are always even numbers, X will never equal one when printing an error message. The second character of an error message can therefore be recognized as any character for which the X register has the same value as it did for the previous character, and this value is anything other than a one.

Using this fact, we can write the following output routine for the C1P. The

routine makes use of memory location $0204, which is not used by the normal ROM operating system, to store the previous value of the X register.

```
0222  EC 04 02    CPX $0204    ;is X same as last time?
0225  D0 06       BNE $020D    ;no, go ahead and print
0227  E0 01       CPX #$01      ;yes, is the value one?
0229 F0 02       BEQ $020d    ;yes, go ahead and print
022B  29 7F       AND #$7F      ;no, error mask bit 7
022d  8E 04 02    STX $0204    ;save X for next time
0230  4C 69 FF    JMP $FF69    ;do normal output
```

While the routine is shown assembled at $0222, it can actually be located anywhere in memory without change.

Here is a short BASIC program which POKES the above routine in place, pointe the output vector at it, and then exercises it by printing some graphics characters and then caues a syntax error.

```
10 REM POKE CODE IN PLACE
20 FOR X=546 TO 562: READ I: POKE X,I: NEXT
30 REM POINT OUTPUT VECTOR AT IT
40 POKE 538,34: POKE 539,2
50 REM PRINT SOME GRAPHICS
60 PRINT CHR$(128); CHR$(129) + CHR$(130); CHR$(131)
3T2570 THIS IS A SYNTAX ERROR
80 DATA 236,4,2,208,6,224,1,240,2
90 DATA 41,127,142,4,2,76,105,255
```

DEPTH CHARGE

by Peter Schreiber

.....

This program will run on a C2, C4, and C8 system. It will not run on a C1 without modifications.

When you bring up BASIC, answer "3000" to "MEMORY SIZE?" (on a 4K system), or POKE 134,15.

The machine program below must be entered befor entering the BASIC program. (See Cassette Save article to save the machine program on tape and the system manual to load the machine program from tape.)

```
0F00   A2 00 86 FE A9 D0 85 FF A0 00 A9 20 91 FE C8 D0
0F10   FB E6 FF E8 E0 08 D0 F4 A9 00 8D 44 DE 85 E0 A0
0F30   F0 0C 99 00 D1 B9 CA 0F 99 81 D1 C8 D0 EF A9 01
0F40   8D 02 12 A9 80 8D 00 DF A9 0E 85 F7 A9 D4 85 F8
0F50   A0 00 A9 B5 91 F7 C8 A9 B6 91 F7 A9 63 85 0B A9
```

```
0F60   0F 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 A0 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 74 A2

0FC0   53 43 4F 52 45 3A 20 20 20 00 54 49 4D 45 3A 20

0FD0   31 32 30
```

