the

AARDVARK JOURNAL

october 1981 vol. 2, no. 4

IN THIS ISSUE

The "Beginners Corner" is back this time. The one in this issue was written by Howard Drake. He goes over the OSI Key Input routine in detail and also provides a simple calculator for use with any system. We've covered the Input-a-Key routine before, but never in this detail. From the questions that we have gotten this past month, it would seem that there are a lot of people interested in it.

We also have yet another update to the Word Processor. This one is a routine that allows easier input of lower case letters.

We've got a fun program that will dump out the BASIC Source Code so that you can see how each line actually looks inside the machine, and a Supertrace program.

We also have a lot of material on Disks. About once a year we try to dedicate a large portion of at least one Journal to Disk users. We have instructions on modifying disk BASIC to use the features of the C1E ROM, some notes on PICO DOS, a couple more disk modifications and improvements and corrections to some prior mods. We also have the story of one man who put on his own disk using a non OSI disk. We also have a couple of long articles on Speeding up Random Access Files on disk.

We also have a lot of programs submitted by Stankewicz and Robinson. S & R are the pair that did the excellent Night Rider, the original Maze, and several other good programs for our Catalog. Now they've contributed a couple interesting programs to the Journal also.

WHAT'S NEW AT AARDVARK

AARDVARK now carries D & N Micro Services boards and Mittendorf boards and kits.

The D & N Micro products line is particularly recommended for the 48-pin buss OSI machines. We have been recommending their boards for so long we finally decided we had to carry them. The line runs all the way from combination memory and disk controller boards that sell bare for $50 up to 24K assembled memory and disk controller boards for $380. They also offer a lot of neat I/O boards, a proto-type board, a backpane and even a neat little board that puts all of those onto a C1P. It's an excellent line, and we're proud to carry it.

Mittendorf offers a number of interesting kits. Their most famous one is the Video Conversion kit for High Res Graphics for the C1P. We'll have a complete review of that in the next AARDVARK Journal. We have one in now, but we just haven't had time to test and to write the article up for this issue.

From the program side, we have a couple new games by that famous Stankewicz and Robinson team that designed the original Minos game. The best new game is Night Rider ($14.95). This is partially machine code high speed version of an old arcade idea. In it you see a road stretching out before you twisting and winding as you race your car. The graphics are excellent and very realistic.

They have also now given us OSI GRAND ($9.95). This charming little dittie was initially misnamed by the authors "DEAD BABIES". It emulates a hotel fire and gives you the job of catching people tossed off the roof and bouncing them into a waiting ambulance. When you miss they splatter beautifully and the only problem with it is that sometimes children tend to miss intentionally.

The same team has come up with a new arcade game called Galactic Debris, a good version of Potato Chip Invasion. However, the game goes on for many many variations and levels of difficulty and takes several days to master completely. It is an excellent arcade piece.

For disk users, we have an all machine code full screen editor for the C1P MF. This one was done by Dave Pompea who has written frequently for this Journal. It is available on the new Superdisk for the C1P for $24.95. It is also available to people who have previously purchased the old Superdisk for $9.95. It certainly does make the C1P a much more usable machine.

OSI KEY INPUT & OTHER IDEAS

BY HOWARD G. DRAKE

NOTE (We have been over this before in various places, but this is the best explantion yet of the valuable trick.)

"Thats so simple: Why didn't OSI tell us"

OSI's polled keyboard is a very powerful tool. However, when I bought my C2-4P over two years ago, there were no instructions on how to use this keyboard. I had had my 2P for two months before I heard about the OSI Graphics Manual. This manual shows how to scan the keyboard from a BASIC program by POKEing and PEEKing values at memory location 57088. (If this doesn't sound familiar, immediately call your OSI dealer and ask for the graphics manual.) This method using location 57088 is okay if only a few different values must be inputted and real time operation is required. However, if real time operation is not required there is a simpler method that I prefer. This method makes use of a routine in ROM. The code required is:

```
100 POKE11,0:POKE12,253: X=USR(X):
A=PEEK(531)
```

To see how this works, enter and run this simple program.

```
100 PRINT"PLEASE HIT ANY KEYS"
110 PRINT"0-9,A-Z,OR PUNCTUATION MARKS"
120 POKE11,0:POKE12,253
130 X=USR(X):A=PEEK(531)REMPEEK(9834)FOR
C4PMF,(9815)FOR C1PMF
140 PRINTA,CHR$(A)
150 GOTO130
160  REM FOR  DISK ELIMINATE LINE NO.120
AND REPLACE USR(X) WITH  DISK!"GO
252B"IN LINE130
```

This method is not new and has appeared in several of the small OSI related newsletters. (The place I first saw it was in the AARDVARK Catalog.) However, in this article I'll pursue the ideal further and show an application.

HOW IT WORKS

In the monitor ROM is a routine that scans the keyboard and stores the ASCII value of the key pushed at memory location 531 decimal. By using the USR function in BASIC, we can jump to this already written routine in ROM. This routine scans the keyboard until a key is pushed. There is no time limit as to how long this routine will sit and wait for a key to be pushed. (Thus this method is not suited to real time action games.) When a key is pushed the ASCII value of the symbol on the key top is computed and this value is stored at location 531. Control then returns back to the BASIC interpreter and our program. All we need to do then is to see what value is stored at location 531 by using the PEEK function.

The USR function transfers control from a BASIC program to a machine language subroutine whose starting address is stored in memory locations 11 and 12. Thus before calling the USR function in line 130, we must store the starting address of the keyboard scan routine in memory locations 11 and 12. This is done in line 120.

Try running the program above and hit the return key, then try the line feed key. Change line 140 to: 140 PRINT A :and run again and try the return and line feed keys. Now change line 140 back to: 140 PRINT A,CHR$(A) : and type RUN. Release the shift lock key and then push various keys. Table 1 shows the value stored in location 531 for all key depressions with various combinations of the shift keys. All these possible values mean a lot of information can be inputted with a single keystrokes.

A SIMPLE CALCULATOR

The program in listing 1 simulates a simple caluclator and makes use of the key detection method described above. This calculator has normal algebraic heirarchy i.e. it completes any pending multiplication or division operation before completing addition or subtraction. This program scans the keyboard, branches to the appropriate subroutine when a key is pushed, performs an operation and displays the results.

TO RUN

1. TYPE PROGRAM IN, AS ANY STANDARD BASIC PROGRAM
2. TYPE RUN
3. LEAVE THE SHIFT LOCK KEY DOWN AND DO NOT
   PUSH THE SHIFT KEYS WHILE USING
4. TO ENTER 0-9 AND DECIMAL POINT (.)
   PUSH THAT KEY
5. FOR + PUSH THE +; KEY
   FOR - (MINUS) PUSH THE =-KEY
   FOR X PUSH THE *: KEY
   FOR / PUSH THE ?/ KEY
   FOR = PUSH THE <, KEY
   PUSH RUB OUT ONCE TO CLEAR ENTRY
   PUSH RUB OUT TWICE TO CLEAR ALL
6. PUSH CTRL C TO EXIT PROGRAM

In addition to the key detection routine, this program has a couple of other tricks that are useful. The first trick is that several different machine language routines may be called from a BASIC program with USR. To do this simply POKE the appropriate starting addrress in locations 11 and 12 before calling USR. The second trick is storing a machine language routine in memory locations 546 to 768. This area is not used by BASIC.

SIMPLE CALCULATOR #1

```
10 REM..SIMPLE CALCULATOR REV 02
11 REM..HOWARD G. DRAKE
12 GOSUB 100
13 POKE56832,0:REM..DELETE FOR C1P
14 L=1:D=54026:Y=1:Z=0:W=0:F=1:G=1:R=0:E=0
16 POKED+5,48:IS$=""
18 POKE11,0:POKE12,253:U=USR(U):A=PEEK(531)
20  IFA=127 THEN  GOSUB  170:R=R+1:ON  R GOTO
16,14
22 R=0:B=A-43:IF B<1 GOTO 18
26  ON  B  GOSUB  80,50,30,70,30,30,30,
30,30,30,30,30,30,30,60,40
28 GOTO 18
30  IF  E=1 THEN IS$="":E=0:REM..BUILD NUMBER
STRING
32 IF IS$="" THEN GOSUB 170:L=1
34 IS$=IS$+CHR$(A):POKE D+L,A:L=L+1:RETURN
40  W=VAL(IS$):Z=Z+F*Y*W^G:G=1:F=1:Y=1:
REM..ADDITION ROUTINE
42 IS$="":Z$=STR$(Z):GOSUB 170
44  FOR  L=1  TO LEN(Z$):POKE
D+L,ASC(MID$(Z$,L,1)):NEXT L
46 RETURN
50 W=VAL(IS$):Z=Z+F*Y*W^G:G=1:F=-1:Y=1:
REM...SUBTRACTION ROUTINE
52 IS$="":Z$=STR$(Z):GOSUB 170
54 FOR  L=1 TO  LEN(Z$):POKE
D+L,ASC(MID$(Z$,L,1)):NEXT L
56 RETURN
60 W=VAL(IS$):Y=Y*W^G:G=1:REM..MULTIPLICATION
ROUTINE
62 IS$="":Z$=STR$(Y):GOSUB 170
64  FOR L=1 TO  LEN(Z$):POKE
D+L,ASC(MID$(Z$,L,1)):NEXT L
66 RETURN
70 W=VAL(IS$):Y=Y*W^G:G=-1:REM..DIVISION
ROUTINE
72 IS$="":Z$=STR$(Y):GOSUB 170
74 FOR L=1 TO LEN(Z$):POKE
D+L,ASC(MID$(Z$,L,1)):NEXT L
76 RETURN
80
W=VAL(IS$):W=Z+F*Y*W^G:G=1:F=1:Y=1:Z=0:REM..=
ROUTINE
82 IS$=STR$(W):E=1:GOSUB 170
84 FOR  L=1 TO LEN(IS$):POKED+L,
ASC(MID$(IS$,L,1)):NEXT L
86 RETURN
100 RESTORE:REM..MACHINE  LANGUAGE SCREEN
CLEAR ROUTINE
110 FOR L=744 TO 767:READ M:POKE L,M:NEXT L
120  DATA
169,32,160,8,162,0,157,0,208,232,208,250,238
130  DATA
240,2,136,208,244,169,208,141,240,2,96
170 POKE 11,232:POKE 12,2:U=USR(U):RETURN
200 REM..VARIABLE TABLE
210 REM..L-SCREEN LOCATION MARKER
220 REM..D-SCREEN LOCATION
230  REM..Y-FIRST  OPERAND FOR MULTIPLICATION
AND DIVISION
240 REM..Z-FIRST  OPERAND  FOR  ADDITION AND
SUBTRACTION
250 REM..W-VALUE INPUTTED
260 REM..IS$-STRING INPUTTED BY OPERATOR
270 REM..F-FLAG FOR ADDITION OR SUBTRACTION
280  REM..G-FLAG  FOR  MULTIPLICATION  OR
DIVISION
290 REM..R-FLAG FOR "RUB OUT"
300 REM..E-FLAG FOR =
```

E.STODDARD, GRAYSLAKE, ILLINOIS

How do I get my 65D-V3 to print out the zero in the second decimal place when I'm using it to generate price sheets? An example is:

```
LIST PRICE IS $41.35  DISCOUNT IS 40%
COST IS       $25.10
Printer prints $25.1
```

Dear Mr. Stoddard:

The solution to your problem is to use the STR$(X). It works like this. If you assume that (A) is a number that you want set into dollars and cents, then use lines like these:

```
100 A$=STR$(A)
110  IFINT(A/100)=A/100THENA$=A$+".00"
GOTO150
120 IFINT(A/10)=A/10THENA$=A$+"0"
150 PRINTA$
```

That should add the 0's right.

