the AARDVARK JOURNAL

february 1981 vol.1, no.6

CHANGES AT AARDVARK

AARDVARK has now grown bigger. We have more staff working, more programs to offer, and now we have more space. An idyllic era has come to an end with the moving AARDVARK out of its quarters attached to our home to a much more spacious office a short distance away. AARDVARK now has its own building - small though it may be. We didn't have too much choice about the change. We had gotten so crowded that we were beginning to make mistakes just from being pushed too often to find a place to put something. We'd gotten to the point whre you had to pick up two things to find room to put down one, and then you had your hands full of whatever you'd picked up. We started to get errors in documentation and errors in programs simply due to the fact that it was hard to keep things straight in too small an area. There will, unfortunately, be some changes in the kind of services we can offer now that the business has grown larger. We are going to try to keep the same kind of close contact that we have had with our customers and readers in the past and therefore the phones will ring both at the old quarters and at the new office as well. You'll still be able to chat with me or get answers to questions after regular business hours. However, all the files are at the new office and if you have specific questions about your order or your subscription where we have to refer to the files to get you an answer, you'll have to call between 9 am and 4 pm (EST), or call one day and wait until the next day for an answer. Also, we are going to stop answering the phone on Sunday and after about 9 at night. We like keeping the phones open late as it saves our customers money if they call on evening rates, but it has been getting a little silly recently, and we are going to cut the phones at a reasonable hour.

We think you can summarize it all by saying that despite the fact that this is a growing enterprise, we plan to keep it as informal and friendly as we can.

BEGINNER'S CORNER
TYPING IN MACHINE CODE PROGRAMS

You don't really have to know how to program in machine code in order to make use of a number of the routines that have been published in places like MICRO, COMPUTE, and (best of all) the AARDVARK JOURNAL. However, they do not normally bother to explain just how you are suppose to get that long list of stuff into your computer. It can be somewhat confusing the first time you try it.

There is a fairly standard form for printing these programs, which, once you understand what they are doing, makes it easy to get into your computer one way or the other.

Rather than give examples here, I'll refer you to the aritcles in this months' journal. They all follow the format that we are going to describe.

The first thing we have to define is the funny word 'field'. A field is all the letters in a row up until you hit a space. You'll notice if you look at a program from, say, MICRO, that if you stand back and look at it there are actually columns down the page with bits missing out of each column here and there. These are actually seven fields of varying length. On a well filled line where all seven fields are used, the first thing you see is a number which is normally four digits long and which may be from 10 to 60000. That is the line number if you are going to use an assembler. The next field is a four digit hex number from $0000 to $FFFF. That is the actual memory location this line would be stored at if it were assembled in memory as shown here. The next field over is normally from one to three pairs of hex digits. It can be either a single pair, two pair, or three pair of digits. That is the actual data that would be entered at that address if the program were to be put in memory as printed. If there is only a single byte, that is, a single pair of digits, then they are placed in memory at the address which is shown in

field #2. If there are two bytes of information, then they are placed into memory at that address and the next address. If there are three bytes, they are entered at the given address and the succeeding two. That's also why the numbers in the second field are not continuous. For instance, if you put in a two byte peice of information at location 0203, the next place you can put something is 0205 as that pair went actually into 0203 and 0204. The fourth field over is the first thing you need if you are actually going to put this in with an assembler rather than hand enter it. That column, if it is filled (and it is not always) is the label column. We'll get back a little later to explaining how to enter that label. The next field over, which by the way is always filled, is called the op code. It is the neumonic representaion of the number code that appears as the first pair of hex digits in field #3. For instance, if in field #3 the first thing you see is A9, you will see LDA (load accumulator) as the op code in field #5. The sixth field is called the operand field. It may or may not be filled. If the field is filled, it can be filled with a number of different things. the label for another address or variable, hex number (they always begin with a $ sign), or a decimal number.

The last field is a very important one, but thank goodness we don't have to type it in because there's an awful lot of it. In a well written machine code program there are a lot of remarks for a couple of reasons: (1) It's easy to forget why you did something in a machine code program. It's even harder to read native code here than it is in BASIC and (2) since the remarks are not compiled into the machine, they don't take up space in run time the way they do in BASIC. However, since we only want to type the program in and get it working and do not care about the remarks since we have them on paper, you can ignore the last field when you are typing the program in.

TYPING IT ALL IN

First, the simple method (assuming that you do not have an assembler of any sort). If you don't have an assembler and you have to use the OSI monitor, all you're going to need is the second field and the third field on these listings. Start by hitting (BREAK) and (M) and you should see four zeros and a 4C appearing in the upper left hand corner of the screen. The first thing you have to do is to go to the starting address of the program. To do that you press (.) - period - type in the four digits of the beginning address and they should appear on the screen as you type them. If you make a mistype, just continue typing and the characters will rotate through. When you are at the first sequential address of the program, you hit the (/) which puts you in data entry mode. You then enter the first two digits in field #3, you should see them appear next to the address you just entered. When they are right, hit (RETURN) and the next address will be displayed. Put the next two digits from the data field at this address. You simply keep hitting return and entering the data as you go down the list as they are normally given sequentially. At any one time you should be able to check by looking at the listing to see what you should have at that address to see if you are right. If you get out of sequence, hit the (.), go back to the address before the error, hit (RETURN) and read the addresses and data bytes until you find where you got out of step. When you have the entire program entered into memory, you* should store it before you attempt to do any manupilation with it. Now, with a OHIO SCIENTIFIC machine, there are a number of different ways of storing the system and they all need outside help of some sort, that is, they all require a program that is not in the monitor. Unfortunately, that is the scope of another article which we will try to squeeze in elsewhere in this journal but which we can not get in at this point. Suffice it to say, that if you have the means to store the program, you should store it at this point before you run it. In any case, you have either stored the program or decided you're not going to store it, you have to identify the start address which is normally given in the text accompanying the program. You hit the (.), type in the address you want to start at and hit (G) for GO.

Now you can't do much with the program in terms of putting it anywhere else in memory or modifying the code. It has to be put in pretty much as the original author planned it and the whole process is rather error prone due to the fact that you are reading nothing but hex digits and its easy to mistype one. It is, however, a simple procedure made simplier by the fact that

OHIO SCIENTIFIC has included the basic routines we need in their monitor ROM. Most systems do not have similar facilities.

THE EASY WAY - USING AN ASSEMBLER

There are a number of assemblers around and we're going to include an article elsewhere in this journal on the ones that are available, but they all work just about the same as there is a standard pattern for the syntax in an assembler the same way there is a standard pattern to BASIC syntax. Warning - the keyword is "about". Variations in assemblers do exist and cheaper assemblers may lack some of the capabilities such as labeling that the more expensive ones have.

To enter the program in assembler, you are going to need to read the first column (or field) which is the line numbers, the fourth column which is the label column, the fifth column which is the op codes, and the sixth column which is the operands. The only thing that is particularly confusing about this is that the stuff for the people without assemblers is in the middle between the first and the fourth columns, and while you don't need any of that its still in your way. To type in a program in assembler, you begin by loading the assembler. You then type in the line number which appears in the first column on the left. Second, look at the label column which is the fourth column over. If there is a label there it is to be typed in immediately following the line number. There is to be no space between them. Put in at least one space after the line number or the line number-label combination. Third, type in the neumonic in the op code column such as LDA. You then have to leave at least one space between the op code and the operand if there is one such as $00, or LOOP 1, or whatever is in the operand column. You can then go on to the next line number as you don't really need to add the remarks. Remarks

The AARDVARK JOURNAL is published six times yearly by Rodger Olsen, AARDVARK TECHNICAL SERVICES, 2352 S. Commerce, Walled Lake, MI 48088. Subscription rate $9.00, overseas $14.00. Bulk rate postage (permit #5) paid at Walled Lake, MI 48088.

are very important in assembler programming as it is easy to forget what you did. But you already have a copy Of the remarks on paper so you don't have to type them in again. The process may actually take a little longer than direct hand coding, but it gives you a couple other advantages. The process is more error free, even though there is more typing, since the typing vaguely looks similar to English, it can be read more easily and errors can be detected more rapidly. You also have a little more flexibility in things like where to put the program. Most assemblers, although not all, will allow you to place the program virtually anywhere in free memory, normally the starting address specified in the first line or two with a star (*) or the term 'ORG=' and by simply changing that address the assembler will be instructed to adjust all of the other addresses to fit. You, therefore, have some advantage in using the assembler.

When you have it typed in, the same cautions apply. You should definitly store the source code (That's what you have ben typin mpile it and you should definitly store the object code (what you get when the sour the assembler) before you attempt to run it. In the absence of that, be brave (or foolhardy) by exiting the assembler, typing in the address of the program and hitting (G). That should get you up and running - if you typed it all in right.

MACHINE CODE UTILITIES

The almighty assembler. Now even you neophytes should know what an assembler is. That's a program that allows you to type in an English like neumonic and which then translates that into a machine code number. Beyond that there are a lot of features that assemblers may or may not have. You may be able to assemble anywhere in memory or only in limited areas. You may or may not have relative branching and labeling capacities, that is, what the branch from one section to another should be and it may or may not remember the names of places where you want to store things or jump to. Once the assembler assembles a program in memory, you then break out of the assembler and run the machine code program. There are a number of assemblers available for OHIO SCIENTIFIC equipment.

The old standby is the MOSTEK assembler which is sold by OSI. It runs $35 at your local dealers. It has most the features you would want in a medium size assembler. It will accept numbers in hex, binary or decimal, supports relative branching and labels, will store the program in memory or on tape, and has a large number of error codes which it reports back in case you made a mistake. If you are a serious machine code programmer, it's a good deal. It does take seven to eight minutes to load unless you re-record it at higher baud rates. It also burns up a lot of memory, takes 8K to run and doesn't leave a lot of space left over to assemble programs into. Remember, you have to assemble the programs into areas that are not currently being used by the assembler itself. However, at $35 and being full featured, it is a good deal for the serious or soon to be serious programmer. I might note at this point that OSI also sells a disk based assembler which is virtually identical, but has a few more features. Unfortunately, OSI is having a hard time selling it as it was inadvertently included on almost every systems disk, including games disks, sold by OSI over the past year or so. Sales of the seperate assembler are somewhat slow.

Moving on for people with small machines, there is an assembler put out by Bill's Micro Services (210 S. Kenilworth, Oak Park, IL 60302). Bill hasn't been advertising recently and his company seems to be somewhat comatose, however, he did sell a 3K assembler editor for the C1P, and also said it was going to be available for the C2/4 machines. I have not used the program, but I have had reports from several readers that it is a usable and workable assembler although it is somewhat limited in features, that is to say it does not support relative branching and labels. However, I have also heard that while Bill seems to be lying somewhat low, if you send him $12.95 he will still send you his assembler editor for the C1P. That might be a good choice for the beginning computerist. On the underside, there is also a mini-assembler available from AARDVARK. We sell one in BASIC for $9.95. I have to admit that it is more of a training tool than it is a serious assembler, and it does not come close to matching all the features of a big, full sized MOSTEK assembler. It is a two pass assembler that supports indirect addressing and labels and it uses all the standard MOSTEK neumonics. It is however limited to programs of 256 bytes long (actually fairly long in machine code) and longer programs have to be assembled in two or more passes. It is an easy to use and inexpensive assembler for the new computerist. It also outputs the program either as a self loading 65V machine code format tape or as DATA statements and as it is not assembled directly to memory it allows you to assemble programs in the same area that is being used by the assembler itself. The process is to output a tape that when reentered in the machine will overload the area that the assembler occupies. Its a fairly handy beginner's tool.