After the machine program is in, load the following BASIC program.

```
10 REM       KEYBOARD IS USED AS FOLLOWS::
20 REM
30 REM          (1)  MOVES SHIP TO THE LEFT
40 REM          (2)  MOVES SHIP TO THE RIGHT
50 REM          (3)  DROP DEPTH CHARGE
60 REM
70 REM       YOU HAVE 2-MINUTES TO CLEAR OUT THE SUBS, AND
80 REM       THE SUBS HAVE THE SAME TIME TO CLEAR OUT YOU!
90 REM
100 PRINT:PRINT:INPUT"LEVEL OF DIFFICULTY (1-3)";LE
110 POKE11,0:POKE12,25:IFLE<1ORLE>3THEN100
120 X=USR(X):R=LE*2-1:T=120:N=53637
130 FORX=1TO3:READF(X),G(X),I(X):H(X)=F(X):NEXTX
140 X=USR(X):A=54273+PEEK(247):L=L+1:IFL<6THEN170
150 T=T-1:L=0:Q=N:Q$=STR$(T):IFT=99ORT=9THENQ=Q+1:N=Q
160 GOSUB1000:IFT=0THENQ$="* TIME *":Q=53524:GOSUB1000:GOTO380
170 IFS=0THEN210
180 POKES,32:S=S+AN:IFS>55231THENS=0:GOTO210
190 FORX=1TO3:IFS=F(X)ORS=F(X)-1ORS=F(X)+1THEN280
200 NEXTX:POKES,45
210 IFTP=0THEN400
220 TP=TP-64:IFPEEK(TP)>>178THEN340
230 IFTP<54336THENPOKETP+64,135:TP=0:GOTO250
240 POKETP+64,32:POKETP,33
250 GOSUB300:IFPEEK(57088)<>32ORS<>0THEN140
260 IFPEEK(224)THENAN=63:S=A+127:POKES,45:GOTO140
270 AN=65:S=A+129:POKES,45:GOTO140
280 FORY=F(X)-1TOF(X)+1:POKEY,32:NEXTY:V4=V4+G(X):F(X)=H(X)
290 Q$=STR$(V4):Q=53509:GOSUB1000:POKES,32:S=0:GOTO210
300 FORX=1TO3:IFF(X)=I(X)THENF(X)=H(X)
310 NEXTX:FORX=1TO3STEP2:POKEF(X)+1,32:POKEF(X),6:POKEF(X)-1,5
320 NEXTX:POKEF(2)-1,32:POKEF(2),7:POKEF(2)+1,8
330 QQ=1:FORX=1TO3:F(X)=F(X)-QQ:QQ=QQ*-1:NEXTX:RETURN
340 POKETP+64,135:POKEA-1,32:POKEA,32:POKEA+1,32:FORX=1TO5
350 POKEA,233:FORY=1TO30:NEXTY:POKEA,32:FORY=1TO30
360 NEXTY,X:Q$="YOUR SHIP HAS BEEN SUNK!!":Q=53954:GOSUB1000
370 Q$="AND YOU MUST GO DOWN TOO!":Q=54018:GOSUB1000
380 FORX=1TO5000:NEXTX:FORX=1TO30:PRINT:NEXTX
390 POKE530,0:POKE56900,1:END
400 IFF(3)-A<>831ANDF(1)-A<>703ANDF(2)-A<>574THEN250
410 LT=LT+1:IFLT<>RTHEN250
420 X=1:IFF(1)-A<>703THENX=2:IFF(2)-A<>574THENX=3
430 TP=F(X)-63:LT=0:GOTO250
440 DATA55007,15,54974,54841,10,54881,55137,20,55100
1000 FORY=1TOLEN(Q$):POKEY+Q,ASC(MID$(Q$,Y,1)):NEXTY:RETURN
```

CHALLENGER SHAPE PLOTTER
by Salomon Lederman

Here's a routine that will take a shape stored in memory and draw it on the screen. Before I explain how my routine works let me tell you what it does. Let's say that you want some complicated shape to move across the screen. Let's say that you are writing a demo and you need to have a train chugging across your screen. Well, you'd probably write a BASIC routine that would read lots of data or do lots of POKEs.

To move your train you would have to somehow erase the shape you just drew and redraw it one square over. This is quite a pain to do. The data looks very messy, wastes memory, and the routine may be slow if you have a big shape that does alot of moving around. Also erasing a complicated shape may not be easy. You can't just do a screen-clear because there is other stuff on the screen that you don't want to erase. One more point to consider, what if you have some elaborate background that you want to preserve? What do you do? Redraw it each time you move the train? Or perhaps you waste a chunk of memory to save you background, which means that you'll have two shapes to put on the screen. You must feel desperate at this point which is probably why you hate to move graphic characters. Why must cars be only one character in size? Why don't you add zing to your games by having big, detailed figures?