BERND PENNEMANN, GERMANY

I have found some mistakes in the AARDVARK Journal, Vol.1, No.4, page 15. The listing includes "GOTO230" in the lines 580 and 660 which should be replaced by "GOTO240".

Thanks for the adventure game article, but the listing seems to be worthless without a table of the commands ("GO...") and an explanation about what can happen to you and what must be done ("find a bomb..").

Most of us know the Input-without-scroll from the AARDVARK catalog which replaces the GET-command used in other BASICs. If you want to use this replacement together with your home-brewed or bought editor, use:

```
"X=PEEK(536):POKE11,X:X=PEEK(537):
POKE12,X:X=USR(X)"
```

Something like "POKE11,PEEK(536)" does not work, because both, PEEK and POKE, use the Vector $11,12. Therefore, the location 11 (dec.) will be unchanged.

Some BASICs have a very good feature which can be used in Word Processing much more satisfactory than the INPUT-statement. It is called LINE INPUT. With this feature used, you can input a whole line including quotation marks and commas to a string variable. Of course you can enter only one string per line. The following routine simulates this command and will not go to the direct mode of BASIC if you press RETURN after the "?" appears; instead, the string will be empty.

```
10  POKE11,89:POKE12,163:PRINT"?";:
X=USR(X):X=19
20  IFPEEK(X)<>0THENA$=A$+CHR$(PEEK(X)):
X=X+1:GOTO20
```

This routine will branch to the input-a-line routine located in ROM at A357. Then it will search through the input-buffer, starting at $0013 and adding up all characters in A$ until a $00, which is the end-off-line indicator, is found.

Dear Mr. Pennemann:

One of the pleasures in playing Adventures is to discover what words work. Same goes for the purpose of the game. Discovering wht you have to do is part of the pleasure of exploration. Some other authors of OSI Adventures do include a listing of keywords but their Adventures can often be solved in 3 to 5 hours. I think that spoils the fun.

CHARLES HEPNER, STERLING HTS., MICHIGAN
(This program is a lot of fun to play with - and may be even be instructive!)

For sometime now I have been dumping memory in various formats. One of which is in BASICline orientation. The two listings are for BASIC MEMORY DUMP and an Output from the Execution of the program. The program prints the line numbers of the BASIC lines in both decimal and Hex. This program illustrates how basic lines with their pointers are stored in memory.

```
 10 REM MOD OF HEX MEM DUMP
 15 REM BASIC LINE DUMP
 20 INPUT"BEG LINE#";B
 40 INPUT"END LINE #";E
 50 A=PEEK(121)*256+PEEK(120):REM BASIC IN ROM A=769
 60 H$="0123456789ABCDEF"
 70 LN=PEEK(A+3)*256+PEEK(A+2):IFLN>ETHENPRINT"NEXT LINE=";LN:END
 80 PTR=PEEK(A+1)*256+PEEK(A):IFPTR=0THENEND
 90 IFLN<BTHENA=PTR:GOTO00070
 100 PRINTLN;TAB(7);:D=A:L=4:GOSUB00170:PRINT" ";:L=2
 110 N=PTR-A:REM # BYTES TO PROCESS FOR CURRENT LINE
 120 FORZ=1TON:D=PEEK(A):A=A+1:PRINTTAB(7):GOSUB00170
 130 IFZ=2ORZ=4THENPRINT" ";
 140 IFZ*2=58THENPRINT
 150 NEXTZ:A=A-N:REM?" ;
 160 PRINT:A=PTR:GOTO00070
 170 C$="":X=1
 180 Q=INT(D/16):H=D-(Q*16):D=Q:X=X+1
 190 C$=MID$(H$,H+1,1)+C$
 200 IFQ>0ORX<=LTHEN00180
 210 PRINTC$;
 220 RETURN
```

SAMPLE RUN

```
BEG LINE#? 10
END LINE #? 220
 10    317F 9931 0A00 8E204D4F44204F46204845582O4D454D2044554D5000
 15    3199 AF31 0F00 8E204241534943204C494E452044554D5000
 20    31AF C231 1400 842242454720C494E4523223B4200
 40    31C2 D631 2800 8422454E44204C494E452023223B4500
 50    31D6 0332 3200 41ABBB2831323129A5323536A3BB2831323O293A8E20424153
       494320494E20524F4D20413D37363900
 60    3203 1D32 3C00 4824AB2230313233343536373839414243444454622OO
 70    321D 4F32 4600 4C4EABBB2841A33329A5323536A3BB2841A332293A8A4C4EAA
       45A097224E455854204C494E453D223B4C4E3A8000
 80    324F 7032 5000 505452ABBB2841A33129A5323536A3BB2841293A8A505452AB
       30A08000
 90    3270 8732 5A00 8A4C4EAC42A041AB5054523A88303030373000
 100   3287 AD32 6400 974C4E3B9C37293B3A44AB413A4CAB343A8C303031373O3A97
       2220223B3A4CAB3200
 110   32AD DF32 6E00 4EAB505452A4413A8E2023204259544553205 44F2050524F43
       45535320464F52204355525245 4E54204C494E4500
 120   32DF 0333 7800 815AAB319D4E3A44ABBB2841293A41AB41A3313A979C37293A
       8C303031373000
 130   3303 1633 8200 8A5AAB32A95AAB34A0972220223B00
 140   3316 2433 8C00 8A5AA532AB3538A09700
 150   3324 3733 9600 825A3A41AB41A44E3A8E3F22203B00
 160   3337 4A33 A000 973A41AB5054523A88303030373000
 170   334A 5833 AA00 4324AB22223A58AB3100
 180   3358 7B33 B400 51ABAE2844A63136293A48AB44A42851A5313629 3A44AB513A
       58AB58A33100
 190   337B 9133 BE00 4324ABC32848242C48A3312C3129A3432400
 200   3391 A533 C800 8A51AA30A958ACAB4CA0303031383000
 210   33A5 AE33 D200 9743243B00
 220   33AE B433 DC00 8D00
NEXT LINE= 17021

OK
```

CHANGING THE 'OK' PROMPT
BY GREG WILDER

Are you BASIC IN ROM users tired of seeing the famous 'OK' prompt after all immediate mode commands. If so, run this program first. (It will erase and leave a machine code program behind.)

The routine will accept any word, name or punctuation mark as a prompt. Now you can have 'READY', 'C1P ON-LINE', or even an '*' as a prompt. The routine resides in the highest available RAM and uses 25 bytes of memory plus the length of the prompt.

ALTER PROMPT ROUTINE

```
10 INPUT"ENTER PROMPT";PR$:L=LEN(PR$)
20 POKE3,76:POKE4,240:POKE5,28
30 FORX=7408TO7423:READA:POKEX,A:NEXT
35
POKE134,(PEEK(134)-1):POKE133,256-25-L
40 BA=7402-L:FORY=1TOLEN(PR$):POKEBA+Y
   ASC(MID$(PR$,Y,1)):NEXT
50
FORX=BATOBA-3STEP-1:READA:POKEX,A:NEXT
55 POKE7403,13:POKE7404,10:POKE7405,13:
   POKE7406,10:POKE7407,0
60 DATA162,0,189,224,28,240,6,32
70 DATA238,255,232,208,245,76,201,168
75 DATA10,13,10,13
80 POKE7411,231-L:NEW
```

MARK GUZDIAL, ROYAL OAK, MICHIGAN
Congratulations on your Compiler. I think it's great news, but what is the minimum system configuration? In other words, I want one badly, but will it run on my 8K cassette-based C1P?
First, in Bob Retelle's article on memory saving in the Oct. 80' issue (I know thats a long time ago) he mentions 207 bytes he found for USR use from $0130 to $01FF. Because the 6502 stack pointer is one byte long with a ninth bit added which is permanently one, all of page one is 6502 stack space. Since the stack builds down, and Basic initializes it to $01FE, lower addresses in the $01XX section of memory may be safe, but you always run the risk of the 6502 overwriting your memory if subroutines go too deep, or Basic uses RPN on an extensive computation, or any other extensive use of the stack.

Secondly, you might mention that the programs presented by Charles A. Stewart in the last issue of the Journal are very efficient memory relocators, they are not program relocators. Nowhere does he try to reconcile jumps, jump to subroutine, or branches in the memory moving, so the programs he moves (assuming machine language programs) will probably not run at the new address unless they've been designed for relocatibility.

Dear Mr. Guzdial:
Yes. The Compiler will run in 8K, but it takes 5.7K leaving only 2+K for both source and object code-about enough for a USR(X) routine.

MODIFICATION FOR LETTER WRITER
BY STANLEY HARSHFIELD

This modification enables the use of the upper/lower case OSI character generator, without messing around with the Shift Lock key. In the upper/lower mode, it converts the keyboard to a standard typewriter keyboard, using either the Right or Left shift keys for caps. For capitals O and N, use the left shift only, as one would normally. When the Right shift is used with these two keys, the backspace and up arrow functions are enabled.

There are two versions. Unfortunately, due to the special DOS software, its version is somewhat different.

Incidentally, note the corrected PEEK values in both versions from those given in BATTLEFLEET.

Alternately, Lines 90-105 could be accessed as a subroutine, located at the beginning of the program, to allow all string inputs throughout to be upper/lower case.