On the other end of the scale, if you have a C8 complete with 8" disks, Pegasus Software (P.O. Box 1004, Dept AA, Honolulu, HI, 98616) offers a disk based assembler that has a few more features that the OSI assembler and which sells for $75. There have also been a number of mini-assemblers for PETs and APPLEs published in such places as KILOBAUD and MICRO. Those done in MICROSOFT BASIC will often transfer easily to an OSI machine if you have patience, masochism, and a buck and a half for the magazine.

The next thing you need is an editor. Now, editor is a very loose term in microcomputing circles. You've got to watch out what you're buying because you could be buying almost anything. Technically, all an editor consists of is a program that will modify another program. In that sense, you already have an editor already built into

every OSI machine. That little thing you come up with when you hit the M known as the monitor is actually a little machine code editor. It allows you to change memory locations, and to execute programs and to load programs. A good editor has both editing and troubleshooting features. At a minimum it allows you to change data in memory locations enter programs and save them on tape or disk, and provides some sort of troubleshooting facilities such as a break point feature. OSI, again, sells one of the best on the market. Their $15 editor allows you to examine the memory locations, change them, search through memory for strings or data bytes, does block moves of memory and relocates programs. You get all that for $15, not a bad deal. I've seen a few others on the market, but haven't seen anything that I would recommend as being worthwhile. I'm sure there are a number of them, but I'm just not aware of them.

If you have a taped based machine, you also need someway to save programs. This is extremely important as OSI did not provide a method of saving programs from the system itself. You must have some kind of external editor that saves them. There are several ways to do it and there are several programs out on the market. Here at AARDVARK, we sell an AUTOLOADER tape for $5.95 that saves programs in a self starting 65V format. We also sell what we call a POKER MAKER routine for $5.95 that takes programs that are already placed in memory by the assembler or by hand assembly and puts them out to the cassette as DATA statements so they can be read from BASIC. We also sell the C1E and C2E monitor ROMs which are system ROMS with extended monitors in them and which are capable of outputting a machine code tape.

There are a lot of other ways to handle it. If you have been programming in BASIC for a while, it is fairly easy to write your own routine which will PEEK memory locations and print the data bytes out to the cassette. There have been a lot of very simple routines written for the KIM and the APPLE which can be adapted fairly easily to the OSI if you happen to have copies of the JOURNAL around. OSI has also published an AUTOLOADER program, which unfortunately didn't work too well, but once it's debugged it's fairly handy.

Also a number of the assemblers we've talked about here are capable of outputting machine code tapes. In any case, you should plan, before you begin to program, how you are going to save the program when you are done. By the way, disk people don't have to worry about it as their disk systems have the SAVE and CALL commands which allow them to write memory directly up to the disk.

DEBUGGING TOOLS

One of the hardest things about writing machine code programs is debugging the blasted things. You don't have the interactive features that you have with BASIC, the programs are not so easily modified for checking, and in general they are a pain to debug. Therefore, you should plan, if you are going to get into machine code properly, to have some means of debugging programs. There are a number of them available that make the job fairly easy. The first tool you can have is the extended monitor. Most extended monitors, including OSI's and the one in the C1E and C2E chips support what is called break points. They allow you to put a break in the program where the machine will stop, read out the contents of the registers and then continue. We are also publishing a breakpoint program for tape based machines in this copy of the JOURNAL.

While for the machine code die-hards and other masochists, the break points alone are sufficient for any real man to debug his program with, I personally prefer a trace program. Trace programs in machine code are the Cadillacs of the machine code utilities. Most trace programs are actually 6502 emulators, that is they actually pretend to run the program and then report back to you what happens during running. It is difficult from the user point of view to see the difference between a pretend run with phony registers and a real one with real registers. What happens is that while you execute the program, the trace program will show the content of the registers and the program counter (the memory location the computer is running), and other interesting data as the program executes. Most of them will either single step through it or will step through a given number of bytes before stopping and displaying the contents of the registers.

That makes machine code much, much easier to debug, you can actually watch the program execute. There are at least three or four of them available for the OSI. On the top end of the scale, Pegasus of Hawaii offers a trace program for floppy disk, $95.00.

Moving a little further down the scale, AARDVARK now sells a trace utility with some other editor features such as direct string manipulation and memory manipulation for the 5 1/4" or 8" disks for $24.95. That program takes about 800 bytes and should be available for tape machines around March 1st for $19.95. For those of you with little tighter budgets and less expensive tastes we also offer a trace and single step program in BASIC for the grand sum total of $12.95. Now this is a BASIC program that will single step through a machine code program and therefore is somewhat limited as the machine code program must be in space not used by the BASIC program. It therefore will trace programs between $1C00 and $1F00 or any program located above the first 8k. This is another handy training tool as it allows you to watch machine code already in your system being executed.

BOOKS

The next thing you are going to need is a book of 6502 machine code. There are lots of them on the market - most of them are junk. Particularly stay away from the Zaks books. The early issues seemed to have bugs in darn near every program and I haven't seen any evidence that they have been fixed. Out of the mess, I have found three books that have been particularly useful to me. One of my favorites is PROGRAMMING AND INTERFACING THE 6502 by Marvin D. Long. Its a Howard Sams book and therefore is overpriced at $13.95, but it seems to be more error free and clear than most of the books in the field. MOSTEK also puts out a book called PROGRAMMING AND INTERFACING THE 6502 which is

very dry reading and darn near boring, but which contains largely error free information and which has all the information you really need if you want to bore thorough it long enough. I also found SCLIBI's 6502 GOURMET COOKBOOK to be of a help after I had read the other two.

Starting in the next issue, the AARDVARK JOURNAL is going to carry tutorials on machine code programming but we're going to concentrate on techniques and we will not attempt to teach the basic stuff you should be getting out of the other books on the market. We will, for instance, not discuss what the term LDA means, but will hopefully discuss some nice ways to use it.

LOADING AND SAVING WITH CASSETTE

by Chris Loelke

I'm sure that we have all had our problems on occasion loading and saving programs on cassette. Usually, the error rate is even greater with 'foreign' cassettes (ones that we paid good money for). The following procedures should help you to eliminate a lot of errors.

Errors are caused by one or more of the following items.

1) computer problems
   a. BAUD rate not adjusted correctly
   b. multivibrator adjusted incorrectly
   c. defective component or PC track.
2) cassette problems
   a. head alignment
   b. cassette player speed
   c. wow and flutter (wobbley sound)
   d. frequency response
   e. dirty machine
   f. poor quality drive mechanism

To determine whether the cassette or the computer is at fault is relatively easy. Beg, borrow or steal a different cassette machine. Make sure that is is known to be in good shape. Just because it is new is not an indication of its condition. Over half of all new machines suffer from head-alignment or speed errors. Generally, though, a player that costs in excess of $300 can be trusted.

If you are still having problems, it is time to tear into the computer and make a couple of simple checks. To check the baud rate a frequency counter is required. The counter is connected to pin 4 of the UART (6850). This is one of the larger chips and it resides near the 6502 CPU. The frequency should read 4800 HERTZ plus or minus 5 HERTZ. If yours is not within that range, try adjusting R17 (the blue trimpot near U13 (555)). A word of caution, if the frequency was out by more than 10%, you may find that you can no longer load some of your own software after performing the test. Such are the breaks.

The next adjustment to be checked is the pass-band of the multivibrator (U22 (74123)). It is adjusted with trimpot R26 near U22. To adjust it, an oscilloscope would be handy, but we will adjust it here without one. The setting of the pot appears not to be very critical. Turning it all the way to one end will cause the computer to stop loading completely; while turning it towards the other side will give about a 50% error rate. The ideal setting seems to be near the end where the computer stops loading. To get the closest setting, reduce the volume of the cassette to the point where you get a lot of errors, then try and reduce them as much as possible with the trim pot. Som patience and numberous tries should give good results. I played with mine for about an hour before I got the feel of it. The PEEK-A-PORT program in the pre 1981 AARDVARK catalogs and the routine for making a test tape (KILOBAUD, Sept, 1980) should be of some help in the adjustment process.

If you have decided that after all you cassette machine is the culprit (most cases), a good cleaning and a couple of checks may be helpful. To perform a head alignment by ear, insert the tape and adjust the screw that tensions the sound as the tape is being played. To perform a speed test, compare the playing time of your own machine with that of at least three other machines. If the error is greater than 1%, the speed will have to be adjusted. Articles on how to clean your machine and more details can be found in many books and magazines. I found the article in June, 1980, KILOBAUD very helpful.

If you decide to perform a head alignment, make sure that you are aligning it to a known good tape. To make absolutely sure, it is advisable to purchase a tape made especially for this purpose.

All of the above adjustments refer to the C2/4P, the C1P has the same adjustment, the only difference being the locations of the trimpots and their designations. I have done all the adjustments described above on my own machine and have not suffered from a misload in over 6 months. The following program will help diagnose whether your cassettes or the recorder are at fault. There are two programs in the listing. Program 1 (lines 10-40) will generate a program of all the characters from 14 to 255. Program 2 does a comparison check to see if the characters saved on the tape agree when you try to load them back into the machine. After you have typed in the program, type "RUN" (no C/R yet) and start the recorder. After the leader has passed, hit C/R. You should save for at least 20 minutes (so use a long tape). Rewind the tape, wait for the leader to pass and then type 'RUN80'. If the program stops immediately, try again as false starts can occur. You should be able to go for at least 24000 cycles before getting an error. To see if the program is operating correctly, shut off the recorder and the error statement will come up. If you get an error, it is necessary to rewind the tape to the beginning as continuation is not possible.

```
1 REM CHRIS LOELKE
10 REM TAPE TEST FOR C1P, C2/4/8P
15 SAVE
20 X=14
30 POKE14,0:?CHR$(X);: X=X+1
40 IFX=255THEN20
50 GOTO30
60 REM ** RUN 80 FOR PLAY TEST **
70 REM CHANGE A TO 61440 FOR C1P
80 ?:?"RUNNING TEST":?
90 X=14:A=64512
100 WAITA,1:B=A+1: IFPEEK(B)=13 OR PEEK(B)=0
THEN100
112 IFPEEK(B)=10THEN100
113 IFPEEK(B)<>XTHEN ? "ERROR AFTER ";Y;"
CYCLES":END
115 X=X+1:Y=Y+1: IFX=255 THEN90
120 GOTO100
```

A MODIFICATION BOX
by Russ Terrell

Here is an idea for any C1P users who have joysticks and/or a light pen. The problem you face is that you do not want to hardwire them into the C1P and thereby take away from its mobility. Also, you end up with wires running everywhere. The solution to this problem is an utility box that will hold both the light pen and joystick circuits.

BILL OF MATERIALS

1) Experimenter box with aluminum cover 5 1/16 x 2 5/8 x 1 5/8. (cat. 270-233)
2) 16 pin DIP jumper cable - (cat. 276-1976)
3) 20 conductor ribbon cable (cat. 268-770)
4) 2 16 pin DIP sockets (or wire wrap) (cat. 276-1998 or 276-1994)
5) Male 12-pin MOLEX connector
6) Project baord, mounting hardware
7) Parts for the other projects.

** NOTE: I used the box listed and built the light pen circuit to fit the box, but have not, as yet, added the joystick circuit. If you have already built both, get a bigger box. Many different sizes and shapes are available.