At this point I'll introduce my shape plotter. It can solve all your problems and it's absolutely free with this publication. No hidden costs, no monthly charge, no salesman will call, ever. This routine does everything but dice and slice.....seriously though this machine language routine will take care of your shape plotting and erasing and it will preserve your background always. The only thing that you have to do is learn how to encode your shape in the format that I use. The routine is basically a swap routine. It swaps your picture from memory onto the screen. The stuff from the screen goes into the memory where your shape was. The second time that you run the routine the swap will reverse itself so that your train is back in memory and your trees and house are back where they should be. Of course this routine is fast since it's in machine language.

To move a shape you call the routine once to put the train on the screen. You later call the routine again to erase the train (and return the trees and houses). Then you change the location pointers and you go back to square one to draw the train again, in a new location.

Your shape is stored beginning at $1500 (Hex). You can encode your shape as data or enter it through the monitor, or from tape.

The shape is stored in the format A B C1 S1 C2 S2 C3 S3...Cn Sn starting at $1500. A and B represent the starting address in low, hi format (in Hex). C1 represents the first character of your shape. Thus a block would be 161 or $A1. After your character is S1 which tells you how many squares to move until the next character. Two contiguous characters would have S1 equal to 1. Then C2 represents your next character, and so on. To end the drawing use a 0 (zero) for your Sn value.

For example, a square of four (4) blocks is encoded as follows, starting at $1500: C9 D1 A1 01 A1 1F A1 01 A1 00. Here your shape is at $D1C9.

Here's the routine:

```
.........

1800  A9 02 85 E8 A9 15 85 E9 AD 00 15 85 EA AD 01 15
1810  85 EB A0 00 B1 EA AA B1 E8 91 EA 8A 91 E8 E6 E8
1820  D0 02 E6 E9 B1 E8 D0 01 60 18 65 EA 85 EA A5 EB
1830  69 00 85 EB E6 E8 D0 02 E6 E9 4C 14 18 00
```

```
.....................
  Assembler Listing
.....................

   1800   LDA #$02
   1802   STA $E8
   1804   LDA #$15
   1806   STA $E9
   1808   LDA $1500
   180B   STA $EA
   180D   LDA $1501
   1810   STA $EB
   1812   LDY #$00
   1814   LDA (EA),Y
   1816   TAX
   1817   LDA (E8),Y
   1819   STA (EA),Y
   181B   TXA
   181C   STA (E8),Y
   181E   INC $E8
   1820   BNE 02
   1822   INC $E9
   1824   LDA $E8
   1826   BNE 01
   1828   RTS
   1829   CLC
   182A   ADC $EA
   182C   STA $EA
   182E   LDA $EB
   1830   ADC $00
   1832   STA $EB
   1834   INC $E8
   1836   BNE 02
   1838   INC $E9
   1838   JMP $1814
```

TOY PIANO
by D. Schwartz

Here is a short program that lets you play simple melodies on the Keyboard of a C4 or C8 computer. The number keys 1 thru 8 play the notes of one octave of a major scale, from middle C to high C.

The program was written by the "try it and see if it doesn't work" method, so it could be cleaned up a bit.

The values RAT(X) are the divider ratios POKED to location 57089 to produce the notes, and the BM(X) are the "BIT MASKS" representing the various keys. While the range of songs that can be played with one octave and no sharps or flats is somewhat limited, you might be surprised at some of the songs it can play, such as the Beatles' "Nowhere Man" and the Christmas carol "Joy to the World".

The only tricky part of the program is the "FOR I=.1 TO 7.1 in line 205. The reason for going from .1 to 7.1 instead of frome 0 to 7 is so that a value of 0 (zero or".1") returned by line 210 can be distinguished from a value of 0 (zero) by line 221. (Note: I(.1) is taken as I(0), I(1.1) is really I(1),etc.) A little obscure, perhaps, but it was easier to throw that in than to rewrite the whole program.