UPPER/LOWER KEYBOARD ENTRY

```
64  REM C1P ROM VERSION
65  POKE11,0:POKE12,253
66   PRINT"IS  THIS TO BE:":PRINT:PRINT"
1>UPPER CASE ONLY"
67    PRINT"  2>UPPER/LOWER
CASE":PRINT:PRINT:INPUTTO
90  A$="":PRINT"COMMAND:":ONTOGOTO103
91  X=USR(X):P=PEEK(531)
92  IFP=13THEN102
93  Q=PEEK(57088):IFQ=252ANDP=94THEN101
94  IFQ=252ANDP=95THEN98
95  IFQ=250ORQ=252THEN99
96  IFP>64ANDP<91THENP=P+32
97  GOTO100
98  PRINTCHR$(95);:A$=MID$(A$,1,
LEN(A$)-1):GOTO91
99  IFP>80ANDP<107THENP=P-16
100 IFP=64THENP=80
101 PRINTCHR$(P);:A$=A$+CHR$(P):GOTO91
102 PRINT:GOTO105
103 INPUTA$
104 IFLEN(A$)<1THENA$(L)=A$:L=L+1:
GOTO90
105 IFASC(A$)=94THEN120

64  REM C1P-MF VERSION
65  POKE8955,43:POKE8956,37
66   PRINT"IS THIS  TO BE:":PRINT:PRINT"
1>UPPER CASE ONLY"
67   PRINT"  2>UPPER/LOWER
CASE":PRINT:INPUTTO
90   A$="":PRINT:PRINT"COMMAND:":
ONTOGOTO103
91  X=USR(X):P=PEEK(9504)
92  IFP=13THEN102
93  IFP=76THEN97
94  Q=PEEK(57088):IFQ=252ANDP=94THEN101
95  IFQ=250ORQ=252ORQ=255ORQ=218THEN99
96  IFQ=220ANDP=95THENPRINTCHR$(8);:
A$=MID$(A$,1,LEN(A$)-1):GOTO91
97  IFP>64ANDP<91THENP=P+32
98  GOTO100
99  IFP>80ANDP<107THENP=P-16
100 IFP=64THENP=80
101 PRINTCHR$(P);:A$=A$+CHR$(P):GOTO91
102 PRINT:GOTO105
103 INPUTA$
104 IFLEN(A$)<1THENA$(L)=A$:L=L+1:
GOTO90
105 IFASC(A$)=94THEN120
```

REVISION FOR SINGLE STEP TRACE PROGRAM

SINGLE STEP TRACE AND LISTER PROGRAM
BY TODD BAILEY

The BASIC TRACE program in the C1E Monitor ROM handbook gave me an idea for a Single Step Trace and Lister program. This program intercepts the call to the "CTRL C" check as in the handbook's program. But I have added a call to the get key subroutine so that the program will wait till a key is pressed before executing the next instruction. A flag was added at dec.220 for no trace, full time trace, or Single Step Trace. The program also checks for which key was pressed in the Single Step mode. A "CTRL A" will execute one step with no line number display. A "CTRL S" will execute a step and display its line number (CTRL and just about any other key pressed will do this also). The "CTRL C" works as before. By pressing and holding "CTRL A" or "CTRL S" you can step through your program as long as you hold the keys down. Releasing the keys will stop program execution.

The "CTRL A" function in the Single Step mode can also be used to control listing. Type in "LIST" and hold down the press it again to continue your listing.

The program itself can be relocated anywhere in memory. However, setting up the vector to the program can be a job! To set the vectors with POKES, you must either position the program so that you can change the vector with a single POKE, or place a RTS at the location to which the vectors will be pointing between your two POKEs. Otherwise your first POKE statement will be executed, the computer will jump to wherever the vector then points for a "CTRL C" check and will jump to garbage and never get to your second POKE statement. Another way around this would be to use a USR call to jump to a short machine code program to set up both vectors at once.

Listing one and two are the Assembler listing of the program for the OSI ROM and C1E ROM. Listing three and four are Basic POKE programs. Again, one for OSI's ROM and one for the C1E.

Another neat thing that I have found to use the C1E ROM for is in run running BASIC with the Assembler/Editor. You can load in the Assembler and use the block move command to move it up to the top of memory. When you cold start in Basic set the memory size below where you stashed the Assembler. Then, if you want to use the Assembler later, use the Block move again to put it back where it belongs and go from there. One of these days I am going to relocate the Assembler so that I don't have to play these games!!

I changed the jump to print the line number from $B953 to $B95A to bypass the printing of "IN LINE" before each line number. It looks nicer this way.

LISTING #1

```
10 0000              ;SUPER TRACE FOR OSI
MONITOR
20 0000            ;CHECK TRACE FLAG
30 0000 A5DC         LDA $DC  ;GET FLAG
40 0002  C901        CMP #$01 ;COMPARE TO
ONE
50 0004 F00B         BEQ PRLN ;FULL TRACE
60 0006 900C         BCC END  ;NO TRACE
70 0008            ;TRACE ROUTINE
80 0008 2000FD       JSR $FD00 ;GET KEY
90  000B C903          CMP #$03 ;IS IT A C
?
100 000D F005        BEQ END  ;NORMAL CTRL
C
110 000F 9003         BCC END  ;IS IT AN A
?
120 0011 205AB9 PRLN JSR $B95A  ;DISPLAY
LINE NUMBER
130 0014 4C9BFF END  JMP $FF9B  ;EXIT TO
CTRL C
```

LISTING #2

```
 10 0000          ;SUPER TRACE FOR C1E
 20 0000          ;CHECK TRACE FLAG
 30 0000 A5DC       LDA $DC   ;GET FLAG
 40 0002 C901       CMP #$01  ;COMPARE TO
ONE
 50  0004 F00B          BEQ PRLN    ;FULL
TRACE
 60 0006 900C       BCC END   ;NO TRACE
 70 0008          ;TRACE ROUTINE
 80 0008 2000FD    JSR $FD00  ;GET KEY
 90 000B  C903      BCC END     ;IS IT A C
?
100  000D F005          BEQ END     ;NORMAL
CTRL C
110 000F 9003      BCC END     ;IS IT AN A
?
120 0011 205AB9 PRLN JSR $B95A  ;DISPLAY
LINE NUMBER
130 0014 4C94FB END  JMP $FB94  ;EXIT TO
CTRL C
```

LISTING #3

```
10   REM-SUPER TRACE  &  LISTER FOR OSI
MONITOR ROM
20   REM-POKE220,0 FOR NORMAL OPERATION
30   REM-POKE220,1 FOR FULL TIME TRACE
40   REM-POKE220,2 FOR SINGLE STEP TRACE
50    REM-IN  SS MODE CTRL A EXC ONE STEP
AT A TIME/NO LINE DISPLAY
60    REM-AND CTRL  S EXC ONE  STEP AND
DISPLAYS THE LINE NUMBER
70     REM-CTRL  A  MAY ALSO BE USED IN A
LISTER FUNCTION,JUST TYPE IN LIST
80    REM-AND PRESS  AND HOLD CTRL A  TO
LIST AS MANY LINES AS YOU WANT-
90    REM-RELEASE  CTRL A TO STOP  THE
LISTING, AND PRESS IT AGAIN TO-
100 REM-RESTART YOU LISTING-
110  FORA= 667  TO  689  :READ  B:
POKEA,B:NEXTA
120 POKE220,0: POKE541,2
130  DATA165,220,201,1,240,11,144,
12,32,0,253,201,3
140  DATA240,5,144,3,32,90,185,
76,155,255
```

LISTING #4

```
10  REM-SUPER TRACE & LISTER FOR C1E
MONITOR ROM
20  REM-POKE220,0 FOR NORMAL OPERATION
30  REM-POKE220,1 FOR FULL TIME TRACE
40  REM-POKE220,2 FOR SINGLE STEP TRACE
50  REM-IN SS MODE CTRL A EXC ONE STEP
AT A TIME/NO LINE DISPLAY
60  REM-AND CTRL S EXC ONE STEP AND
DISPLAYS THE LINE NUMBER
70  REM-CTRL A MAY ALSO BE USED IN A
LISTER FUNCTION,JUST TYPE IN LIST
80  REM-AND PRESS AND HOLD CTRL A TO
LIST AS MANY LINES AS YOU WANT-
90  REM-RELEASE CTRL A TO STOP THE
LISTING, AND PRESS IT AGAIN TO-
100 REM-RESTART YOUR LISTING-
110 FORA=660 TO 682 :READ
B:POKEA,B:NEXTA
120 POKE220,0 :POKE541,2
130 DATA165,220,201,1,240,11,144,
12,32,0,253,201,3
140 DATA240,5,144,3,32,90,185,76,
148,251
```

CORRECTION , JAMES SCHAFER, IOWA

The hardware mod, "Making Your Disk Quiet" by Dave Pompea, found in last month's Journal, needs a touch up before it will work.

1. Pins 6 and 7 of the 74123 are reversed.
   (R) should be pin 7 and (C) should be pin 6.

2. The (motor on) signal willnot reach the disk
   until you install a jumper across the two
   pads in position 4 of the disk interface
   connector board.

Thanks to Dave's mod, my disk can be on for long periods of time without unnecessary wear on my diskettes and drive.

CRAIG LINDBLAD, MESA, ARIZONA

I have been disappointed at how little has been written about PICO DOS. I am still saving up to buy the OS65D operating system but feel that PICO DOS has use for the programmer such as quick loading of files which you use often. (checkbook balancing, games, etc.) I have discovered after PEEKing and POKEing around, how to get back into PICO DOS after hitting the BREAK key and warm starting. The following POKEs will do it:

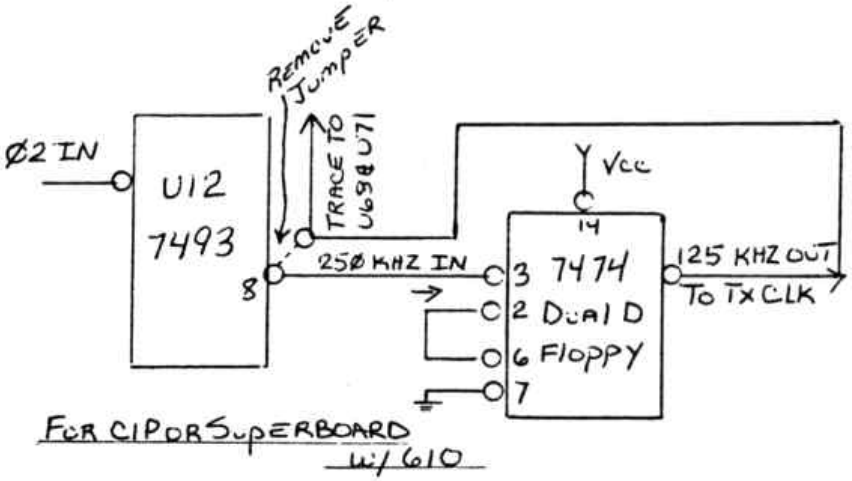
POKE542,153:POKE543,37:POKE544,164:
POKE545,37.

You can now use your SAVE and LOAD commands as before. Come on you PICO DOS users - let's come up with some documentation. P.S. Start by POKEing the first value only to test for an error message. It will save time having to do it all over again.

Dear Mr. Lindblad:
Did you know that PICO DOS will store machine code? Assemble the code below 8K, boot up the PICO DOS and "SAVE". To make it self starting, POKE the start vector (HH,LL) into locations 01 and 02. P.S. Dave Edson figured out how to do it.

THOMAS OWEN, MIAMI FLORIDA

When conversion has been made to 2MHZ system operation (C1P) you will encounter errors when trying to write to a disk because the transmit clock has doubled from 125KHZ to 250KHZ. Leave system at 2MHZ and add the flip-flop to divide it back down. All disk operations will work fine after this mod has been made. I know a lot of people who have doubled their speed on a C1 and after adding disk could not understand why it wouldn't work at 2MHZ.