1) Cut a project board to fit inside the case. Mount one of the 16-pin sockets to it. Use the method that works best for you (wire wrap, point to point, etc). If you already have the board(s) built, cut a small board to mount the 16 pin socket on.
2) Build the project(s) on the rest of the board.
3) Next, wire the 16 pin socket to your projects. The pin layout I used for the light pen was:

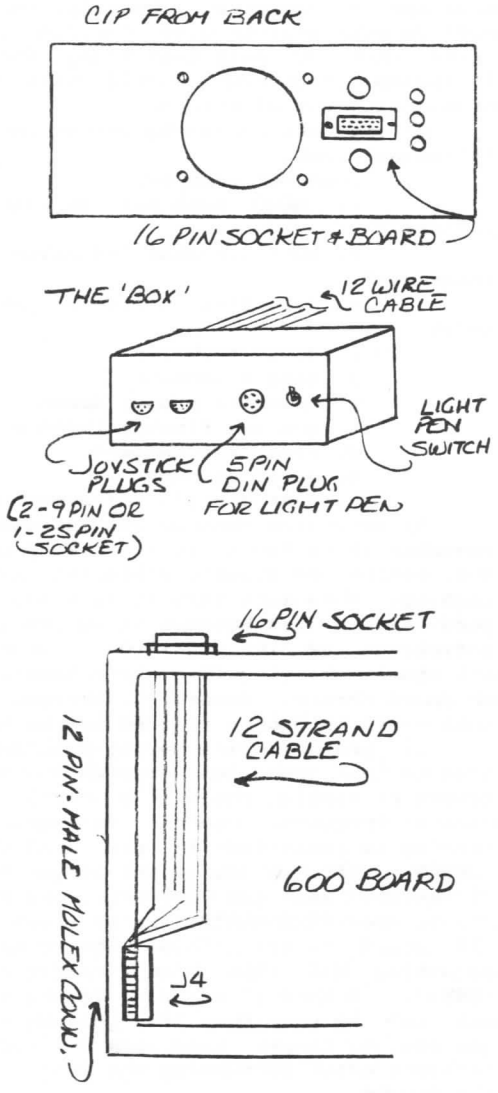
C1	1	16	+5v
C2	2	15	
C3	3	14	
C4	4	13	
C5	5	12	
C6	6	11	R7
C7	7	10	R6
GND	8	9	R1

4) Drill and cut all mounting holes for the boards and sockets. The case must be notched so that the jumper cable will fit through the slot when the top of the case is on.
5) Mount the board(s) to the case. Mount the sockets for the joysticks and/or light pen to the case.
6) Take the ribbon cable and cut to down to 12 wires. Cut it to the approprate length (about 17 inches). Strip and solder the wires to the MOLEX male connector as follows:

1) R1 7) C5
2) R7 8) C6
3) C1 9) C7
4) C2 10) R6
5) C3 11) GROUND
6) C4 12) +5

**NOTE: I get +5 V by soldering a wire to R67, C58 and tying it to pin 12 of the noise port.

Cut a peice of project board the side of a 16-pin socket and mount the other socket to it. Solder the other end of the cable to the 16 pin socket using the same pin layout as for the previous socket. Connect the cable to J4. The 16 pin socket can exit the computer case through one of the holes on the left side. Mount the socket board to the case between these two holes.
7) Plug in the jumper cable to each of the 16 pin sockets. Watch the pin numbers or you may smoke some chips (it might be a good idea to clearly mark pin 1 on both the cable and connectors so it will be easier to set up). Run the test programs for the joystick and/or light pen.



E.Z. LISTER
by Kerry Lourash

If you have 58 free bytes in some nook or cranny of your C1P or C2P BASIC in ROM computer you can use E.Z. LISTER. E.Z. lets you look thru a BASIC program without typing in a lot of LIST commands.

Suppose you have a long program in memory. Set the E.Z. to list eight lines at a time by POKE564,8. Type LIST and hit RETURN. The first eight lines of the program will be listed and you can list eight more lines each time you hit the space bar. If you see a line you want to change, hit any other key and you'll be in the immediate mode automatically. You can start listing at any line number or list any range of line numbers.

The LIST command is not a subroutine; when done, it jumps to the immediate mode. This fact makes it rather hard to control. The E.Z. acts as an on-off valve for the LIST command by patching in to the output routine, which sends characters to the screen and tape. As the characters are output, E.Z. checks the first byte of the input buffer (location $13) for the LIST token. This indicates a LIST command is being executed. When it finds the token, E.Z. increments it and uses it as a flag to show whether the nine counter (location $14) should be initialized.

Now the character to be output is examined. If it is a carriage return ($0D) the line counter is decremented. When the specified number of lines have been displayed, the contents of the line counter are zero. E.Z. jumps to the $FD00 subroutine for an input from the keyboard and returns with the input of register A. If the input is a space, E.Z. restores the LIST token, which causes X additional lines to be displayed. If any other key is input, the stack is reset and E.Z. jumps to the immediate mode.

Set the number of lines displayed by POKE564,(# of lines). E.Z. is patched into the output routine by POKE538,34:POKE539,2. The program is relocatable but the POKEs will have to be changed. For a C2P, the JMP $FF69 must be changes to JMP $FF67. If a warm start is done, E.Z. will have to be patched into the output routine again. If you want to specify the number of screen lines instead of BASIC lines, change the terminal width (location $15) to the number of characters/line displayed on your screen. POKE15,24 for a C1P and POKE15,64 for a C2P. Before SAVEing, you must POKE15,72 or POKE15,255. Now location $0234 will control the number of screen lines displayed. Be warned that this mode may not display all of the last BASIC line LISTed.

```
100 0222  8D0202      STA $0202     SAVE A, X, Y REGISTERS
110 0225  48          PHA
120 0226  8A          TXA
130 0227  48          PHA
140 0228  98          TYA
150 0229  48          PHA
160 022A  AD0202      LDA $0202     LIST TOKEN?
170 022D  A613        LDX $13
180 022F  E099        CPX #$99
190 0231  D006        BNE PLUS      NO, CHECK FOR LIST +1
200 0233  A008        LDY #8        SET COUNTER
210 0235  8414        STY $14
220 0237  E613        INC $13       'LIST' +1
230 0239  E094   PLUS CPX #$9A      IS OUTPUT CHAR = LIST + 1
240 023B  D011        BNE CONT      NO, TO REGULAR OUTPUT
250 023D  C90D        CMP #$D       IS CHAR CR?
260 023F  D00D        BNE CONT      NO, TO REGULAR OUTPUT
270 0241  C614        DEC $14       DECREMENT COUNTER
280 0243  D009        BNE CONT      BRANCH IF NOT ZERO
290 0245  2000FD      JSR $FD00     GET INPUT
300 0248  C920        CMP #$20      IS IT 'SPACE'
310 024A  D00A        BNE STOP      NO, IMMEDIATE MODE
320 024C  C613        DEC $13       RESET LIST FLAG
330 024E  68     CONT PLA           RESTORE A, X, Y REGISTERS
340 024F  A8          TAY
350 0250  68          PLA
360 0251  AA          TAX
370 0252  68          PLA
380 0253  4C69FF      JMP $FF69     TO REG. OUTPUT ($FF67 FOR C2)
390 0256  A2FE   STOP LDX #$FE      SET STACK
400 0258  9A          TXS
410 0259  4C74A2      JMP $A274     TO IMMED. MODE, CHANGE TO $A27D TO
OMIT 'OK'
```

HOW TO CHANGE DISK BUFFER LOCATIONS
by Tim Walkenhorst

OSI has a good disk operating system, but one thing that has always been a problem is that it wastes one or two tracks whenever disk files are used because the buffers are located from $327E to $427E. Your program is stored starting at $427E when two buffers are used. The disk header begins at $3279 and this is where the DOS starts storing material on the disk. This header is used to set where BASIC starts and the length of the program and the number of tracks needed for the program. To save valuable disk space, we can move the buffers.

For my system, I decided to move the buffers to the highest 4K of memory of the 32K available. I also use the RANDOM ACCESS buffer #6 the most, so I put it at the end because I did not want to trap 2K. To do it, I did the following:

1) Call the track zero copy routine in BASIC - DISK!"CA 0200=13,1 (5 1/4" disk). If you have an EXTENDED MONITOR, call the monitor first by DISK!"EM", then call the track zero copier as above.
2) Select the track zero copy (2) and enter R4200
3) Now enter E to exit to the DOS. If you don't have an extended monitor, enter BA. With an extended monitor, the command is RE EM. 4326 should get 4326 7E. Type FF, hit the LINE FEED key to open the next location. 43AC get 43AC 7E. Type FF. Continue until all locations on the following chart are entered.
5) Using BASIC, POKE 17190,255. Do this for all the remaining locations.
6) To save these changes, repeat step 1.
7) Recall the track zero copy routine, W4200/2200,8 and you are finished.

The address locations are the ones to change when track zero is written out to $4200. If you wish to check the actual locations, Subtract 8192 decimal or $2000 hex. (REM FROM THE EDITOR - This is also one of the easiest solutions to the problem of what to do when you forget to put disk buffers under a program before you write it.)

TABLE OF VALUES FOR CHANGING DISK BUFFER LOCATIONS

FUNCTION INFRO	ADDRESS		32K		24K		CURRENT	
#6 BUFFER								
START LO	17190	$4326	255	$FF	255	$FF	126	$7E
START HI	17191	$4327	119	$77	87	$57	50	$32
END LO	17192	$4328	255	$FF	255	$FF	126	$7E
END HI	17193	$4329	127	$7F	95	$5F	58	$3A
INPUT LO	17324	$43AC	255	$FF	255	$FF	126	$7E
INPUT HI	17325	$43AD	119	$77	87	$57	50	$32
OUTPUT LO	17347	$43C3	255	$FF	255	$FF	126	$7E
OUTPUT HI	17348	$43C4	119	$77	87	$57	50	$32
#7 BUFFER								
START LO	17198	$432E	255	$FF	255	$FF	126	$7E
START HI	17199	$432F	111	$6F	79	$4F	58	$3A
END LO	17200	$4330	255	$FF	255	$FF	126	$7E
END HI	17201	$4331	119	$77	87	$57	66	$42
INPUT LO	17405	$43FD	255	$FF	255	$FF	126	$73
INPUT HI	17406	$43FE	111	$6F	79	$4F	58	$3A
OUTPUT LO	17430	$4416	255	$FF	255	$FF	126	$7E
OUTPUT HI	17431	$4417	111	$6F	79	$4F	58	$3A

A BREAKPOINT UTILITY
by Robert Woodward

I own a SB-II and do a lot of assembly language programming. BASIC is easier, but machine language has always been a lot more fun for me. The hardest part about it, though, is debugging and the SUPERBOARD doesn't provide anything to help, except examination of memory locations. It doesn't furnish any way to list register contents. The following program prints out the contents of all registe when a BRK op-code is encounter in program execution, and returns to the program as if nothing has happened. It has a few good points and a lot of bad ones, but it works. Hopefully, it will speed op programming for somebody and the effort that went into writing it will be well worth it.

The good points:

1.) It's completely relocatable (can be palced anywhere in memory, but you'll need a JMP if you move it and use it with a BRK.) and uses no memory for variable storage.

2) It's transparent to the user program (returns with all registers the same as they were before the BRK was encountered.)

3) Most importantly, it works!

The bad points:

1) It uses 62 bytes of page 1 - your stack!

2) The screen doesn't look very pretty after a few times.

Some words of caution:

1) If your stack pointer is > $C0, you'll wipe out this program unless you put it somewhere else.