Note: The control C enable/disable is set up for a disk system. For a cassette system, change the first POKE in line 120 to POKE 530,1 and make line 155 POKE 530,0.

```
10 REM TOY PIANO PROGRAM FOR C4 AND C8
20 REM NUMBER KEYS 1 THRU 8 PLAY THE 8 MAIN TONES
30 REM OF THE MAJOR SCALE MIDDLE C TO HIGH C
40 REM
50 REM
100 DIM RAT(7):FOR X=0 TO 7:READ RAT(X):NEXT
105 K=57088
110 FOR X=0 TO 7:READ BM(X):NEXT
120 POKE 2073,96:POKE 57088,254
130 IF PEEK(57088)=0 THEN POKE 56832,1:GOTO 130
150 GOSUB 200
155 POKE 2073,173
160 IF I=   THEN POKE 57089,RAT(I):POKE 56832,3
170 GOTO 120
180 DATA 188,167,149,141,125,112,100,94
190 DATA 128,64,32,16,8,4,2,1
200 POKE K,128:PK=PEEK(K)
205 FOR I=.1 TO 7.1
210 IF PK=BM(I) THEN RETURN
220 NEXT I:POKE K,64:IF PEEK(K)=128 THEN RETURN
221 I=0:RETURN
```

Cassette Save/Hex Memory Dump

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

Here is a cassette save routine by which you can save your machine language programs on tape. This save routine incorporates a hex memory dump which permits you to read the code as it is being transferred to tape. The machine code is executed from a Basic start by way of the USR function, after you have entered the start and end addresses. The following is a list of both Basic and machine language routines:

```
LIST
 1Ø P=247:INPUT"INPUT THE STARTING ADDRESS";A$:GOSUB4Ø
 2Ø PRINT:INPUT"INPUT THE ENDING ADDRESS";A$:GOSUB4Ø
 3Ø PRINT:PRINT:POKE11,34:POKE12,2:X=USR(X):PRINT:END
 4Ø IFLEN(A$)<4THENFORX=LEN(A$)+1TO4:A$="Ø"+A$:NEXTX
 5Ø B$=RIGHT$(A$,2):GOSUB6Ø:B$=LEFT$(A$,2):GOSUB6Ø:RETURN
 6Ø FORX=2TO1STEP-1:N=ASC(MID$(B$,X,1))-48:IFN>9THENN=N-7
 7Ø A=A+N*16^(2-X):NEXTX:POKEP,A:P=P+1:A=Ø:RETURN
OK
```

Location	Machine Language	Mnemonics
Ø222	A9 ØD	LDA #$ØD
Ø224	2Ø 8A Ø2	JSR $Ø28A
Ø227	A9 2E	LDA #$2E
Ø229	2Ø 8A Ø2	JSR $Ø28A
Ø22C	A5 F8	LDA $F8
Ø22E	2Ø 9C Ø2	JSR $Ø29C
Ø231	A5 F7	LDA $F7
Ø233	2Ø 9C Ø2	JSR $Ø29C
Ø236	A9 2F	LDA #$2F
Ø238	2Ø 8A Ø2	JSR $Ø28A
Ø23B	A2 ØØ	LDX #$ØØ
Ø23D	AØ ØØ	LDY #$ØØ
Ø23F	B1 F7	LDA $F7,Y
Ø241	2Ø 9C Ø2	JSR $Ø29C
Ø244	A9 ØD	LDA #$ØD
Ø246	2Ø 15 BF	JSR $BF15
Ø249	A9 2Ø	LDA #$2Ø
Ø24B	2Ø 2D BF	JSR $BF2D
Ø24E	E6 F7	INC $F7
Ø25Ø	DØ Ø2	BNE $Ø254
Ø252	E6 F8	INC $F8
Ø254	38	SEC
Ø255	A5 F9	LDA $F9
Ø257	E5 F7	SBC $F7
Ø259	A5 FA	LDA $FA
Ø25B	E5 F8	SBC $F8
Ø25D	1Ø Ø3	BPL $Ø262
Ø25F	6Ø	RTS
Ø26Ø	EA EA	NOP