RANDOM ACCESS FILES
BY DICK SNELL

I had an application where I needed to use large random access files (I needed thirty-nine tracks worth), but the DOS for my C4P-MF (with 24K) was inadequate. It had the following drawbacks. First, the CREATE program wouldn't let me start a file below track 12 (I wanted to use a data-only disk and start on track one). Second, the DOS limits you to record lengths of powers of two (8,16,32,64,etc.) and I never could get the different record lengths to work consistently. I wanted to use record lengths of 27 bytes (plus one byte for the carriage return at the end of every record written to the buffer). Third, the DOS is too slow. It rereads a track into the buffer even though that track is already in the buffer.

The following system overcomes all of these drawbacks. It might look complicated at first, but it is powerful and worth using, especially if you need record lengths other than powers of two.

First, let's look at the housekeeping that has to be set up in the early part of the main program. Don't forget to put a buffer in front of your program (use the CHANGE program).

```
110  RL=28:B1=12926:NRPT=INT(2048/RL):
NBUPT=RL*NRPT
112 HC=256:FT=20
114 S$="CA 327E="+RIGHT$(STR$(100+FT)
,2)+",1":DISK!S$
116 CT=FT:NT=FT
```

In line 110, RL is the Record Length in bytes (plus one byte for the carriage return symbol). In this example, RL is set to 28 since I wanted records 27 bytes long. B1 is the first memory location for buffer #1 (12926=$327E). NRPT is the Number of Records Per Track; it is equal to the number of bytes per track (2048) divided by the record length in bytes. NBUPT is the Number of Bytes Used Per Track; it is equal to the record length in bytes multiplied by the number of records per track. For this example, NRPT=73 and NBUPT=2044, which means we "waste" four bytes for each track. As you experiment with other record lengths, you'll see that the number of "wasted" bytes per track varies quite a bit.

In line 112, HC is a constant used later in hexadecimal conversions. FT is the First Track in the file; for this example I am using track 20 as the first track.

Line 114 brings the first track of the file into the buffer. This is equivalent to the DISK OPEN command. Line 116 sets CT and NT equal to the first track. CT is the Current Track in the buffer. NT is the New Track which is to be brought into the buffer.

Next let's look at the subroutine which actually does the track and memory pointer manipulations. This subroutine is equivalent to the DISK GET command.

```
10 NT=FT+INT(RN/NRPT):IFCT=NTTHEN16
12  S$="SA"+RIGHT$(STR$(100+CT),2)
+",1=327E/8":DISK!S$:CT=NT
14 S$="CA 327E="+RIGHT$(STR$(100+NT)
,2)+",1":DISK!S$
16 Q1=B1+(RL*RN-NBUPT*INT(RN/NRPT)):
Q2=INT(Q1/HC)
18  POKE9133,Q2:POKE9156,Q2:POKE9132,
Q1-HC*Q2:POKE9155,Q1-HC*Q2:RETURN
```

Line 10 calculates the new track required to be brought into the buffer. RN is the record number you wish to access. For example, if you want to examine the 244th record in your file, set RN=244 in the calling program. If the new track is the same as the track which is currently in the buffer (CT), then the track saving and calling steps (lines 12 and 14) are skipped.

If the correct track is not already in the buffer, then line 12 saves the information currently in the buffer to the appropriate track and updates the value for CT. Similarly, line 14 calls the appropriate track into the buffer.

Lines 16 and 18 calculate the necessary parameters and set the buffer #1 input and output pointers to the appropriate values.

There is one more step before we can use the system, preparing a data disk. The call statement in line 14 will give a #9 error unless something is already written on the track. In other words, an initialized track won't work! I use the following program to fill my data disk (all 39 tracks) with zeros and carriage return symbols.

```
1000 POKE9156,50:POKE9155,126
1010 FORI=0TO1024:PRINT#6,"0":NEXTI
1020 FORI=1TO39
1030 S$="SA"+RIGHT$(STR$(100+I),2)+
     ",1=327E/8":DISK!S$
1040 NEXTI:END
```

Once you have a data disk prepared, using the system is straightforward. All you need to do in your main program is to set RN equal to the record number you wish to access, and then GOSUB 10. The subroutine will bring the appropriate track into the buffer and set the buffer input and output pointers to the correct values. As an example, add the following statements to the above.

```
1   GOTO110
130 X$="THIS IS RECORD NUMBER"
140 FORI=0TO1459:RN=I:GOSUB10
150 PRINT#6,X$+STR$(I):NEXTI
160 GOSUB12:END
```

When you have this program ready, put a prepared data disk into your disk drive and run the program. The program will put the record number message (0 through 1459) into the appropriate locations on the different tracks. You can see the results by calling the various tracks into D200. The GOSUB 12 statement in line 160 is the equivalent of a DISK CLOSE statement. It should always be your last operation when you finish working with the files, otherwise you will lose the information most recently written into the buffer.

I use this system extensively for handling random access files with various record lengths, and it works beautifully. I hope it will be just as helpful in your applications.

SPEEDING UP DISK DIRECT FILES

Written for C8P-DF, C4 in ()

One of the nice features of OSI BASIC is that it supports DIRECT or RANDOM ACCESS files. However, the use of these files are time consumming, annyoying, and wasteful due to the manner in which the system handles them.

The system treats a direct file as a collection of records. The records of any one file are all the same length and the length may be any power of 2 up to 256 bytes long (e.g. 2,4,8,16,32,64,128,256) (ref. Aardvark Journal #2 and correction Aardvark Journal #4). In applications where data is not accessed sequentially but accessed based on a decision, the direct file is essential.

A look at sequential files will help point out the advantages of direct files for certain applications. A sequential file could be used in the following manner to read "record" N.

LISTING #1

```
900 DISK OPEN,6,FILE$
910 FORI=1TON:INPUTR$:NEXT
920 DISK CLOSE,6:RETURN
```

This method requires that the file be read from the beginning every time any element is read; it may be sped up by adding some controls.

LISTING #2

```
10 DISK OPEN,6,FILE$
.
.
.
900 IFN>LAST THEN FIRST=LAST+1:GOTO920
910 DISK CLOSE,6:DISK
OPEN,6,FILE$:FIRST=1
920 FORI=FIRST TO N:INPUT R$:NEXT
.
.
.
999 DISK CLOSE,6:END
```

The addition above allows the program to continue reading from the last required "record" if the new "record" is beyond it in the file. However, the one feature of direct file records which I have ignored until now is that direct file records may have more than one output in them where the sequential file which is masquerading as a direct file, has one string or variable as its' "record".

If, for instance, you wanted to store a mailing list, everyone uses mailing lists as examples don't they, it would be easier to output the name, address, city, state, and zip to a record in a direct file as opposed to storing the data in a sequential file. Why it is easier is another topic. To do this in a direct file is as follows:

LISTING#3

```
10 DISK OPEN,6,FILE$
.
.
.
100 GOSUB900:REM WRITE RECORD
.
.
.
200 GOSUB950:REM READ RECORD
.
.
.
900 DISK GET 6,RECORD
910 PRINT#6,NAME$:PRINT#6,ADDRESS$:
PRINT#6,CITY$
920 PRINT#6,STATE$:PRINT#6,ZIP$
930 DISK PUT 6:RETURN
950 DISK GET 6,RECORD
960 INPUT#6,NAME$,ADDRESS$,CITY$,
STATE$,ZIP$
970 DISK PUT 6:RETURN
.
.
.
999 DISK CLOSE,6:END
```

This brings us to the time consuming and annoying part of OSI direct files which is that everytime a record is accessed the system reads in the disk track which contains the record and if the operation is a PRINT the system writes out the track to disk when finished. These computers are supposed to be smart; why can't the silly thing figure out if the disk track containing the new record is already in the disk buffer because the old record was on the same track. There are two ways in which the system is inefficient in direct file storage; the first is the use of only 12 pages/track instead of 13 and the second is the fact that each PRINT statement generates a carriage return ($0D) and a line feed ($0A) when only the carriage return is used by the INPUT statement and the line feed is ignored. In the mailing list example there are 5 bytes in each record taken up with line feeds.

Now that we know the problem, what can be done about it? Since the computer is faster than the disk drive, it seems to me like a good trade to expend a few more RAM bytes in the program and save a few disk bytes in every record. In order to do this several utility subroutines need to be developed. I prefer to put these in subroutines in order to provide transportability from program to program.

FUNCTION 1: OPEN FILE

The first thing we need to do is locate the file on the disk and create a buffer area at the top of memory. The track is located by reading the directory from track 8(12).

VARIABLES

```
R  - Record being requested   A$ -
Temporary
RL - Record length                 I  -
Temporary
RT - Records/track              J  -
Temporary
TF - First track              FILE$ - File
name
TK - Present track              W  - Write
flag
Z$ - Carriage return
```

LISTING #4

```
10 POKE132,255:POKE133,177:CLEAR
20 INPUT"FILE";FILE$:RL=64:GOSUB9010
```

LISTING #5

```
9000 REM READ DIRECTORY & INITIALIZE
9010 TK=1:RT=INT(3328/RL):Z$=$CHR$(13)
9020 IF  LEN(FILE$)<6  THEN
FILE$=FILE$+"":GOTO9020
9030 DISK!"CA BF00=08,1:GOSUB9100
9040 IF TK<>0THEN RETURN
9050 DISK!"CA BF00=08,2:GOSUB9100
9060 IF TK<>0THEN RETURN
9070 PRINT F$;"NOT FOUND":END
9100 A$=""
9110 FORI=48896TO49151STEP8:IFPEEK(I)=
35GOTO9150
9120  FORJ=ITOI+5:A$=A$+CHR$(PEEK(J)):
NEXTJ
9130 IFA$<>FILE$THENA$="":GOTO9150
9140  I=PEEK(I+6):TK=I-6*INT(I/16):
RETURN
9150 NEXTI:RETURN
```

Now we know the first track of the file. Change the POKE at 133 for systems with less than 48K memory. The CLEAR will reinitialize BASIC's pointers.

FUNCTION 2: WRITE RECORD

Both the writing and reading of records uses device #5, memory I/O. In order to write an entire record, execute a:

LISTING #6

```
9200 REM WRITE RECORD
9210 I=TF+INT(R/RT):IFI=TKGOTO9240
9220 IFW=1THENGOSUB9860
9230 TK=I:GOSUB9810
9240 GOSUB9960
9250 PRINT#5,A;Z$;B$;Z$;
9260 W=1:PRINT#9:RETURN
.
.
.
9800 REM CALL DISK TRACK
9810 POKE9098,128:POKE9099,178:
POKE9105,128:POKE9106,178
9820 PRINT#5,"DISK!"; CHR$(34);"CA
B300=";  RIGHT$(STR$(TK),2);",1"
9830 PRINT#5,"GOTO9840":DISK!"IO
10,10:END
9840 POKE9105,0:DISK!"IO 02,02:RETURN
9850 REM SAVE DISK TRACK
9860 POKE9098,128:POKE9099,178:
POKE9105,128:POKE9106,178
9870 PRINT#5,"DISK!";CHR$(34);"SA";
STR$(TK);",1=B300/D"
9880 PRINT#5,"GOTO9890":DISK!"IO
10,10:END
9890 POKE9105,0:DISK!"IO 02,02:RETURN
.
.
.
9950 REM SET DEVICE #5 OUTPUT
9960 X=(R-(TK-TF)*RT)*RL:Y=INT(X/256)
9970  POKE9105,X-(Y*256):POKE9106,179+Y:
RETURN
```

This code checks to determine if the track in the buffer is the same one being call for. If it is, no disk transfers occur. If it is not, then the WRITE FLAG is tested; if the buffer is called. Line 9250 may be any set of PRINT statements, however, each user variable should be followed by a Z$ (carriage return) and the semicolon used to pack the output. The PRINT#9 ends the PRINT statements and dumps a carriage return/line feed to a null device.