2) If something on your system will be interrupting, (IRQ) you must check the Status Register to see if the interrupt was a forced break (BRK).

```
; INTERRUPT ROUTINE REGDMP
; DISLAYS REGISTERS ON BRK ON SB-II, RETURNS TO PROGRAM
;
; ORDER OF REGISTER LIST:       S,Y,X,A,P,PCL,PCH
;
;
01C0            . = 01C0
;
01C0    48      REGDMP: PHA             ; SAVE A
01C1    8A              TXA             ; GET X
01C2    48              PHA             ; SAVE X
01C3    98              TYA             ; GET Y
01C4    48              PHA             ; SAVE Y
01C5    BA              TSX             ; GET CURRENT SP IN X
01C6    8A              TXA             ; ALSO IN A
01C7    18              CLC             ; CLEAR CARRY FOR ADC
01C8    69 06           ADC     #6      ; GET ORIGINAL SP BEFORE INT
01CA    48              PHA             ; SAVE IT
01CB    A9 0A           LDA     #$0A    ; LINE FEED
01CD    20 2D BF        JSR     $BF2D   ; PRINT CHAR IN A
01D0    A0 07           LDY     #7      ; 7 REGS TO LIST
01D2    BD 00 01 GTREG: LDA     $100,X  ; GET A REG FROM LIST, X HAS SP
01D5    85 FC           STA     $FC     ; SAVE FOR CONVERSION ROUTINE
01D7    8A              TXA             ; GET X
01D8    48              PHA             ; SAVE X
01D9    98              TYA             ; GET Y
01DA    48              PHA             ; SAVE Y
01DB    20 AC FE        JSR     $FEAC   ; MONITOR DISPLAY CONVERSION TO ASCII
01DE    AD CC D0        LDA     $D0CC   ; GET HIGH NIBBLE CHARACTER
01E1    20 2D BF        JSR     $BF2D   ; PRINT IT
01E4    AD CD D0        LDA     $D0CD   ; GET LOW NIBBLE CHARACTER
01E7    20 2D BF        JSR     $BF2D   ; PRINT IT
01EA    A9 20           LDA     #$20    ; SPACE
01EC    20 2D BF        JSR     $BF2D   ; PRINT IT
01EF    68              PLA             ; GET Y
01F0    A8              TAY             ; RESTORE Y
01F1    68              PLA             ; GET X
01F2    AA              TAX             ; RESTORE
01F3    E8              INX             ; GET NEXT REGISTER
01F4    88              DEY             ; ANOTHER REG DONE
01F5    D0 DB           BNE     GTREG   ; IF NOT FINISHED, DO ANOTHER REG
01F7    68              PLA             ; POP OLD SP OFF STACK
01F8    68              PLA             ; GET Y
01F9    A8              TAY             ; RESTORE Y
01FA    68              PLA             ; GET X
01FB    AA              TAX             ; RESTORE X
01FC    68              PLA             ; RESTORE A
01FD    40              RTI             ; RETURN, RESTORE PC,PS,SP
;
;
                .END
```

The idea for this routine comes from using many big operating systems that provide a register dump on a trap. Carlson's BASIC IN ROM was very helpful with some memory locations and listing of the monitor ROM. The trick of using the monitor display for ASCII conversion comes from a Bruce Hoyt tape dumpting program I saw. Since then I've seen the same trick in a few other programs.

C1P PARALLEL I/O INTERFACE TO A HYCOM PRINTER

by N. Feliss

(REM. The program described and presented in this article should help any one interfacing a parrallel printer to the C1P.)

I recently contstructed a parrallel interface to run high speed electro-chromic printer by HYCOM (Model DC4004 48 cols., 177 cps, $170; IB-28A interface board, $165). In figs. 1 and 2, I show how a M6821 PIA can been linked to the C1P system at ad E000 to E00B. Basically, the data and address lines were wire-wrapped from the socketed U1 (on the C1 board) to a small vector board. A 74LS138 provided address decoding and 8T28 provided the necessary data line buffering (see fig. 1). However, for easiest implementation and minimum work, a PIA printed circuit board with 8K RAM can be obtained from AARDVARK for $29.95. With either system, the user must make sure that there are two data buffer chips (8T28) on the C1P board. The user must supply these two chips in order to provide data output. After the PIA has been wired in with either system it must be checked for correct operation with the C1P system. The following program intitializes ports A and B of the PIA as outputs:

```
10  X=57352:REM BEGINNING ADDRESS OF PIA (E008)
20 POKEX+1,0:POKEX,255:POKEX+1,4:
   POKEX,0:POKEX+3,0:POKEX+2,255
30POKEX+3,4:POKEX+2,0
```

Before testing the two data output ports of the PIA, it is important to tie the B side of the PIA (PB0 - PB7) high via 10K resistors (the B side is normally open collector output). Also, if the PIA is decoded at a different address, line 10 in the above program must be changed to reflect the new address. To test for correct PIA hook-up, simply type:

POKEX,N (A side output)

POKEX+2,N (B side output)

where N can be any interger between 0 and 255. For example, typing in POKEX,0 will

cause all the bits (PA0 - PA7) on the A side to be latched low. Taking a DVM or logic probe and testing the status of these bits will reveal whether the system is functioning or not. If this part of the circuit is working then one can procede on. If not then the PIA interface is suspect and the data, address, and clock lines, and chip select lines to the PIA must be checked with a logic probe or an oscilloscope.

In order to use the parallel data output for directing data or BASIC program listings to a printer, the ROM BASIC I/O subroutine must be modified. An I/O vector is written into RAM ($021A=69 and $021B=FF) upon power on, cold, and warm starts by the ROM BASIC system. This vector, FF69, directs all BASIC output to the TV screen and/or cassette. By changing this vector to a location in the unused RAM space (0222-22FA) one can implement a different output routine. The cassette flag is at 0205. If a '1' is presen all data is directed to the TV and cassette; if a '0' is in that location, the cassette routine is bypassed.

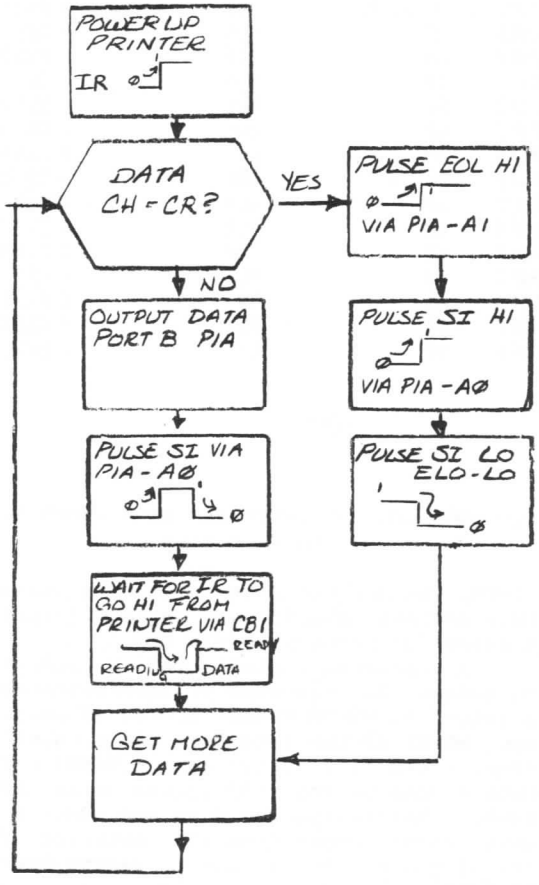
Using this same approach, a small program can be written using a flag to indicate writing to the printer and TV screen. A flow chart describing the correct logic flow is presented in fig. 6. The printer flag is located at 0222, the beginning of the user-Ram space. The flow chart for the printer subroutine is presented in fig. 7. Several handshake lines are used to indicate the "ready state" of both the computer and printer. The computer handshake lines to the printer are designated SI, shift in, and EOL, end of line. The printer handshake line is designated IR, input ready.

Basically the operation of the printer works in this fashion: 1) a character in the A register is directed to the PIA-B side for output to the printer. 2) the SI line is pulsed high-low via PIA-A0 causing the interface card to accept the data character. 3) the IR line is tested for a high level indicating to the computer that the interface card is ready to accept another character. 4) another character is loaded into the A register and tested for carriage return repeated, if it is a CR then EOL is pulsed high-lo. 6) Step one is now repeated.

A software listing of the printer ocntrol program is presented in figs. 8 and 9. This program is written in 6502 machine language starting at user RAM address 0223. A complementary PIA initialization program is presented in fig. 10. This BASIC program sets the PIA for correct handshake operation, initializes ports A and B as outputs, turns on the print flag at 0222, and intitializes a new output routine vector at address 021A and 021B (021A=23 and 021B=02). These two programs can be joined into one BASIC program with the use of DATA statements. An example of this technique is given in fig. 11. This short program can be executed whenever the printer is to be used after power is turned on to the entire system. For a cold or warm start, operation, only statements 80 and 90 in fig. 11 are needed and can bee typed in one at a time and executed. Whenever it is necessary to turn the printer off, the print flag location can be loaded with a zero.

For best operation, the printer should be positioned as close as possible to the C1P computer. The I/O cables from the PIA to the HYCOM printer control board should not exceed two feet. It appears that the printer control board uses CMOS circuitry and as a result, it is very noise susceptible. Therefore, for reliable operation keep the cables short, use braided shielded cable for the handshake lines and remember to ground the shield. Also, flat ribbon cable can be used for the data output lines from the B port to the PIA of the printer control board.