Location	Machine Language	Mnemonics
Ø262	E8	INX
Ø263	EØ 1Ø	CPX #$1Ø
Ø265	DØ 2Ø	BNE $Ø287
Ø267	A2 ØØ	LDX #$ØØ
Ø269	A9 ØD	LDA #$ØD
Ø26B	2Ø 2D BF	JSR $BF2D
Ø26E	A9 ØA	LDA #$ØA
Ø27Ø	2Ø 2D BF	JSR $BF2D
Ø273	A9 2Ø	LDA #$2Ø
Ø275	2Ø 2D BF	JSR $BF2D
Ø278	A5 F8	LDA $F8
Ø27A	2Ø 91 Ø2	JSR $Ø291
Ø27D	A5 F7	LDA $F7
Ø27F	2Ø 91 Ø2	JSR $Ø291
Ø282	A9 2Ø	LDA #$2Ø
Ø284	2Ø 2D BF	JSR $BF2D
Ø287	4C 3D Ø2	JMP $Ø23D
Ø28A	2Ø 2D BF	JSR $BF2D
Ø28D	2Ø 15 BF	JSR $BF15
Ø29Ø	6Ø	RTS
Ø291	85 F6	STA $F6
Ø293	2Ø AD Ø2	JSR $Ø2AD
Ø296	A5 F6	LDA $F6
Ø298	2Ø B1 Ø2	JSR $Ø2B1
Ø29B	6Ø	RTS
Ø29C	85 F6	STA $F6
Ø29E	2Ø AD Ø2	JSR $Ø2AD
Ø2A1	2Ø 15 BF	JSR $BF15
Ø2A4	A5 F6	LDA $F6
Ø2A6	2Ø B1 Ø2	JSR $Ø2B1
Ø2A9	2Ø 15 BF	JSR $BF15
Ø2AC	6Ø	RTS
Ø2AD	4A	LSR A
Ø2AE	4A	LSR A
Ø2AF	4A	LSR A
Ø2BØ	4A	LSR A
Ø2B1	29 ØF	AND #$ØF
Ø2B3	Ø9 3Ø	ORA #$3Ø
Ø2B5	C9 3A	CMP #$3A
Ø2B7	3Ø Ø3	BMI $Ø2BC
Ø2B9	18	CLC
Ø2BA	69 Ø7	ADC #$Ø7
Ø2BC	2Ø 2D BF	JSR $BF2D
Ø2BF	6Ø	RTS

If you want to execute the program from the monitor, place the start address of the code to be saved at $F7,F8 (lo,hi) and the end address at $F9,FA (lo,hi). Also you should add the following jump:

Location	Machine Language	Mnemonics
Ø25F	4C 43 FE	JMP $FE43

Actually, the jump can be made to a location of your choosing but I find this one convenient to use since you remain in the monitor.

While this routine is great for saving your machine language code, it is also very useful for simply reading code from the hex memory dump. I haven't bought my own assembler yet, but for reading my programs in machine code this program is better then nothing. The speed of the program is slowed down, though, because of the output of data to the cassette machine. The following few changes can increase the speed of the program by the elimination of the cassette outputs:

Location	Current	New
Ø241	2Ø 9C Ø2	2Ø 91 Ø2
Ø244	A9 ØD	EA EA
Ø246	2Ø 15 BF	EA EA EA

Note:: My system is a C2-8P, and this program takes advantage of the 32x64 screen width, so all of you C1P users aren't going to get the screen output that I had intended. Also the cassette port is located at $FCB1 and not $BF15, as on my system. With this change the save routine should execute, but see Micro mag., p. 11:17 " A Close Look at the Superboard II ", by Bruse Hoyt.

"We did it! We crossed a computer with King Kong and got a hairy reasoner!"

C 1 P is alive and well in . . . GERMANY

BY KLAUS ERNST

I just got some recent issues of the German hobby computer magazine CHIP. I was very pleased to see a lot more (from previous issues) ads for OSI machines, hardware mods, and software. They even had a listing of a BASIC program by Axel Unterschuetz for saving machine programs (for C1P). Here is a translation of the description and the INPUT and PRINT commands.