FUNCTION 3: READ RECORD

The method in which a record is read is similar to a write. To read an entire record, execute a:

LISTING #7

```
9300 REM READ RECORD
9310 I=TF+INT(R/RT):IFI=TKGOTO9340
9320 IFW=1THENGOSUB9860:W=0
9330 TK=I:GOSUB9810
9340 GOSUB9910
9350 INPUT#5,A,B$
9360 RETURN
.
.
.
9900 REMSET DEVICE #5 INPUT
9910 X=(R-(TK-TF)*RT)*RL:Y=INT(X/256)
9920 POKE9098,X-Y*256:POKE9099,179+Y:
RETURN
```

As in the write routine, the code checks to determine if the track in the buffer is the same one being called for. If it is not, then the WRITE FLAG is tested. If the buffer is dirty, then the old track is written and the WRITE FLAG is reset before the new track is called. Line 9350 may be any set of input statements but these inputs should exactly match the outputs on line 9250.

FUNCTION 4

At the end of the program, this function must be called to insure that all the data which was written is indeed placed on the disk. To call, execute a: GOSUB9410

LISTING #8

```
9400 REM CLOSE FILE
9410 IF 2=0 THEN RETURN
9420 GOSUB9860:RETURN
```

This code checks the WRITE FLAG and saves the buffer to disk if the flag is set.

Listings 5-8 are written to be put together as a package and were listed separately for clarity in explanation. Listing #4 is user dependent but all of the functions must be performed. Like all things in the real world there are advantages and disadvantages. I believe the advantages win in this case.

The disadvantages are: First, the records may be over written past their defined length and garbage up the beginning of the next record. Although, I never tested OSI BASIC to determine if and how it protects you from this. A page of memory just below the disk buffer is dedicated to pass to DOS the CALL and SAVE commands. Lastly, the subroutines take up program space.

The advantages are: First the disk file space is 8.5% more efficient since there are 13 pages instead of 12 on a track and additional bytes are picked up by not putting linefeeds in the records. The program track storage is less because OSI BASIC stores the disk buffer along with the program on the disk. The average read/write time of a record is much lower since these subroutines run much faster than disk I/O. These routines will work for any record length within the physical capabilities of the disk. Lastly you don't have to sit there and hear the ?!$%* being pounded out of your disks and drive by the PUT/GET routines. In closing, consider initializing a track of 64 byte records. The PUT/GET routines would go to the disk no less than 96 times where the subroutines listed go to the disk only 2 times.

NOTE: To change the subroutines for systems smaller than 48K use the following.

OFFSET = 4*(48-MEMSIZE)
POKE'S to 133 = LISTED VALUE - OFFSET
POKE'S to 9099 & 9106 = LISTED VALUE - OFFSET
"B3" in lines 9820 & 9870 = $B3 - OFFSET

USING THE C1E ROM WITH OS65D
BY DAVE POMPEA

When I plugged my new C1E ROM into my C1P/MF system and tried out the new features I was astounded by all the stuff that they had put in it. However, the editor and screen window were not usable with OS65D. A short note in the manual said that a patch would be made available soon, but that was six months ago. I grew impatient and wrote my own. The main problem is that the C1E ROM uses memory from $200 to $232 and OS65D BASIC also uses that area for the interpreter. BASIC would bomb if the new editor or screen driver were used. The solution is to swap this area of memory with a save area depending on which routine (BASIC or C1E ROM) was being executed. A nice place for the save area and the program to do the swapping is the space used by the DOS screen driver ($2599 & up) since we don't need it anymore. The other change needed is that the C1E rom doesn't recognize a backspace code ($08), so we changed it to delete code ($5F) which it can do. To use the patch, just run the basic program. It will poke the code in and then clear the screen to reset the cursor to the top of the screen.

To change scroll windows or screen width, poke the change to the save area, not to $200-$232. The save area starts at $25EF (9699).

```
 10                ; C1E DOS patch by Dave Pompea 9/81
 20                ;
 30                ; For Aardvark Journal
 40 2599           *     = $2599
 50                H7FFE
 60 2599 4CCF25          JMP OUTPUT
 70 259C A232    SWAP    LDX #$32
 80 259E BD0002  SWAP1   LDA $200,X
 90 25A1 BDE325          LDY S.AREA,X
100 25A4 9DE325          STA S.AREA,X
110 25A7 98              TYA
120 25A8 9D0002          STA $200,X
130 25AB CA              DEX
140 25AC 10F0            BPL SWAP1
150 25AE A5FE            LDA $FE
160 25B0 AC1626          LDY P0.FE
170 25B3 8D1626          STA P0.FE
180 25B6 84FE            STY $FE
190 25B8 A5FF            LDA $FF
200 25BA AC1726          LDY P0.FF
210 25BD 8D1726          STA P0.FF
220 25C0 84FF            STY $FF
230 25C2 60              RTS
240 25C3 200D25 SW.IN    JSR $250D
250 25C6 4C9C25          JMP SWAP
260 25C9 209C25 SW.OUT   JSR SWAP
270 25CC 4C0D25          JMP $250D
280 25CF 48     OUTPUT   PHA
290 25D0 20C325          JSR SW.IN
300 25D3 68              PLA
310 25D4 48              PHA
320 25D5 C908            CMP #8
330 25D7 D002            BNE OUT1
340 25D9 A95F            LDA #$5F
350 25DB 2036F8 OUT1     JSR $F836
360 25DE 20C925          JSR SW.OUT
370 25E1 68              PLA
380 25E2 60     OUT.RT   RTS
390 25E3 10     S.AREA   .BYTE $10,$20,$31,0,0,0,0,$A9
390 25E4 20
390 25E5 31
390 25E6 00
390 25E7 00
390 25E8 00
390 25E9 00
390 25EA A9
400 25EB 88       .BYTE $08,$B3,$09,$01,$39,$ED,$04,$59
400 25EC B3
400 25ED 09
400 25EE 01
400 25EF 39
400 25F0 ED
400 25F1 04
400 25F2 59
410 25F3 8F       .BYTE $8F,$1B,$00,$31,$64,$31,$31,$32
410 25F4 1B
410 25F5 00
410 25F6 31
410 25F7 64
410 25F8 31
410 25F9 31
410 25FA 32
420 25FB 46       .BYTE $46,$FB,$9B,$FF,$94,$FB,$70,$FE
420 25FC FB
```

```
420 25FD 9B
420 25FE FF
420 25FF 94
420 2600 FB
420 2601 70
420 2602 FE
430 2603 7B             .BYTE $7B,$FE,$17,$85,$D0,$85,$D3,$BD
430 2604 FE
430 2605 17
430 2606 85
430 2607 D0
430 2608 85
430 2609 D3
430 260A BD
440 260B 85             .BYTE $85,$D0,$9D,$85,$D2,$CA,$60,$00
440 260C D0
440 260D 9D
440 260E 85
440 260F D2
440 2610 CA
440 2611 60
440 2612 00
450 2613 20             .BYTE $20,$85,$D0
450 2614 85
450 2615 D0
470 2616 00       P0.FE .BYTE 0
480 2617 00       P0.FF .BYTE 0
490                     ;
500 253B          *     =   $253B
510 253B C325           .WORD SW.IN
520 252C          *     =   $252C
530 252C C925           .WORD SW.OUT
540 2531          *     =   $2531
550 2531 2046FB         JSR $FB46
```

SEPARATE THAT DISK DRIVE
CHARLES MAGUIRE III

(We have had a lot of questions about putting non-OSI drives on OSI systems - here's how to do it.)

It was way back at the beginning of January that I started the long and cheap way of upgrading my C2-4P. I got everything working the end of June!

I learned quite a few shortcuts OSI uses. The biggest one was the paddle board they use to crosswire the ribbon cable; I didn't know they were available so I cross jumpered the 34 pin ribbon cable to the 24 pine connector BY HAND (time consuming?).

Now down to the disk drive. I bought an MPI B51 from my dealer for about $280 (including case and power supply). Plugged it in and it didn't work with the FL-470 (D & N) board and the monitor ROM mod--then I started to learn what a Data separater is and how you go about making it work.

I told my dealer to get me a data separater. He got it from MPI directly and sold it to me for $40. I plugged it in and still nothing worked. Then I borrowed an OSI drive, plugged it in and a few seconds later I was looking at the BEXEC* program of OS65D. That narrowed my problems to the disk drive.

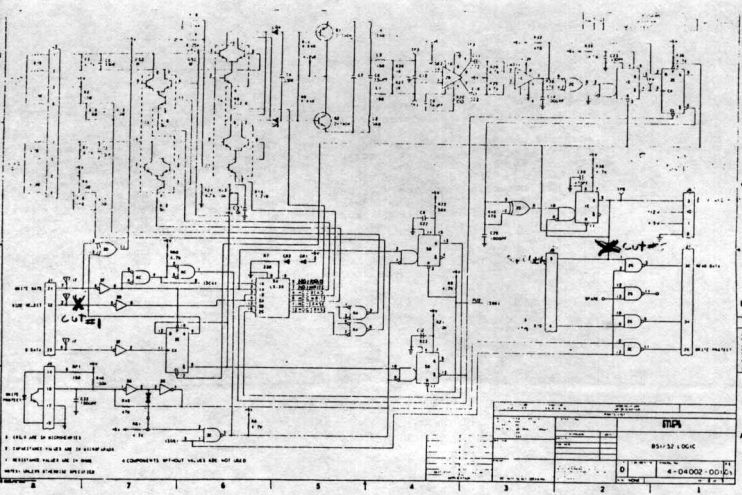
From there I spent about 35 hours looking and comparing the OSI drive with my drive (sound easy?-drives were different assemblies but still both were B51's). The first thing found was a small cut in one of the traces -- it looked like a spec of dust! This cut was the side select line from pin 32. It is located underneath J1 (34 pin edge connector-see diagrams) on the solder side of the main PC board.

That was easy to find but the drive still didn't work. Since the drives were of different assemblies I could not compare them one for one.

From there I started comparing signals with a scope. I started looking at the board again when I found some differences. And there it was--another foil cut. This one was very neatly done and hidden on the component side of the board--underneath the edge of the chip. This is where you might have to do some trace searching. On my drive the traces in this area were slightly different. I had to make the cut on the back side of the board. But wherever you make the cuts, be sure to check the schematics so you get the right ones at the right place.

As soon as I made these cuts to my drive, I plugged it in and everything worked. I have used it for several weeks now and have had no data loss.