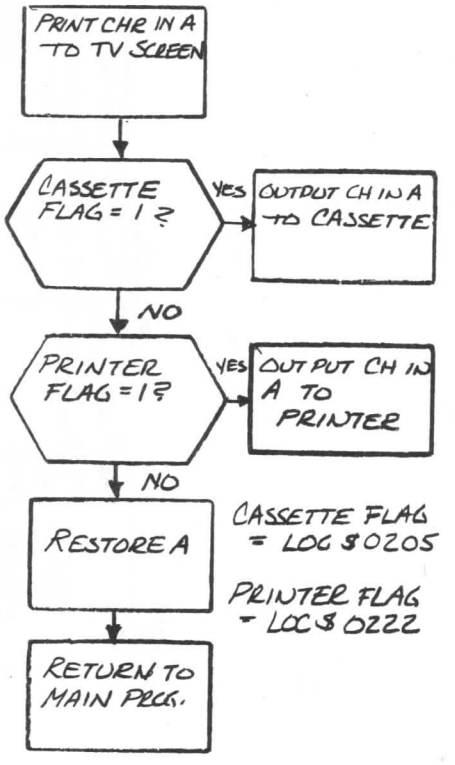
In summary, a PIA can be easily linked to the C1P or SUPERBOARD system for parallel I/O connection to a printer or typewriter system, i.e., a Selectric printer. A small program for transmitting a character from the A register to the PIA output port with handshake logic can be easily written for any system. The operations described in this article can be easily modified and tailored for adaptation to any system requiring parallel I/O data linkage with handshake logic.



```
100 0223    202DBF      JSR A  $BF2D   PRINT CH IN A TO TV
110 0226    48          PHA
120 0227    AD0502      PHA
130 022A    F019        BEQ    L1
140 022C    68          PLA
150 022D    20B1FC      JSR A  $FCB1   OUTPUT CH IN A TO CASSETTE
160 0230    C90D        CMP I  $0D     CH=CR?
170 0232    D046        BNE    L6      NO, BRANCH OUT
180 0234    48          PHA
190 0235    8A          TXA
200 0236    48          PHA
210 0237    A20A        LDX I  $0A     SET CNTR TO 10
220 0239    A900        LDA I  $00     LOAD NULLS IN A
230 023B    20B1FC      JSR A  $FCB1   OUTPUT 10 NULLS TO CASSETTE
240 023E    CA          DEX
250 023F    D0FA        BNE    L3      CONTINUE
260 0241    68          PLA
270 0242    AA          TAX
280 0243    68          PLA
290 0244    60          RTS            FINISHED WITH CASSETTE OUTP
300 0245    AD2202      LDA A  $0222   FLAG FOR PRINTER, 1=PRINT
310 0248    F02F        BEQ    L4
320 024A    68          PLA
330 024B    C90D        CMP I  $0D     CH=CR?
340 024D    D013        BNE    NCR     NO, BRANCH OUT
350 024F    48          PHA            YES, SAVE CR/EOL, SI=HI
360 0250    A902        LDA I  $02     SET EOL HI
370 0252    8D08E0      STA A  PIA     STORE IN A SIDE/OUT TO PRNTR
380 0255    A903        LDA I  $03     SET SI HI NOW
390 0257    8D08E0      STA A  PIA     STORE IN A SIDE/OUT TO PRNTR
400 025A    A900        LDA I  $00     NOW SI AND EOL LOW
410 025C    8D0830      STA A  PIA     STORE IN A SIDE/OUT TO PRNTR
420 025F    4C7902      JMP A  L4
430 0262    8D0A30      STA A  PIA+2   STOR CH IN B SIDE
440 0265    48          PHA
450 0266    AD0AE0      LDA A  PIA+2   CLEAR BIT 7 B SIDE
460 0269    A901        LDA I  $01     BIT 0=1 FOR SHIFT IN
470 026B    8D08E0      STA A  PIA     STORE IN A SIDE/OUT TO PRNTR
480 026E    A900        LDA I  $00     SET SI LOW
490 0270    8D08E0      STA A  PIA     STORE IN A SIDE/OUT TO PRNTR
500 0273    EA          NOP
510 0274    AD0BE0      LDA A  PIA+3   SEE IF PRNTR RDY (0-1) TRANS
520 0277    10FB        >PL    L5
530 0279    68          PLA
540 027A    60          RTS            FINISHED
                    END
```

```
5 REM PRINTER CONTROL PROGRAM
10 X=57352
15 GOSUB1000
20 POKEX+1,0:POKEX,255:POKEX+1,4:POKEX,0:POKEX+3,0
30 POKEX+2,255:POKEX+3,38:POKE546,1:POKE538,35:POKE539,2
100 END
1000 REM THIS SUB POKES PRINTER MC INTO MACHINE LANGUAGE
1001 REM FOR VECTOR PUT 34 INTO 11 AND 2 INTO 12
1002 FORIN=547TO634
1004 READOP:POKEIN,OP
1006 NEXTIN
1010 DATA32,45,191,72,173,5,2
1020 DATA240,25,104,32,177,252,201
1030 DATA13,208,70,72,138,72,162
1040 DATA10,169,0,32,177,252,202
1050 DATA208,250,104,170,104,96,173
1060 DATA32,2,240,47,104,201,13
1070 DATA208,19,72,169,2,141,8
1080 DATA224,169,3,141,8,2224,169
1090 DATA0,141,8,224,76,121,2
1100 DATA141,10,224,72,173,10,224
1110 DATA169,1,141,8,224,169,0
1120 DATA141,8,224,234,173,11,224
1130 DATA16,251,104,96
1150 RETURN
```

OSI / PIA OUTPUT ROUTINE



CIP (UI) TO PIA INTERFACE

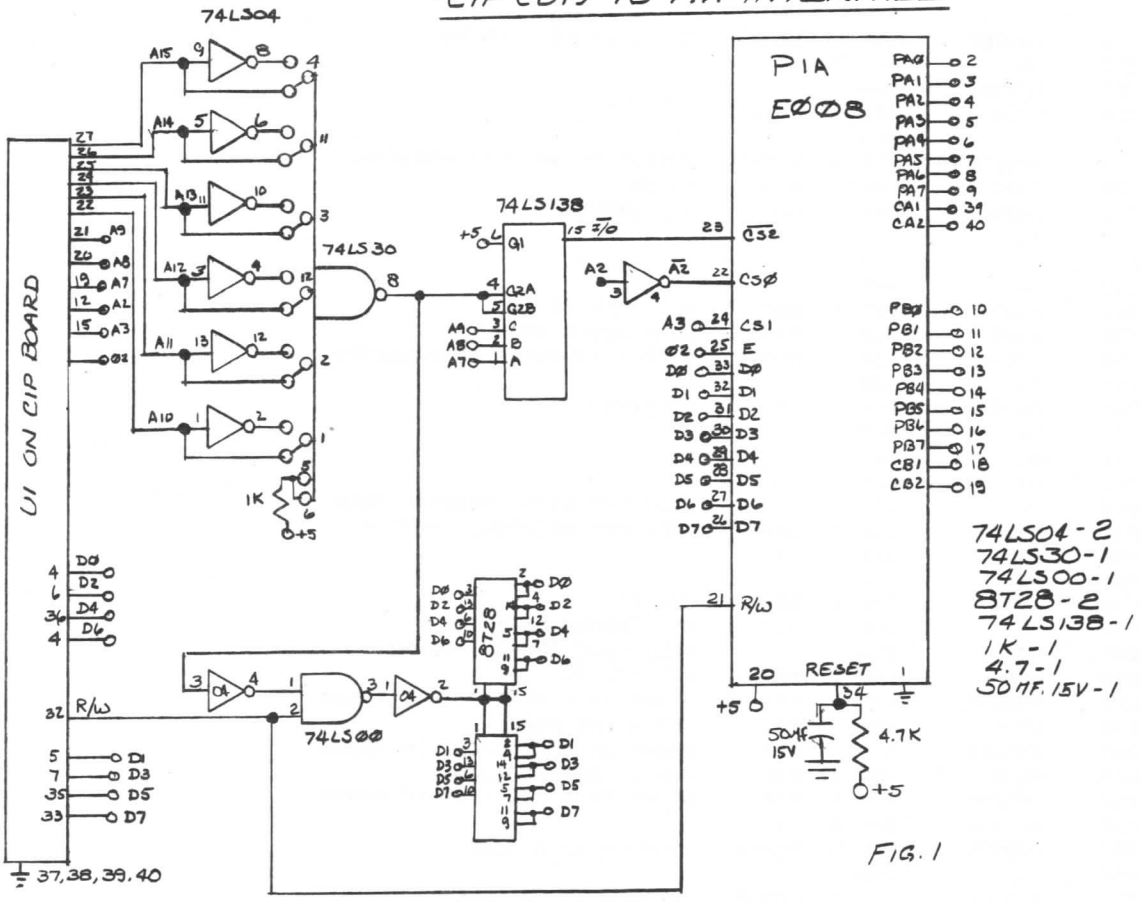


FIG. 1

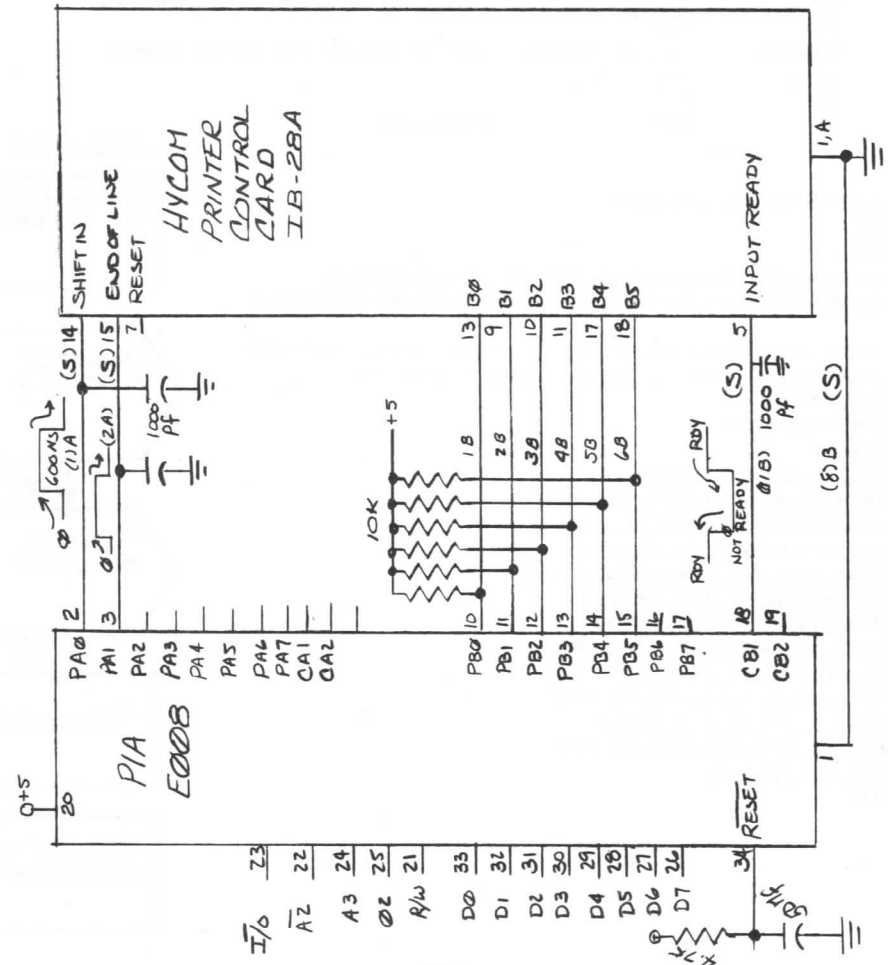


FIG. 2

PICO DOS DISK INITIALIZATION

This program will initalize a blank PICO DOS disk for program storage. It will not put the DOS on track 0 so you will have to boot up with a regular PICO disk.

```
5 INPUT"INPUT PROGRAM#";N
10 NT=3*4*(N-1):FORI=NTTONT+3:?"TRACK "I
20 ST=(I-INT(I/10)*10)+(16*(INT(I/10)))
30 POKE250,ST:REM TRACK IN BCD
40 POKE11,224:POKE12,38:REM TRACK SEARCH
ROUTINE
50 X=USR(X):REM SEEK TRACK
60 POKE11,48:POKE12,39:REM INIT ROUTINE
70 X=USR(X):REM INIT TRACK
80 NEXT
```

NOTE POKE9656,9 WILL LET YOU STORE A PROGRAM IN SPACE #97

This came from Dave Pompea, who also wrote our new GALAXIAN program.

FRED ELLIOT, PORTLAND, OR 97266

Two more lines are needed in the instructions for converting the C2/4 CURSOR to more than 4k systems. The 30s in lines 560 and 700 need to be changed to the value of X given in the table published with the instructions. The additional steps make the CURSOR work great with my 8K system.

WE GET LETTERS

THOMAS OWENS, AUTEC-ANDROS ISLAND, SITE 2, FPO MIAMI 34058

Thought I'd write and enclose the necessary pokes to OS65D required to straighten the video out on my SUPERBOARD with 48 characters per line. You've probably gotten a lot of letters on this so it may help other people.

As an explanation: my screen is $D000 to $D7FF with the cursor at $D748. 48 character width is seen in line 100 as $D748 to $D777 HEX. I have this program in the directory under the name POKER and just select it initially when I boot the disk, then it runs the BEXEC*. It works well and (backspace and all) and only takes a second to run.

```
10 REM POKER
20 POKE9667,215:POKE9677,215:POKE9685,215
30 POKE9750,215:POKE9757,215:POKE9764,215
40 POKE9776,215:POKE9786,215:POKE9801,215
50 POKE9804,215:POKE9811,215:REM-WE JUST SET
ALL 'D3s' TO 'D7s'
60 POKE9800,64:REM SCREEN WIDTH
80 POKE9743,255:POKE9723,63
100 POKE9766,72:POKE9770,72:POKE9670,119
110 POKE9725,7:POKE9730,16:REM BACKSPACE JMP
VALUES
120 RUN"BEXEC*"
```

Another method with disk is to just put the routine in the BEXEC* and when you boot the disk the routine is POKEd into memory.