"It got me a little annoyed, that the skimpy OSI-C1P Monitor does not provide the possibility to save machine code programs on cassette. There is a load routine provided though, which is called by "L". It can handle the following commands (from cassette):

1) Period(.) to signal addresses being loaded (address mode)

2) Slashes(/) to signal data being loaded (data mode)

3) HEX numbers(0-9, A-F) in ASCII

4) in data mode: "CR" to increment address

5) in address mode: "G" to start program at specified address

I wrote this simple program to be able to save simple, short machine code routines (clear screen, etc.)

You enter starting and last address, the program outputs (to cassette port):

1) 3 periods(...) to switch to address mode

2) starting address (HEX)

3) 3 slashes (///) to switch to data mode

4) HEX codes, separated by RETURNs

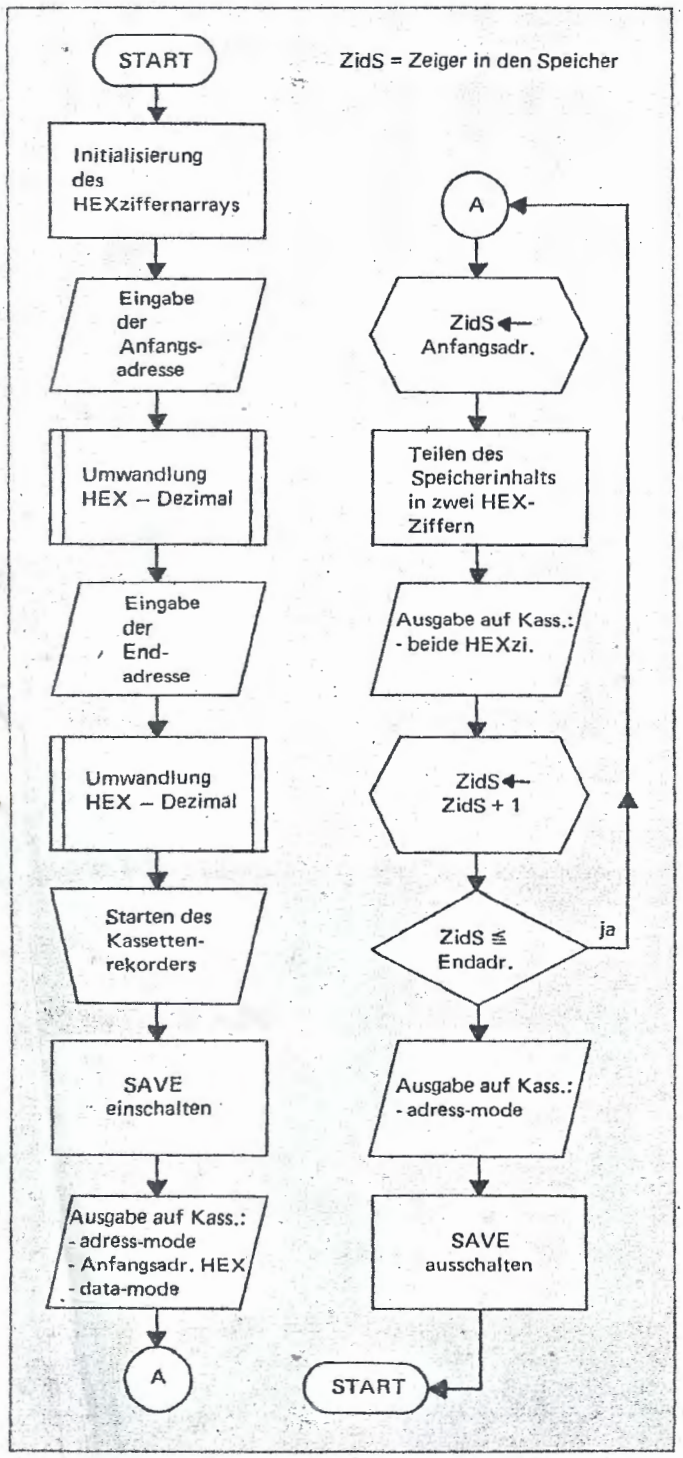
5) 3 more periods (...) to stop loading

Saved programs can be loaded by hitting "BREAK" then "M" and "L" and then starting cassette.