If you have any questions or problems please call (717) 854-5830.



1A - CA3054	1G - SHUNT	2E - 74LS74	3D - 74LS14	4F - LM339	5G - 75478	6Ga - 75478	
1C - LM311	2A - CA3054	2F - 74LS14	3E - 743B	4G - 22K DIP	6A - 75463	6Gb - 75478	
1E - 74LS123	2B - NE592	2G - 743R	3G - 74LS14	5A - 74LS139	6C - 74LS33		
1F - TERM	2D - 74LS86	3C - 74LS169	4C - 74LS10	5B - 74LS123	6E - 74LS123		

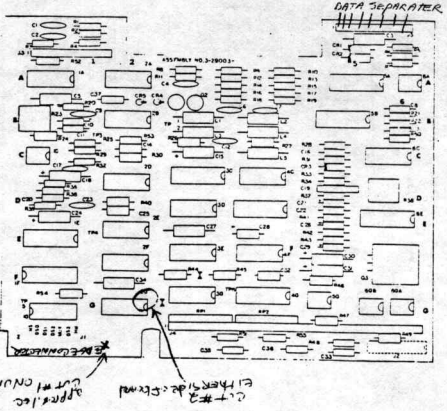


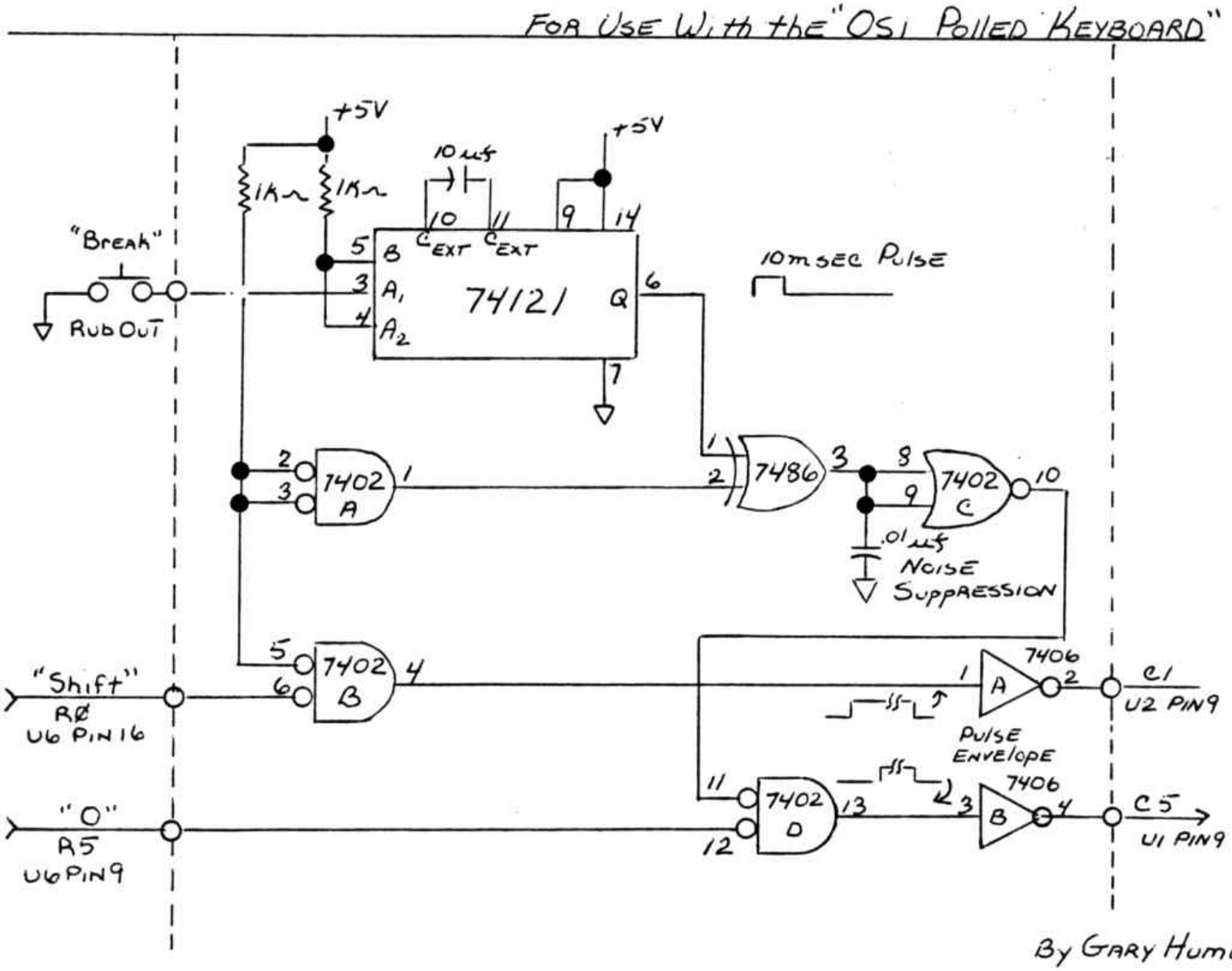
Figure B-1: PCBA Component Layout

GARY HUME, WARSAW, NEW YORK

After reading the last Aardvark Journal and in particular the letters on the BREAK key modifications, I decided to share my keyboard modification to an 8PDF system. First I disconnected the leads from the BREAK key. I then mounted a small push-button switch in the top left corner of the keyboard out of the normal path of finger travel while typing, labeled it 'BOOT' and connected the leads from the BREAK key to it.

With one problem solved, I then designed and built a small TTL circuit to combine the SHIFT '0' for backspace into one key. The BREAK key is a natural to activate the TTL circuit. As an added touch, I switched the BREAK key cap with the RUBOUT key cap. The only thing left is to change a few game instructions and remember to use the RUBOUT key for deletion.

I used four IC's to make the SHIFT key and delay the '0' key by ten miliseconds. The 7406's provide open collector outputs to interface with the keyboard outputs. The circuit is mounted near the keyboard with leads soldered directly to the PC board traces.



RALPH SHERRICK, HARRISBURG, PENNSYLVANIA

After reading my Aug. Aardvark Journal I was especially interested in Dave Pompea's disk modification and Dave Sugar's pokes for OS65D.

I think Dave's mod is fantastic-the way OSI should have done it the first time around. However, Dave stopped a little short of something for the man who has everything-or at least the guy who has dual floppy drives.

I have taken Dave's modification and added logic gates for dual drives and now I have complete automatic operation. No special addressing, no switches to throw, just call up either drive and you have it without fuss. Note that the numbers shown in the Journal (on U76B6,7) were reversed. Also be sure to tie (CLR-1,CLR-2) pins 3 & 11 high on the 74LS123 as shown and watch polarity on capicitors C1-C2.

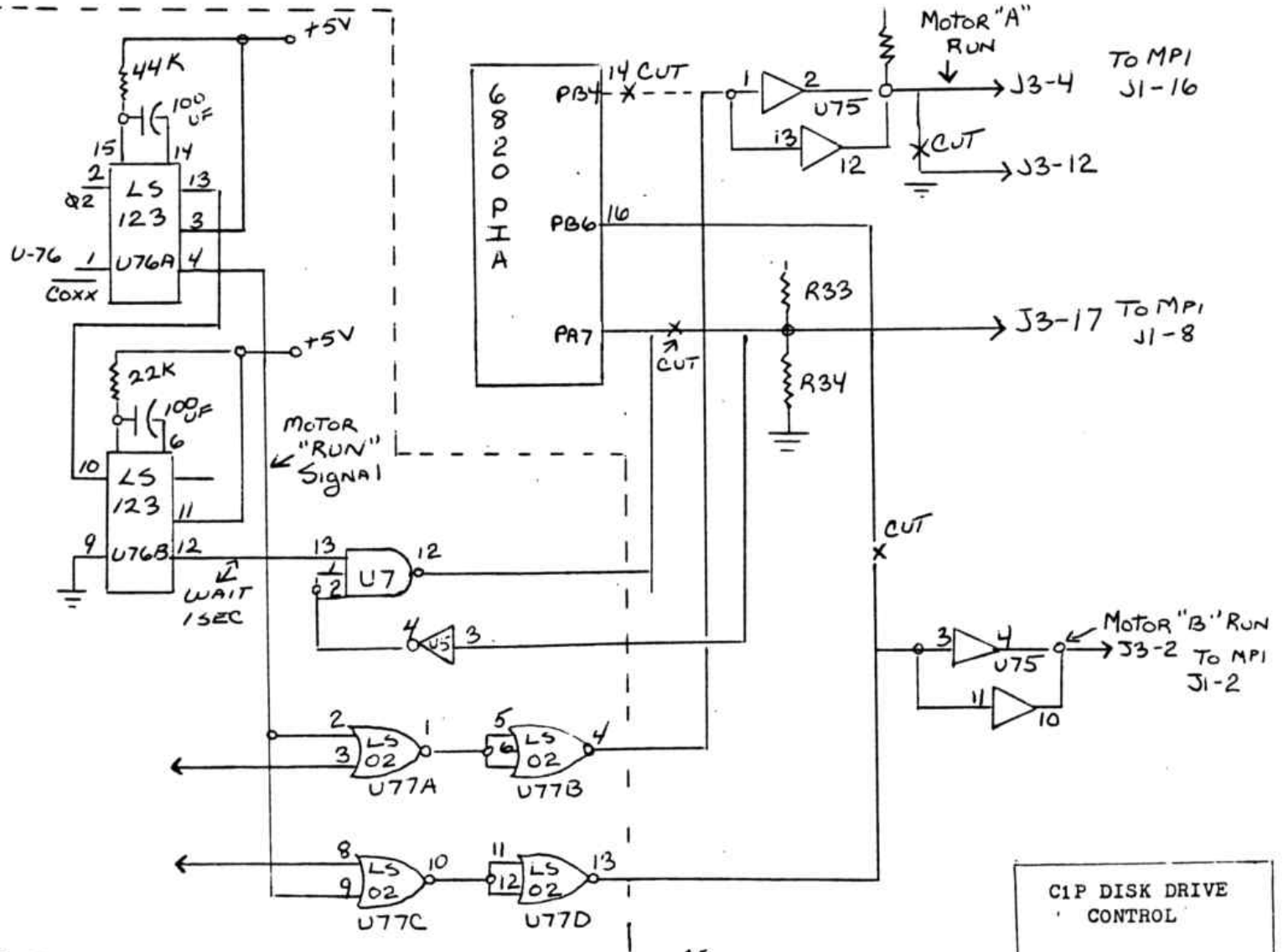
To Dave's instructions for the modification, add the following for 2 drives:

4-B Solder a jumper from 610 board J3-2 to MPI J1-2.
(This is listed as a spare by MPI)

6-B Cut the trace from U72-16. You can use the plated
thru holes nearby for solder pads.