Thanks for pointing out the intitial area to work in. I'll be glad to help anyone with their video problems.

(REM From editor. This will also work with 64 character wide conversion and, if you leave out the D7 changes, with the model 2 C1MF)

DAVID KILROY, WINDHAM, OH 44288

A little tidbit - to test at what frequency your disk based machi running, type PRINT PEEK(9851).

If less than 50 it's at 1MHZ

If 50 to 100, it's at 2MHZ

If greater than 100, it's at 3.3MHZ

E. MORRIS, MIDLAND, MI 48640

Those OSI backpane connectors which everyone seems to have a hard time finding can be purchased from Technical Products, Box 12983 University Station, Gainsville, FL 32604, at $4.95 for a set of 4 pairs (8 pieces).

GEORGE RANCHOR JR, KLAMATH FALLS, OR 97601

The statement SAVE:PRINTCHR$(15):LIST will create a program tape that will not echo to the screen when loading.

NELSON REYNOLDS, MT CLEMENS, MI 48043

On Pete Kellner's letter on loading tapes without printing to the screen, I found out how it works. Seems that the input routine at $A357 has a little routine to detect the input of a CONTROL O ($07) for either keyboard or tape input. As the input routine also strips the MBS from the incoming character, the stop token ($87) will also be detected as a CONTROL O. When the input routine detects a $07, it sets the control O flag at $0064 which supresses further printing until cleared (in this case, at the end of the LOAD). Now for implementation. The easiest way I have found so far is to write a line like:

```
63000
SAVE:PRINT:PRINT:PRINTCHR$(15):LIST-62000
```

and then RUN 62000 to save the program. Upon loading, all you will see are the carriage returns from the first two PRINTs and the OK prompt at the end of loading. The first two PRINTs clear the initial noise garbage and let you know that there is something going into the machine. (REM from the editor - ya, I know we have run this hint before, but the letters were nice so I printed them anyway.)

LtCOL CURTIS PRESTON, SHALIMAR, FL 32579

Here's a routine for formatting dollars and cents. Use as a C1-P subroutine at 4000. No matter how you enter the dollar amounts in D, the routine will adjust for two places after the decimal and also return a TAB value (P) to columnize at the right side of the screen.

```
4000 B$=STR$(D)+".00"
4010 FORC=1TOLEN(B$): IF
MID$(B$,C,1)<>"."THEN NEXT
4020 L=C+2:B$=LEFT$(B$,L)
4030 IF RIGHT$(B$,1)="." THEN B$ =
LEFT$(B$,L-1)+"0"
4999 P=23-L:RETURN
```

SCOTT HUNTER, Bohemia, NY 11716

To put some zip into your ALIEN INVADERS game add or replace the following lines:

```
285 TU=TU+1:W=6:IFTU>3THEN760
300 NE=0:Q=0: FORY=WTO1STEP-1: FORX=1TO8: AT=TV+X+X+L2*Y:P=PEEK(AT)
310 IFP<32THEN POKEAT,32: POKEAT+DI,C1: Q=Q+INT(Y/W): IFNE=0THENGOSUB90
683 C1=5-C1:IFQ=0THENNW=W-1
687 (delete)
```

As each successive row from the bottom gets wiped out, the program doesn't bother to check it anymore.

BILLY D. SMITH, RICHMOND, KY 40475

I would like to take exception to Scott Klavon's letter in JOURNAL #5. I also was one of the first to buy the modification kit from Progressive. The price of the kit was $39.95, not $70.00 as indicated. Any modification, no matter how complex or insignificant will void your warrenty with OSI and the company made that point clear up front. The completion time was between two and a half to three hours, including contact with Progressive to receive corrections to the modification I did not have. However, I have a stong background and experience doing this sort of work, and as such, feel that considerable more time would be required by the average hobbist. The modification worked as explained in the modification (sic) after corrections and a switch selectable 32/64 character line was available. I agree with you completely that the modification is not for everyone and have personally recommended against installation to all but a few qualified people, for many of the same reasons your stated in JOURNAL #3. However, that 32 X 64 format without guard bands is definately not the partiular modification Progressive sells. I feel that the mod when properly installed and supported by new ROM chips is quite acceptable by any standards. (I would not use any mod of this nature without ROM support.)

(REM "including contact to get corrections"?? - my earlier comment stands.)

DAVID LOCKWOOD, ELK GROVE, CA 95624

I am very glad to hear that you are now an OSI dealer. Our local dealer has dropped the OSI line and left me in the lurch. You see, I, unfortunately, acquired plans for that now infamous video modification from Canada. Not only is my 600 board incredibly chewed up from hundreds of hours of work trying to make the mod functional, but most (if not all) of the chips were zapped in a construction accident. Can you sell me the OSI BASIC ROM chips? If already have the power supply, cabinet, manuals, etc.

(I'm afraid I have nothing but bad news for you. OSI has in the past sold those ROM BASIC chips separately although they are no longer in the catalog. However, when sold separately, they went for $100 a set. I might suggest at this time your best bet would to be to look for an old SUPERBOARD, either left over models from a dealer or a used one. It might be wiser than buying just the chips for $100 (providing OSI still sells them) as it would give you replacement parts for the entire board.

JIM HABOUSTAK, SOUTH EUCLID, OH 44121

When using variable names for integers on disk systems, using a ' end of the variable name saves considerable memory. It is useful for such thngs as bowling scores, screen pixels, etc. A matrix 'A(62,62)' will leave about 220 bytes of free memory. An 'A$(62,62)' will leave about 8150 bytes free. But an 'A%(62,62) will leave about 12,125 bytes free! (This is on my 32K C8P-DF). I don't know how this will work on other machines or on ROM BASIC.

ROBERT WOOD, ST LOUIS, MO 63130

I recently needed a reverse scroll subroutine in order to present some data in a more logical format. The first program in BASIC was too slow, so I coded it in machine language. The subroutine was written for a SUPERBOARD II using BASIC in ROM. It currently scrolls a 32 x 32 screen but may be changed to any screen format desired. It's written to reside in page 2 and to be used as a USR function, but with a little work it may be placed anywhere. If you try different values of the scroll size (now set at 32), you will get some very interesting results.

```
1 REM REVERSE SCROLL DEMO
2 REM BOB WOODWARD
10 FORQ=546TO597:READV:POKEQ,V:NEXT
20 DATA160,32,173,0,212,153,0,212,56
30 DATA173,37,2,233,1,141,37,2,141,40
40 DATA2,173,38,2,233,0,141,38,2,141,41
50 DATA2,201,207,208,223,169,0,141,37,2,141
60 DATA40,2,169,212,141,38,2,141,41,2,96
70 POKE11,34:POKE12,2
80 FORQ=0TO31: POKE53247+Q,4: X=USR(X): POKE53247+Q,32: NEXT
90 FORQ=31TO0STEP-1:POKE53247+Q,32:NEXT
100 GOTO80
 THE ASSMBLER VERSION:
100  0222    A0  20     SCROL:   LDY  #$20
;SCROLL LENGTH
110224   AD  00 D4 LOOP:    LDA  $D400  ;LAST LINE TO SCROLL
120 0227    99 00  D4           STA  $D400,Y
;LAST LINE + SCROLL INCRE
130  022A     38              SEC
;SET CARY FOR SUB
140 022B     AD 25  02          LDA  $0225
;GET LAST POS LOW BYTE
150 022E     E9 01              SBC  #$1
;NEXT POSITION
160 0230   8D 25 02          STA  $0225
170 0233   8D 28 02          STA  $0228
180  0236   AD 26 02          LDA  $0226
;GET HIGH BYTE
190 0239   E9 00              SBC  #$0
;SUBTRACT CARRY FROM LOW SUB
200 023B   8D 26 02          STA  $0226
210 023E   8D 29 02          STA  $0229
220  0241    C9  CF             CMP  #$CF
;ALL DONE?
230 0243    D0  DF              BNE  LOOP
;NOT YET
240  0245     A9 00             LDA  #$0
;RESTORE POINTERS
250 0247   8D 25 02          STA  $0225
260 024A   8D 28 02          STA  $0228
270 024D   A9 D4             LDA  #$D4
280 024F   8D 26 02          STA  $0226
290 0252   8D 29 02          STA  $0229
300 0255   60                RTS
```

CHARLES STEWART, ADRIAN, MI 49221

Enclosed is a little program I have been using instead of purchasing a TI HEX-DEC calculator. Since the TI is $50+ and I already have the computer, it might as well make conversions for me. I have found it quite handy in my work with machne language.

```
 10 REM CHARLES STEWART
 60 REM DEC TO HEX AND HEX TO DEC CONVERSION
 100 DIMA$(16),S$(16):FORX=1TO16:READA$(X):RE
ADS$(X):NEXT
 110 DATA0000,0,0001,1,0010,2,0011,3,0100,4,0
101,5,0110,6
 120 DATA0111,7,1000,8,1001,9,1010,A,1011,B,1
100,C,1101,D,1110,E
 130 DATA1111,F
 135 S$="0123456789ABCDEF"
 140 X=0:Y=0:W=0:Q=0:I=0:E$="0"
 145 FORX=0TO49:PRINT:NEXT
 150 PRINT:PRINT"A> DECIMAL TO HEX ":PRINT
 160 PRINT"B) HEX TO DECIMAL":PRINT:INPUT"YOU
R SELECTION";A$
 170 IFASC(A$)=65THENPRINT"DEC TO HEX CONVERS
ION":GOTO3010
 180 PRINT"HEX TO DEC CONVERSION":GOTO2010
 2010 PRINT:INPUT"HEX NUMBER";I$:IFLEN(I$)>4T
HEN2010
 2020 IFLEN(I$)<>4THENI$=E$+I$:GOTO2020
 2040 FORX=1TO4:FORY=1TO16
 2050 IFMID$(I$,X,1)=MID$(S$,Y,1)THENB$(X)=A$
(Y)
 2060 NEXTY:NEXTX
 2070 B1$=B$(1)+B$(2)+B$(3)+B$(4)
 2080 PRINT:PRINTI$" IN BINARY=":PRINTB1$
 2100 X=1:W=0:Q=LEN(B1$):I=0
 2120 Y$=MID$(B1$,Q,1):Y=VAL(Y$):I=Y*X:W=W+I:
X=X*2
 2130 Q=Q-1:IFQ<>0THEN2120
 2135 PRINT:PRINT"IN DECIMAL "W
 2140 INPUT"READY TO CONTINUE";A$:GOTO2010
 3010 PRINT:INPUT"INPUT DECIMAL NUMBER";I$:I=
VAL(I$):Y$=" ":Y=65536
 3012 Y=Y/2
 3015 IFY>65535THENPRINT"TOO LARGE":GOTO2150
 3030 X=INT(I/Y):IFX=0THENY$=Y$+"0":GOTO3050
 3040 Y$=Y$+"1":I=I-Y
 3050 Y=Y/2:IFINT(Y)=0THEN3200
 3060 GOTO3030
 3200 PRINTI$" IN BINARY = "Y$
 3210 X=2:Y=4
 3215 RE$=" "
 3220 A$=MID$(Y$,X,Y):FORW=1TO16:IFA$=A$(W)TH
ENRE$=RE$+S$(W):GOTO3240
 3230 NEXTW
 3240 X=X+4:IFX>14THEN3260
 3250 GOTO3220
 3260 PRINT:PRINTI$" IN HEX = "RE$
 3270 GOTO150
```

)C.- A BASIC COMPILER FOR OSI -

This one is good news and bad news time. The compiler is available from PEGASUS in Honolulu. At the current time it is available only for 8" disk and 48K is suggested. Pegasus has plans to offer it on 5" disk in the near future, but no tape version is planned or likely to be even possible.