After loading hit "BREAK",unless there are additional commands on cassette,e.g. the starting address and "G",which would make the program selfstarting."

```
7Ø INPUT"STARTING ADDRESS";AD$
1ØØ INPUT"LAST ADDRESS";AD$
12Ø PRINT:PRINT"START RECORDER"
125 PRINT"TYPE '.' WHEN READY "
REMS
3Ø REM STORING ARRAY
18Ø REM DECIMAL TO HEX CONVERSION
2ØØ REM OUTPUTTING HEX CODE
23Ø REM REACHED LAST ADDRESS
25Ø REM SAVE OFF
27Ø REM NEW RUN
3ØØ REM HEX TO DECIMAL CONVERSION
35Ø REM SEARCH FOR PROPER DEC VALUE
```

Zeichen, an der momentan angezeigten Adresse mit der Programmausführung zu beginnen
Um einfache, kurze Maschinencode-Routinen wie „Clear Screen“ oder ähnliche speichern zu können, habe ich ein einfaches BASIC-Programm entwickelt. Es fragt nach Anfangs- und Endadresse des zu speichernden Bereichs und gibt dann aus:
1) 3 Punkte (...), um in den adress-mode zu gelangen
2) die Anfangsadresse (HEX, 4stellig)
3) 3 „slashes“ (///), um in den data-mode zu gelangen
4) die HEX-Codes, durch RETURN voneinander getrennt
5) wieder 3 Punkte, damit nicht aus Versehen weitere Daten geladen werden
Die so auf Kassette gespeicherten Programme können ohne weiteres vom Monitor eingelesen werden, indem man nach „BREAK“ und „M“ auf „L“ drückt und die Kassette startet.
Nach dem Einlesen muß man wieder auf „BREAK“ drükken, es sei denn, man hat noch weitere Befehle auf der Kassette, zum Beispiel die Anfangsadresse des Programms und ein „G“, dann wird das Programm gleich ausgeführt.



Axel Unterschütz

Beispiel (= RETURN)

```
Anfangsadresse? FE00
Endadresse? FE04

Starte Rekorder!
Tippe „.“ wenn fertig!
? .
...FE00
///A2
28
9A
D8
EA
...
Anfangsadresse?
```

```
 10  DIM C$(15)
 20  DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
 30  REM EINLESEN DES ARRAYS
 40  FOR I=0 TO 15
 50  READ C$(I)
 60  NEXT I
 70  INPUT "Anfangsadresse"; AD$
 80  AN$=AD$: GOSUB 320
 90  AN=AD
100  INPUT "Endadresse "; AD$
110  GOSUB 320: EN=AD
120  PRINT: PRINT"Starte Rekorder! "
125  PRINT"Tippe '.' wenn fertig!"
130  INPUT X$
140  SAVE: PRINT"..."; AN$
150  PRINT"///";
160  FOR J=AN TO EN
170  D=PEEK(J)
180  REM UMWANDLUNG DEC-HEX (2stellig)
190  E=INT(D/16)
200  REM AUSGABE DER HEX-ZAHL (DES CODES)
210  PRINT C$(E); C$(D-16*E)
220  NEXT J
230  REM ENDADRESSE ERREICHT
240  PRINT"..."
250  REM SAVE AUSSCHALTEN
260  POKE 517,0
270  REM NEUER LAUF
280  RUN
290  REM
300  UMWANDLUNG HEX-DECIMAL
310  REM
320  XI=4096
330  AD=0
340  FOR L=1 TO 4
350  REM SUCHE NACH ZUGEHOERIGEM DEC-WERT
360  FOR L1=0 TO 15
370  IF MID$(AD$,L,1)=C$(L1) THEN 390
380  NEXT L1
390  AD=AD+L1*XI
400  XI=XI/16
410  NEXT L
420  RETURN
```