9. Change to read...connect the 74123 pin 4 to added
74LS02 pins 2 and 9.

13. Connect U77"A"-3 to U67-6 (select "A" signal)
14. " " U77"C"-8 to U67-3 (select "B" signal)
15. " " U77"B"-4 to U75-1 (motor "A" run)
16. " " U77"D"-13 to U75-3 (motor "B" run)
17. The "A" drive wiring should be OK but you will need
to cut the trace on the "B" drive at the edge
connector J1-16.
18. Locate a pad at J1-2 and solder a jumper from J1-2
to J1-16.



```
 1 ***** LIVING PATTERNS *****
 3 REM by: BRUCE ROBINSON & AL STANKEWIC
Z
 10 IFPEEK(590)=161THEN15
 13 GOSUB52000
 15 GOSUB49000
 18 U1=PEEK(129):U2=PEEK(130):POKE129,0:
POKE130,212:U$=" ":FORU=1TO7
 20 U$=U$+U$+" ":NEXT:POKE129,U1:POKE130
,U2
 21 FORQ=54021TO54300:POKEQ,32:NEXT
 25 RS=252:ES=222:SR=57100
 26 IFPEEK(SR)=222THEN26
 27 CG=32
 28 I=32:POKE56832,0:IFPEEK(57088)<128TH
ENI=64
 29 POKE11,0:POKE12,253
 30 X=USR(X):J=PEEK(531):IFJ=13THENPRINT
:GOTO30
 40 IFJ=32THENPRINT" ";:GOTO30
 41 IFJ=127THEN45
 42 GOSUB100
 43 GOTO30
 45 POKE11,103:POKE12,2
 50 FORQ=1TO9E9:X=USR(X)
 55 IFPEEK(SR)=RSTHEN29
 56 IFPEEK(SR)=ESTHENRUN
 60 NEXT
 100 IFJ<58ANDJ>47THEN110
 101 IFJ=85THENCU=CU-I:POKECU,CG:RETURN
 102 IFJ=77THENCU=CU+I:POKECU,CG:RETURN
 103 IFJ=72THENCU=CU-1:POKECU,CG:RETURN
 104 IFJ=75THENCU=CU+1:POKECU,CG:RETURN
 105 IFJ=74THENGI=GI+1:GOSUB400:RETURN
 106 GOTO150
 107 IFJ=95THENGOSUB200:RETURN
 108 IFJ>240THENGOSUB300:RETURN
 109 PRINT"*";:RETURN
 110 IFJ=48THENPRINT"**********************
*****";:RETURN
 120 FORK=1TOJ-48:PRINT"*";:NEXT:RETURN
 150 IFJ=89THENCU=CU-I-1:POKECU,CG:RETUR
N
 152 IFJ=73THENCU=CU-I+1:POKECU,CG:RETUR
N
 154 IFJ=78THENCU=CU+I-1:POKECU,CG:RETUR
N
 156 IFJ=44THENCU=CU+I+1:POKECU,CG:RETUR
N
 160 GOTO107
 200 KL=PEEK(512):POKE54016+KL,32
 210 POKE54015+KL,95:POKE512,KL-1
 299 RETURN
 300 FORK=1TOJ-240:PRINT" ";:NEXT:RETURN
 400 IFGI/2=INT(GI/2)THENCG=32:POKECU,CG
:RETURN
 401 CG=42:POKECU,CG:RETURN
 49000 DATABORDER
 49002 READA$:IFA$<>"BORDER"THEN 49002
 49004 FORQ= 5900 TO 5990 :READM:POKEQ,M
:NEXTQ
 49008 DATA165,255,164,232,145,240,136,2
08,251,164,233,145,242,200,208
 49010 DATA253,162,3,181,244,149,235,202
,16,249,160,0,166,239,165
 49012 DATA255,145,235,24,165,234,101,23
5,133,235,144,2,230,236,202
 49014 DATA16,238,166,239,165,255,145,23
7,24,165,234,101,237,133,237
 49016 DATA144,2,230,238,202,16,238,96,2
34,234,36,36,36,36,36
 49018 DATA36,36,36,36,36,36,36,36,36,36
,36,36,36,36,36
 49020 DATA36
 49050 GOTO49500
 49100 :                    E8 (232) TOP
LENGTH     E9 (233) BOTTOM STRT
 49110 E8=232                        TOP LEN
GTH
 49120 E9= BOTTOM LENGTH          EA=INCREM
ENT
 49150 EF (239)= LENGTH OF DOWNWARD BORD
ERS
 49160 F0,F1= TOP LEFT BORDER OF SCREEN
 49170 F2,F3= BEGINNING OF LAST PAGE IN
MEMORY
 49180 F4,F5= CONST. FOR UPPER LEFT
 49190 F6,F7=CONST. FOR UPPER RIGHT
 49200 FF=BORDER GRAPHIC
 49500 POKE241,208:POKE240,0
 49510 POKE243,211:POKE242,0
 49520 POKE245,208:POKE247,208
 49530 REM TOP LEFT
 49540 POKE244,4
 49550 REM TOP RIGHT
 49560 POKE246,29
 49570 POKE232,128
 49580 POKE233,128
 49590 POKE234,32
 49600 POKE255,32
 49610 POKE239,32
 49620 CU=53776
 49649 PRINT
 49650 PRINT" SCREEN SIZE? 1 - 24X24"
 49651 PRINT"              2 - 25X25"
 49652 PRINT"              3 - Wrap-
around"
 49655 POKE11,0:POKE12,253:X=USR(X)
 49656 ONPEEK(531)-48GOTO49980,49657,510
00
 49657 PRINT
 49700 POKE241,207:POKE240,224
 49730 POKE246,30
 49750 CU=53745
 49755 GOTO49990
 49980 PRINT
 49990 POKE5915,251
 50000 DATALIFE
 50001 IFPEEK(590)=161THENRESTORE:RETURN
 50002 READA$:IFA$<>"LIFE"THEN 50002
 50004 FORQ=590TO712:READM:POKEQ,M:NEXT
 50006 POKE12,2:POKE11,103
 50008 DATA161,33,0,1,2,32,34,64,65,66,1
69,0,133,224,133
 50010 DATA226,169,208,133,225,169,28,13
3,227,96,32,88,2,162,8
 50012 DATA169,0,133,228,188,79,2,177,22
4,201,42,208,2,230,228
 50014 DATA202,16,242,201,32,240,14,164,
228,192,3,240,16,192,4
 50016 DATA240,12,169,32,208,8,164,228,1
92,3,208,2,169,42,160
 50018 DATA33,145,226,230,224,230,226,20
8,200,230,225,230,227,165,225
 50020 DATA201,212,208,190,32,88,2,160,0
,177,226,145,224,230,224
 50022 DATA230,226,208,246,230,225,230,2
27,165,225,201,212,208,236,96
 50024 DATA234,234,165
 50025 S=6000
 50026 FORT=0TO9:READM:FORQ=0TO8:POKES+Q
*10+T,M:NEXTQ:NEXTT
 50030 DATA 160,0,177,224,201,42,208,2,2
30,228
 50035 FORQ=0TO8:READM:POKES+Q*10+1,M:NE
XT
```

```
 50040 DATA 66,65,64,34,32,2,1,0,33
 50050 POKE624,76:POKE625,112:POKE626,23
 50060 POKE6090,76:POKE6091,126:POKE6092
,2
 50065 POKE709,76:POKE710,12:POKE711,23
 50070 IFPEEK(57088)>127THENRESTORE:RETU
RN
 50080 FORQ=0TO8:READM:POKES+Q*10+1,M:NE
XT
 50090 DATA128,129,130,66,64,2,1,0,65
 50100 POKE611,24:POKE665,65
 50200 RESTORE:RETURN
 51000 POKE247,204:POKE245,204
 51005 POKE241,207
 51006 CU=53713
 51008 POKE240,160
 51010 GOTO49980
 52000 PRINT:PRINT:PRINT:PRINT:PRINT
 52010 PRINT"To PRINT Stars:"
 52020 PRINT"  #'s give you stars."
 52030 PRINT"  Cont #'s for spaces."
 52040 PRINT"  Cont 0 for backsp."
 52050 PRINT
 52060 PRINT"Star Cursor:"
 52070 PRINT"  Keys are around J,"
 52080 PRINT"  J changes cursor."
 52090 PRINT
 52100 PRINT"RUBOUT Starts patterns."
 52102 FORJ=1TO6:PRINTCHR$(135);:NEXT:PR
INT
 52110 PRINT"R SHIFT Stops patterns."
 52112 FORJ=1TO7:PRINTCHR$(135);:NEXT:PR
INT
 52130 PRINT"ESCAPE Runs again."
 52132 FORJ=1TO6:PRINTCHR$(135);:NEXT:PR
INT
 52900 RETURN
```

```
  1 REM ***PHONE NUMBERS***
  2 REM BY BRUCE ROBINSON
  5 REM IF NOT PRINTING RIGHT, ADJUST LIN
E 120
  8 LP=8
  9 POKE15,81
 10 GOSUB1000
 11 PRINT
 12 X=0
 14 PRINT"HIT ANY KEY TO START":I=USR(I)
:SAVE
 15 PRINTTAB(28);
 16 FORQ=1TO3:PRINTN(Q);:NEXT
 17 PRINT"-";
 18 FORQ=4TO7:PRINTN(Q);:NEXT
 19 PRINT:PRINT:PRINT
 20 FORA=1TOY(1)
 30 FORB=1TOY(2)
 40 FORC=1TOY(3)
 45 FORD=1TOY(4)
 50 FORE=1TOY(5)
 60 FORF=1TOY(6)
 70 FORG=1TOY(7)
 100 PRINTMID$(A$(N(1)),A,1);
 102 PRINTMID$(A$(N(2)),B,1);
 104 PRINTMID$(A$(N(3)),C,1);
 106 PRINTMID$(A$(N(4)),D,1);
 108 PRINTMID$(A$(N(5)),E,1);
 110 PRINTMID$(A$(N(6)),F,1);
 112 PRINTMID$(A$(N(7)),G,1);
 120 X=X+1:IFX>LPTHENPRINT:X=0:GOTO130
 130 PRINT" ";
 200 NEXT:NEXT:NEXT:NEXT:NEXT:NEXT:NEXT
 210 FORQ=1TO8:PRINT:NEXT
 220 POKE517,0
 250 STOP
```

```
 1000 DIMA$(11):REM PREVENTS OSI MACHINE
S FROM CRASHING
 1010 FORQ=0TO9:READA$(Q):NEXT
 1020 DATAOOO,111,ABC,DEF,GHI,JKL,MNO,PR
S,TUV,WXY
 1030 POKE11,0:POKE12,253
 1100 Q=1
 1110 FORQ=1TO7
 1120 PRINT"YOUR NUMBER? ";
 1130 X=USR(X)
 1132 J=PEEK(531)
 1134 PRINTCHR$(J);
 1140 IFJ>47 ANDJ<58THENN(Q)=J-48:Q=Q+1
 1150 IFQ<8THEN1130
 1200 FORQ=1TO7
 1210 Y(Q)=3
 1220 IFN(Q)<2THENY(Q)=1
 1230 NEXT
 1999 RETURN
```

LIFE FOR TWO

Life is one of the original and oldest computer games. It was initially played on teletype type terminals with rather primitive early computers. The rules are simple, but, as in all the good games, they lead to rather complex strategies.