You may have seen the adds for this one. It is advertised as "F-BASIC" and sold mainly on The basis of speed. the authors claim that it is 100 times faster than basic.

Let's cover the good news first. It does compile basic code into native 6502 code and it does run at something like 100 times the speed of BASIC.

Although the language is limited, it does allow you to use just about any legal construct, no matter how inefficient an still write fast - FAST - code. It seems to be pretty much bug free. - I can't guarantee that, however, for reasons that I'll cover later. It also adds a few features that you do not get in regular BASIC. You can, for instance, specify the location in memory for arrays. and you can manipulate the register contents on the 6502 directly. It also adds WHILE and WEND statements to the BASIC. (a WHILE-WEND statement is similar to FOR-NEXT except that a condition is tested rather than a variable incremented in each loop.)

There are some limitations and problems with the program. A list of the reserved words may help to clarify them.

```
 AND      ASC      CHR$     DIM      END      FOR
  GOSUB    GOTO
 IF       INT      NEXT     NOT      OR       PEEK
  POKE     PRINT
 REM          RETURN   THEN     TO       WEND
WHILE
```

That's it - the entire lanquage. Note the lack of INPUT, all string functions except CHR$ and ASC, and trig functions. This is definitely a games writing package - or a package to write USR functions that will be added to a BASIC program. It is a very limited package. I feel that the advertising which lead me to expect something closer to standard MICROSOFT was somewhat misleading. Also not advertised was the fact that this is an integer basic with only positive numbers allowed. The number range is 0-65535 and wraps around If you go overrange. (i.e. 65535+1=0). It is difficult to get used to the differences and we have yet to get a program of any length to run under F-BASIC. You have to get used to doing things that BASIC normally does for you, such as providing a delay between POKEing the keyboard and reading out the keypress. Of course, the difficulty could reflect my skill level more than the utility of the program.

Memory usage may also be a problem. The system may take a lot more space than BASIC would to hold the same program. As the author put it, the compiler was written more for speed than for memory economy. That does sometimes lead to problems. Recently Bob Retelle wrote a missle command program in BASIC in a little under 8K. When compiled, the program (or the portions that worked) ran a lot faster (so fast you couldn't see the missles move.) but the system reported that it used 18431 bytes to assemble it. Not all programs we tried ran 2 1/2 times as long as BASIC, but that was a problem.

Pegasus says that the INPUT and DOS routines are essentially done and simply waiting for final testing before being distributed. They also report that a string handling package is in the works.

Where does it stand now? - as a rather fun but expensive toy. If you can afford $150 for a fun program or if you must have speed at all costs and can get by with the limited BASIC language, it does seems to work - and while we haven't succeeded in getting much working on it yet, it has been fun.

NEW AT AARDVARK

We have been busy again!

As you may have read in the rest of the Journal, we have added a number of machine code utilities. We are paricularly proud of the MACHINE CODE TRACE/EDITOR from Steve Jones. It is one of the best debugging tools that I have ever seen. Steve did an excellent job and by letting us sell it at $24.95, he should be making it available to a lot of people. It is a bargain.

we also added a mini-assembler in BASIC for $9.95. It is limited in the size of program it can do, but It is a two pass assembler with labels and branching. It's a nice item for the beginning programmer. The MACHINE CODE TRACE in BASIC is also a bargain at $12.95.

For disk users, we have a great new copy program. It is in Machine Code and wses less than 2K of memory, making the rest of the memory available for copying. we use it for production here because it is better that dual disk copying . Fantastic Copy (I ran out of names!) $24.95

Best news for the fun loving group is a couple of new games that are as good as the arcade stuff.

GALAXIA - all machine code, available for all video systems. $9.95 on tape, $12.95 on disk. An Invaders like game with fast and aggressive aliens. For all of you who loved (and tired of) INVADERS. This one has the smoothest and fasted graphics I have ever seen on an OSI.

INTERCEPTOR FOR THE C1P 8K TAPE OR DISK $19.95.

You have to protect your cities from Hordes of incoming aliens. You get a couple of automatic cannons to help you, but the action is fast and gets faster with each wave. This is beter than most games you see in arcades. It is fast, smooth, and exciting.

We also added a couple of new data sheets for disk users. MULTI PURPOSE MEMORY SORTING SCHEMES ($4.95), BASIC PROGRAM EDITING USING 6D DISK DATA FILES ($3.00), and HOW TO USE BASIC COMMAND FILES ($1.00). I particularly reccommend the editing and command files data sheets,. they contain disk uses that most people have not thought of.

EPSON PRINTERS -MX-80 I like the new Epson - so much that we dicided to carry them. We'll even run each one on an OSI system before we ship it!. The printer goes for $575 and you need a serial interface that sells for $69

THIS MONTHS' HARDWARE SPECIAL IS THE 8K 610 BOARD for $269.

LAST + AND LEAST - AARDVARK T-SHIRTS IN SMALL, MEDIUM AND LARGE WITH A BIG GREEN AARDVARK AND THE LOGO $5.95

One final note. I have heard from Orion and others that Orion has a whole new catalog including a machine code ASTEROIDS game that is supposed to be great and the new Grafix hi-res video board. Those of you who haven't gotten an Orion catalog recently might want to contact them again.

A NEW INTELLIGENT TERMINAL PROGRAM

I've watched the developement of a new intelligent terminal program for polled keyboard systems being developed at the COMPUTER CONNECTION 38437 Grand River, Farmington Mi 48018. It is a very well done program and I was very surprised when they decided to offer it for sale at $19.95 ($23.95 for 8"). That makes it a real deal. The program offers a lot of options (half and full duplex, even or odd parity, any acia driven, etc.) and offers a neat plus in a routine that allows you to store up to 8 messages for later transmission. (passwords, protocols, names and so on.) It also provides disk buffers for data transmission to and from the main frame.

Those of you looking for terminals should look into this program.

OOOPS AND GOOFS AND RATS!!!

We did it again!. More goofs. The unlabled pot on the video mod schematic last month was a 200 ohm.

There was an extra line in Helicoptor Pilot. Delete line 401 to make it run right.

CLASSIFIED ADVERTISING

C2MF - 24K SINGLE DRIVE MINIFLOPPY. ONE OF THE LAST C2S' BUILT, IT HAS ALL THE FEATURES OF T C4MF (COLOR, SOUND, 2 MG CLOCK) EXCEPT THE A15 BOARD. A REAL WORKHORSE HERE AT AARDVARK $899.00
NOTE - THIS IS A USED MACHINE.
CONTACT OLSEN AT AARDVARK.

SUPERBOARD MODEL 1 8K ANOTHER PIECE OF OUR EQUIPMENT. $225.00
OLSEN AT AARDVARK AGAIN.

****

AT LAST! WE'LL CONVERT YOUR C1P TO RUN, CREATE, AND DISPLAY AND C2/4P SOFTWARE; PERIOD! DONT WORRY, WE'LL DO IT!

ALSO:

CONPLETE RTTY/ASCII/MORSE WITH T/U INSTALLED $170.00

1200 BAUD $30.00

600 BAUD STANDARD ! PLUS NICE DUMP ROUTINE.

WRITE FOR DETAILS : OVES. 1111 CHAPLINE ST. WHEELING WV. 26003

****

ABOUT THE GOBBLER

This months' lead program is really three programs in one. It is also a good example of how a program get grow totally out of control.

Program 1. Artist. I had just returned from Radio Shack and had been impressed with a random picture drawer that they were running. (It was a long, long time ago.). The screen would fill with boxes and a they would go off at random creating a picture. Friend shows up at house and I decide to impress him with how I can write a program (picture maker) while he watches. In twenty minutes we have ARTIST - which I still like to watch. Friend is not repeat not impressed.

program 2. Gobbler. Friend wants to control picture maker. I add controls for up, down, left, and right. - Friend is not impressed.

Program 3. MONGOL HORDES ex-friend said it was boring with total control - that the system should work with probabilties just as real life does. So I set up a system where you influenced rather than totally controlled the Gobbler and called it Mongol Hordes (Mongols were hard to contol too.).

INTELLIGENCE MODE exfriend - now casual aquaintance said the game always stopped too soon as the little Gobbler would eat himself into a corner and stop too soon. So, I added a routine to let the Gobbler remember his last position and back up if he was in a blind alley. Came out with a nice intelligent looking construct.

The stranger at my keyboard still was not impressed. There as a slight bug as the gobbler left behind an image of itself whenever it backed up.

To Heck with him, I like it.

P.S. I'll give a gift certificate to the first person who fixes the bug in the program.

```
5 PRINT"HIT SHIFT TO START"
17 IFPEEK(57088)=1ORPEEK(57088)=254THENR=RND(8):GOTO17
20 FORX=1TO25:PRINT:NEXT:PRINT"PICK A GAME (BY NUMBER)":PRINT
30 PRINT"1 ARTIST":PRINT"2 MONGOL HORDES
40 INPUT"3 GOBBLER";Y:IFY>3THEN20
45 INPUT"GIVE ME A NUMBER FROM 1 TO 10";SR
50 Z=Y+8:IFY=2THENZ=21
60 INPUT"DO YOU WANT INSTRUCTIONS";A$
70 J=Y:IFASC(A$)=89THENONYGOTO510,550,630
80 P1=1:L=64:VB=540:IFPEEK(57088)>127THENVB=600
90 IFVB=600THENL=32:P1=254
100 TIME=1:FORX=1TO23
110 IFVB=600THENPRINT"000000000000000000000000":GOTO130
120 PRINT"000000000000000000000000000000000000000000000000000000000000"
130 NEXT:PO=48:POKE530,1
135 REM FOR DISK POKE530,1=POKE2076,96
140 K=0:W=-L:E=INT(53507+500*RND(9)):IF PEEK(E)<POTHEN140
150 POKEE,88
160 B=E
170 FORX=1TO50:NEXTX
180 Q=RND(1)*Z
190 TIME=TIME+1:IFTIME>40THENPOKEE,32:E=M
200 IFTIME>160THEN490
210 IFTIME>80ANDJ>1THEN490
220 POKEB,32:POKEM,32:POKEE,88
230 IFJ=3THENGOTO320
240 IFQ<1THENE=E+1:GOTO340
250 IFQ<2THENE=E-1:GOTO340
260 IFQ<3THENE=E+L:GOTO340
270 IFQ<4THENE=E-L:GOTO340
280 IFQ<5THENE=E+L+1:GOTO340
290 IFQ<6THENE=E-L+1:GOTO340
300 IFQ<7THENE=E+L+1:GOTO340
310 IFQ<8THENE=E-L-1:GOTO340
320 E=E+W
340 POKE57088,P1:Y=PEEK(57088):IFVB=600THENY=255-Y
350 IFPEEK(57100)=33ORPEEK(57100)=222THEN20
360 IFY=5THENW=-1
370 IFY=3THENW=+1
380 IFY=65THENW=-L
390 IFY=0THENW=L
400 IFY=7THENW=L
420 P=PEEK(E):IFP<>POTHENE=B:IFTIME>40THENPOKEE,32:E=M:GOTO180
430 IFP<>POTHEN180
440 POKEB,32:POKEE,32:M=B
450 B=E:TIME=1:FORX=1TOSR:Q=RND(8):NEXT
460 K=K+1
470 POKEE,88
480 GOTO170
490 FORX=1TO15:PRINT:NEXT:PRINT"YOUR SCORE IS"K
500 GOTO100
510 PRINT"ARTIST":PRINT:PRINT"SIT BACK AND RELAX"
520 PRINT"AND I WILL DRAW PICTURES FOR YOU":FORX=1TO200:NEXT
530 PRINT:PRINT"I THINK YOU WILL LIKE THEM":FORX=1TO300:NEXT
535 PRINT"TYPE (ESC) FOR A NEW GAME
```

```
540 FORX=1TO3000:NEXT:GOTO80
550 PRINT"MONGOL HORDES":PRINT:PRINT"YOU NOW COMMAND A HORRIBLE HORDE
560 PRINT"OF MONGOL X'S"
570 FORX=1TO1500:NEXT
580 PRINT"THERE ARE TWO PROBLEMS":FORX=1TO1500:NEXT:PRINT
590 PRINT"THEY ARE VERY HUNGRY""
600 PRINT"IF THEY RUN OUT OF O'S TO RAVAGE THEY DIE"
610 PRINT"AND THEY DON'T FOLLOW ORDERS
620 PRINT"MUCH MORE THAN HALF THE TIME":GOTO670
630 PRINT"THE GOOB GOBBLER":PRINT
640 PRINT"YOUR FAITHFULL SERVANT GOBBLES GOBS OF GOOBS"
650 PRINT"  (GOOBS ARE TINY CIRCULAR ANIMALS)
660 PRINT:PRINT"YOUR PROBLEM IS TO KEEP HIM FED"
670 PRINT"YOUR CONTROLS ARE"
680 PRINT"(CTRL) TO MOVE UP
690 PRINT"  (L SHIFT) MOVE LEFT      MOVE RIGHT(R.SHIFT)
700 PRINT"          BOTH SHIFTS AT ONCE TO MOVE DOWN"
710 PRINT:PRINT"MOVE FAST, YOUR PETS WILL DIE
720 PRINT"IF LEFT ALONE TOO LONG"
730 PRINT"YOUR SCORE IS THE AMOUNT EATEN
740 PRINT"(ESC) GETS A NEW GAME
750 INPUT"INPUT ANY NUMBER TTO START";T:GOTO80
```