Das Programm zum Abspeichern des Maschinencodes

Programming the 6502- Machine Language for the Challenger Computer
Class 1, by Salomon Lederman November 6, 1980

These pages are intended as an outline for reviewing the materials that I intend to cover in this first class. The only requirement for the course is access to a table of 6502 Op Codes. I strongly suggest that you purchase a book for programming our chip, the 6502. Lance Leventhal authors 6502 Assembly Language Programming, which I reccomend. Most books on the 6502 are written with the SYM,KIM, AIM...computers in mind. Try to avoid books that specialize heavily in these micros since they generally discuss special routines and specific hardware that is not very relevant to OSI computers.

Part I- The following skills are essential to the understanding of machine language:

Hex to Decimal and Decimal to Hex conversions. It is useful to understand binary also.

Simple arithmetic in binary and hex.

Know how to enter simple programs through the ROM monitor.

You should learn how to use an Assembler though it will not be vital for at least the first lesson.

Part II- Some basics about software architecture:

The 6502 recognizes 3 main registers, the Accumulator, the X and the Y registers. All arithmetic is done in the accumulator. The X and the Y registers are used as counters.

All memory addresses are 16 bits wide, or 2 bytes. Thus an address is made up of 4 hex digits. A274 is an address, and so is BD11. Thus the 6502 can recognize any one of 65536 addresses.

Op codes are one byte hex numbers. For example, A2 is an Op code and so is E8. Not all hex numbers represent Op codes. Op codes represent commands for the 6502. Through the use of these commands you can precisely control your micro, quickly and efficiently.

A command consists of an Op code followed by zero, one or two more bytes of information. These extra bytes set parameters for the main command.

Part III- Follow these examples carefully. You will be able to pick up many ideas by actually trying them. For this first class I

will use only a few of the simplest Op codes as building blocks for the first programs.

```
Program 1.   0500 A9 A1      LDA #$A1
             0502 8D C9 D1   STA $D1C9
             0505 00         BRK
```

This program will place a block on the screen. The first instruction used is LDA# which means to load the accumulator with some numeric value which is represented by the next byte in the program. The # symbol means that the instruction is using the next byte as a constant, as opposed to a variable. The $ symbol indicates that the number referred to is in hex. In your graphics manual you will see that A1 represents the decimal number 161 which is the white block. And of course A9 is the code for LDA#. The next instruction is STA$. By this notation I just mean that STA refers to an address. STA is the mnemonic store the accumulator in the address... In this case the Op code 8D refers to the address encoded in the following two bytes of the program. Note that the address is in hex ($), and that the assembly language coding for the address is the hex value of some screen location. However for some weird reason the 6502 designers decided that the two bytes of of an address should be reversed in their order and they always are. Thus $D1C9 is encoded as C9 D1. As an exercise you ought to be able to convert it to decimal. If you don't have a C1 and you don't see a block then you should change bytes at 503 and 504 to some visible screen location. The code 00 tells the 6502 to stop executing your program. If you want to run the program again you have to hit the BREAK key, enter the monitor and run it again.

To summarize: This simple program loads the accumulator with A1 and stores the contents of the accumulator in $D1C9. The program in this case is stored in 0500 through to 0505 so it's a six byte program. Note that in BASIC the same instructions would be POKE53705,161 which takes up ten bytes. Very often machine language programs run in much less memory than their BASIC counterparts and there's no comparison in terms of speed.

In case you are not too used to the ROM monitor here's how you enter this program. Hit BREAK, enter M, then the following:
.0500 / A9rA1r8DrC9rD1r00 The 'r' represents a carriage return.
To execute the program type .0500G

To speed things up a bit I will cut down on the detailed explanations.