I'M NOT CERTAIN THAT I BELIEVE IN BIORHYTHMS, BUT THIS ONE IS
ONE IS AT LEAST ACCURATE. IT DOESN'T HAVE GRAPHICS
AS IT WAS ORIGINALY WRITTEN FOR THE OLD SUPERBOARD WHICH
HAD THE SAME 24 CHARACTER WIDE DISPLAY AS THE C1P AND
AND I COULDN'T SEE DOING ANY MEANINGFUL GRAPHING ON THAT
DISPLAY SIZE.
BY THE WAY, MOST OF THIS PROGRAM WAS WRITTEN BY JANE OLSEN
(AARDVARK V.P.) AS HER SECOND ATTEMPT AT PROGRAMMING.

```
5 PRINT:PRINT:PRINT:PRINT
10 DIMA$(12)
15 PRINTTAB(11)"BIORHYTHM"
19 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:PRINT
20 INPUT"DO YOU WISH INSTRUCTION";Q$:IFQ$="NO"THENGOTO120
21 FORX=1TO15:PRINT:NEXT
25 PRINT"BIORHYTHM IS BASED ON THE THEORY"
26 PRINT"THAT WE ARE CONTROLLED IN PART"
27 PRINT"BY BIOLOGICAL RHYTHMS THAT BEGIN"
28 PRINT"AT OUR BIRTH":PRINT:PRINT:PRINT:PRINT
30 PRINT"THESE RHYTHMS ARE:"
32 PRINT"PHYSICAL (23 DAY CYCLE)"
34 PRINT"EMOTIONAL (28 DAY CYCLE)"
36 PRINT"INTELLECTUAL (33 DAY CYCLE)"
37 INPUT"READY FOR MORE";X$
40 PRINT"THE FIRST HALF OF EACH CYCLE"
42 PRINT"IS CONSIDERED TO BE UP OR"
44 PRINT"ENERGIZED":PRINT:PRINT
46 PRINT"THE LAST HALF OF EACH CYCLE"
48 PRINT"IS DOWN OR A PERIOD OF "
50 PRINT"REGENERATION":PRINT:PRINT:PRINT
51 PRINT:PRINT:INPUT"READY FOR MORE";X$
52 PRINT"THE FIRST,LAST AND MIDDLE DAY"
54 PRINT"OF EACH CYCLE IS CONSIDERED"
56 PRINT"TO BE CRITICAL - THE PERIOD"
58 PRINT"IN WHICH THE POWER CONTROLLED"
60 PRINT"BY THAT RHYTHM IS AT ITS LOWEST"
62 INPUT"READY FOR MORE";X$
64 PRINT"I WILL TAKE YOUR BIRTHDATE"
66 PRINT"AND  COMPUTE THE TOTAL NUMBER OF "
68 PRINT"DAYS YOU HAVE LIVED (INCLUDING"
70 PRINT"LEAP YEARS) AND THE PRESENT"
72 PRINT"POSITION OF EACH CYCLE"
74 PRINT:PRINT:PRINT:PRINT
76 PRINT"INPUT THE DATE IN THIS MANNER:"
78 PRINT"10,12,1943"
80 FORX=1TO4:PRINT:NEXT:FORX=1TO2000:NEXT
120 DIMF(12):DIMJ(2):DIMW(100)
150 READF(1),F(2),F(3),F(4),F(5),F(6),F(7),F(8),F(9),F(10),F(11),F(12)
```

```
160 DATA31,28,31,30,31,30,31,31,30,31,30,31,
200 PRINT"WHAT IS YOUR NAME":INPUTZ$
220 PRINT"GIVE ME YOUR BIRTHDATE":INPUTM1,D1,Y1
279 PRINT"STARTING DATE OF CHART?":INPUTM2,D2,Y2
290 PRINT"HOW MANY DAYS DO YOU":PRINT"WANT CHARTED?":INPUTL
310 X=M1:GOSUB820:J1=J2+((Y2-Y1)*365):X=M2:GOSUB900:J1=J1-J2
340 Y=Y1:R=0:FORW=Y1TOY2:IFINT(Y/4)=Y/4THENR=R+1
360 Y=Y+1:NEXTW:J1=J1+R
370 A$(1)="JAN":A$(2)="FEB":A$(3)="MAR":A$(4)="APRIL"
380 A$(5)="MAY":A$(6)="JUNE":A$(7)="JULY":A$(8)="AUG"
390 A$(9)="SEPT":A$(10)="OCT":A$(11)="NOV":A$(12)="DEC"
480 PRINT"BIORHYTHM CHART FOR ";Z$
500 FORN=1TOL
507 IFINT(N/10)=N/10THENFORTC=1TO5000:NEXTTC
508 PRINT
510 PRINTA$(M2);D2;Y2
535 P=INT(((J1/23)-INT(J1/23))*23)
540 PRINT"P=";P;:IFP>0ANDP<12THENPRINT"UP  ";
550 IFP=0ORP=12ORP=23THENPRINT"CRIT";
555 IFP>12ANDP<23THENPRINT"DOWN";
560 E=INT(((J1/28)-INT(J1/28))*28)
565 PRINT" E=";E;:IFE<14ANDE>0THENPRINT" UP ";
570 IFE=0ORE=14ORE=28THENPRINT"CRIT";
575 IFE>15ANDE<28THENPRINT"DOWN";
580 I=INT(((J1/33)-INT(J1/33))*33)
582 PRINT" I=";I;
585 IFI>0ANDI<17THENPRINT"UP"
590 IFI=0ORI=17ORI=33THENPRINT"CRIT"
595 IFI>17ANDI<33THENPRINT"DOWN"
600 D2=D2+1:J1=J1+1:IFD2>F(M2)THEND2=1:M2=M2+1
630 IFM2<13THEN640
635 M2=1:Y2=Y2+1
640 IFINT(Y2/4)=Y2/4THEN655
645 F(2)=28:GOTO660
655 F(2)=29
660 NEXTN
670 INPUT"DO YOU WISH TO CONTINUE";B$
680 IFB$="YES"THEN290
685 PRINT"TIME FOR A NEW CUSTOMER":PRINT:PRINT:PRINT:GOTO200
820 J2=0:FORI=XTO12:J2=J2+F(I):NEXTI
860 J2=J2-(D1+1):RETURN
900 REM
920 J2=0
930 FORI=XTO12
940 J2=J2+F(I)
950 NEXTI
960 J2=J2-(D2+1)
970 RETURN
```

RENEWAL TIME!!!

This issue is the last of year #1, volume #1, so if you haven't already resubscribed, its time to do so! The JOURNAL subscriptions only go by even years (full volumes) from April to Februrary. We even provided you with a form (below) carefully placed so you can cut it out without losing one letter of the JOURNAL itself (see how thoughtful we are).

JOURNAL SUBSCRIPTION FORM

NAME _____

ADDRESS _____

CITY _____

STATE _____ ZIP _____

( ) CHECK ( ) VISA CARD# _____

( ) MONEY ORDER ( ) MASTERCHARGE EXP. _____

☐ RENEWAL (VOL#2) APRIL 81 - FEB 82 $9.00

☐ - CAN'T KEEP IT TO MYSELF ~ SEND MY FRIEND A SUBSCRIPTION SO HE'LL QUIT WRINKLING MINE IN THE XEROX MACHINE!

☐ NEW - CHECK APPROPRIATE BOX BELOW

☐ VOL#1 - I WANT ALL THE BACK ISSUES! APRIL 80 - FEB 81 - $9.00

☐ VOL#2 - START MY SUBSCRIPTION WITH APRIL 81 - $9.00

☐ I LIKE TO READ - SEND BOTH VOL#1 & VOL#2 - $18.00

OVERSEAS SUBSCRIPTIONS - $14.00 U.S.

FEEDBACK TIME - OPINION WANTED

You may have noticed the new type style in this issue. We went to it in the hopes of increasing ledgibility and partially because it is a denser medium. We get more characters per page with this method.

The journal was done previously on a Paper Tiger and shot down 20% for publication. This issue was done on the Comprint and reduced about 30%. The listing for the Gobbler was done on the new Epson.

Now, here is the opinion we want. Should we: A> Keep the old style of print. B> Keep the new style. or C> Try again cause it still isn't readable.

NOTICE We now have C1S monitor ROMS that will allow graphics to be printed to screen. If you want to trade your non-graphics version in on a new one, either send the ROM back or send a $25.00 deposit which will be returned when we receive the old ROM back.

A few of the early C1E and C2E ROMs had a typo that made the break point utility nonfunctional. We will replace these on the same basis as above.

As the string bug fix in the C1E and C2E has been less effective than we hoped - a lot less effective - anyone who has purchased a C1 or C2E can get a new ROM BASIC #3 at less than our cost ($8.00).

14 CHARACTER FILE NAMES

A closing goody for 65D users. You too can have 14 character names in your directory. Make these changes

```
 AT.....find..... change to
$2DE1.....$07..... $0F
$2D03.....$06.....$0E
$2DFA.....$F8.....$F0
$2DF4.....$08.....$10
```

REMEMBER - you will have to put together a directory program to read 14 character file names and have a Bexec* program ready to put on the disk. When you have it done, you have room for big names - or - if you want to do a little extra work, for big names and files coded for assembler, BASIC and data. Have fun.

P.S. This was written by Chuck Scott as part of the Super executive program he has put together for AARDVARK.

AARDVARK
TECHNICAL SERVICES
1690 Bolton, Walled Lake, MI 48088