Life is played on an N X N matrix. Animals are placed on the matrix dots. The rules are:

1. Any animal surrounded by more than 3 other animals will die from overcrowding.
2. Any animal surrounded by less than 2 other animals will die from lonesomeness.
3. Whenever 3 animals surround the same dot, they will birth a new animal in that dot.

We have included here what Robinson & Stankewicz call "Living Patterns" which is one of the finest partially machine code Life games available for the C1P. As written, it works on 8K Basic in Rom machines only. For owners of other machines, I would suggest that you look in the old issues of the AARDVARK catalog. We published a Life game done in Basic which will run on any OSI machine. It should be emphasized that the R & S game is much better.

The only bad thing about Life, as fascinating as it is, is that There is no comptetion involved and it eventually tires. R & S have therefore, come up with what should be Opus Magnus end-of-the-line end-all-and-be-all of Life games-"Life for Two". This one is competitive. You can either enter new animals on each turn or allow it to free run after you put in an initial pattern. It should run on virtually any OSI machine. I think, aside from the fact that the cells are picked by pressing a letter and a number key, that the instructions are self explanitory. It is one of the more difficult and more competive games which we have published in this Journal.

```
 1 GOTO3:BRUCE ROBINSON
 3 GOSUB25000
 4 LI=32:IFPEEK(57088)<129THENLI=64
 5 GOSUB18000:DIMF(2),A(7,7):GOSUB9000
 7 IFAA>0THENRX=1.5/AA
 8 FORQ=0TO7:FORR=0TO7:A(Q,R)=3:NEXT:NEX
T
 10 C=53454-903*(LI=64)-W1+32*(7-H1)-32*
(7-H1)*(LI=64)
 15 FORQ=7680TO7780:POKEQ,200:NEXT
 30 POKEC-LI-1,204:POKEC-LI-1+W1*3,205:P
OKEC-LI-1+LI*3*H1,203
 31 POKEC-LI-1+LI*3*H1+W1*3,206
 40 FORJ=1TOW1*3-1:POKEC-LI-1+J,131:POKE
C+H1*LI*3-LI-1+J,132:NEXT
 41 FORJ=1TOH1*3-1:POKEC-LI-1+J*LI,140:P
OKEC-LI-1+W1*3+LI*J,139:NEXT
 48 REM FORF=1TO3
 52 FORJ=1TOW1:POKEC-LI*2-3+3*J,J+48
 53 NEXT:GOSUB300
 54 FORJ=1TOH1:POKEC-LI*3-2+3*J*LI,J+64:
NEXT:IFA=1THEN3000
 55 IFL5=2THEN58
 56 SS=2:FORF=1TO3:GOSUB3100:NEXT:GOTO83
 58 FORF=1TO3:GOSUB3200:PRINTA1$", three
 animals";:GOSUB100
 61 A(HX-1,HY-1)=2:NEXT
 63 FORF=1TO3:GOSUB3200:PRINTA2$", three
 animals";:GOSUB100
 68 A(HX-1,HY-1)=1:NEXT:GOTO85
 83 GOSUB200
 85 FORJ=0TOW1-1:FORK=0TOH1-1:IFA(J,K)=0
THENA(J,K)=3
 87 NEXT:NEXT
 88 FORJ=0TOH1-1:A(W1,J)=100:NEXT:NH=3:N
C=3:GOSUB300:GOTO500
 100 E=0:EE=0:FORM=1TO2
 101 IFL5<>2THENPRINTCHR$(13)"Choose an
animal.       ";
 105 GOSUB18500:IFJ=127THENGOSUB12000:GO
TO100
 107 IFJ<64THEN160
 110 J=J-64:IFEE=1THEN168
 115 IFJ<10RJ>H1THEN180
 120 EE=1
 145 HY=J:NEXT:GOTO171
 160 J=J-48:IFE=1THEN115
 168 IFJ<10RJ>W1THEN180
 169 E=1
 170 HX=J:NEXT
 171 R5=A(HX-1,HY-1):IFR5<10RR5>2THENRET
URN
 180 PRINTCHR$(13)"***ILLEGAL***
  ";:GOTO100
 200 CX=INT(RND(8)*(W1-1)):CY=INT(RND(8)
*(H1-1))
 210 J=J+1:IFJ>20THENPRINT:RUN
 230 IFA(CX,CY)=10RA(CX+1,CY)=10RA(CX,CY
+1)=1THEN200
 240 A(CX,CY)=2:A(CX+1,CY)=2:A(CX,CY+1)=
2
 250 RETURN
 300 FORJ=0TOW1-1:FORK=0TOH1-1
 310 L=C+3*J+3*K*LI
 320 ONA(J,K)GOTO340,350:GOTO355
 340 GOSUB3110:GOTO380
 350 GOSUB3120:GOTO380
 355 POKEL,165:POKEL+1,32:POKEL+32,32:PO
KEL+33,32:GOTO380
 380 NEXT:NEXT
 390 RETURN
 500 GOSUB3200:PRINT"COMPUTING NEW BOARD
.";:IFA=1THEN575
 501 IFQ9=0THENQ9=1:GOTO510
 507 IFL$="Y"THEN521
 510 GOSUB5500
 511 GOSUB300:GOSUB516:IFD9=1THEND9=0:IF
L5<>2THEN540
 512 IFD9=1THEND9=0:IFL5<>2THEN540
 515 GOTO521
 516 F(1)=0:F(2)=0:FORJ=0TOW1-1:FORK=0TO
H1-1:ONA(J,K)GOTO517,518,519
 517 F(1)=F(1)+1:GOTO519
 518 F(2)=F(2)+1
 519 NEXT:NEXT:IFF(1)<20RF(2)<2THEN11000
 520 RETURN
 521 IFL5<>2THENPRINTCHR$(13)"COMPUTING
MOVE.         ";
 530 GOSUB5000
 531 IFL$<>"Y"THEN540
 532 GOSUB300
 533 PRINTCHR$(13)"COMPUTING BOARD      "
;
 534 GOSUB5500:GOSUB300:GOSUB516
 540 IFL5=2THENGOSUB18900:PRINTCHR$(13)"
MOVE, "A2$".";
 544 GOSUB100
 545 IFA(HX-1,HY-1)<3THENPRINTCHR$(13);"
Try again!!!";:GOTO540
 549 IFL$="Y"THENA(HX-1,HY-1)=1:GOTO570
 550 IFHX<>CXORHY<>CYTHEN560
 551 FORJ=1TO3:PRINTCHR$(13);"  ***CANCE
LLED***";
 552 FORK=1TO500:NEXT
 553 PRINTCHR$(13)"
";
 554 FORK=1TO300:NEXT
 557 NEXT
 559 'E=0:EE=0:FORM=1TO2
 560 A(HX-1,HY-1)=1:A(CX-1,CY-1)=2
 570 IFL$="Y"THENPRINTCHR$(13)"COMPUTING
 NEW BOARD  ";:GOSUB300:GOTO575
 572 GOSUB300:GOTO580
 575 GOSUB5500:GOSUB300:GOSUB516
 580 REM
 590 GOTO500
 3000 FORRI=1TOAA:GOSUB3200:PRINT"MOVE "
A1$;:SS=2:GOSUB3100:SS=1
 3005 IFRI>AXTHENIFRX>RND(8)THENRI=AA:GO
TO3020
 3007 IFCP=1THEN3500
 3010 GOSUB3200:PRINT"MOVE "A2$;:GOSUB31
00
 3015 IFRI>AXTHENIFRX>RND(8)THENRI=AA
 3020 NEXT:GOTO85
 3100 GOSUB100:A(HX-1,HY-1)=SS-CP:L=C+3*
(HX-1)+3*(HY-1)*LI
 3105 IFSS-CP=2THEN3120
 3110 POKEL,176:POKEL+1,178:POKEL+LI,177
:POKEL+LI+1,175:RETURN
 3115 L=C+3*(HX-1)+3*(HY-1)*LI
 3120 POKEL,189:POKEL+1,190:POKEL+LI,190
:POKEL+LI+1,189:RETURN
 3200 PRINTCHR$(13)"
  "CHR$(13);:RETURN
 3500 GOSUB3200:PRINT"COMPUTING MOVE";:I
FRI=1THEN3600
 3510 GOSUB5000:HX=CX:HY=CY:A(UJ,UK)=2:G
OSUB3115:GOTO3015
 3600 HX=HX+1:HY=HY+1
 3605 IFHX>1+H1/2THENHX=HX-2
 3610 IFHY>1+W1/2THENHY=HY-2
 3620 A(HX-1,HY-1)=2:GOSUB3115:GOTO3015
 5000 F(1)=0:F(2)=0:GOSUB5005
 5002 IFL5=2THEN14000
 5004 GOTO5015
 5005 N=7690:FORXX=0TO6:FORYY=0TO7
 5006 LK=A(YY,XX):IFLK<3THENF(LK)=F(LK)+
1
 5007 IFLK=3THENLK=0
 5008 IFLK>3THENLK=200
 5009 IFYY>=W1THENLK=200
 5010 POKEN,LK:N=N+1:NEXT:NEXT
```

```
 5011 F(2)=F(2)+1
 5012 RETURN
 5015 FORJ=0TOW1-1:FORK=0TOH1-1:IFA(J,K)
=3THEN5017
 5016 POKE7690+J+8*K,A(J,K):GOTO5018
 5017 POKE7690+J+8*K,0
 5018 NEXT:NEXT
 5019 H6=0:C6=0
 5020 FORJ=0TOH1-1:FORK=0TOW1-1
 5030 ONA(J,K)GOTO5040,5050:GOTO5060
 5040 H6=H6+1:GOTO5060
 5050 C6=C6+1
 5060 NEXT:NEXT
 5065 C6=C6/(H1*W1):H6=H6/(H1*W1)
 5070 CC=-1536*C6+492
 5080 IFC6>=.32THENCC=0
 5090 IFC6<=.19THENCC=200
 5092 CH=-294*H6+207
 5094 IFH6>=.36THENCH=100
 5096 IFH6<=.16THENCH=160
 5100 TU=-9E10
 5110 FORJU=0TOW1-1:FORKU=0TOH1-1
 5112 IFA(JU,KU)<>3THEN5150
 5115 POKE7690+JU+KU*8,2
 5125 GOSUB5200
 5126 POKE7690+JU+KU*8,0
 5130 TA=CC*DC-CH*DH
 5134 IFC4<2THENTA=TA-20000
 5136 IFH4<2THENTA=TA+2500:IFH4<1THENTA=
TA+5000
 5140 IFTA>TUTHENTU=TA:UJ=JU:UK=KU
 5150 NEXT:NEXT
 5160 CX=UJ+1:CY=UK+1
 5165 IFL$="Y"THENA(UJ,UK)=2
 5170 RETURN
 5200 POKE11,94:POKE12,29:X=USR(X)
 5214 POKE11,0:POKE12,253
 5250 DC=PEEK(8104)-100
 5252 DH=PEEK(8103)-100
 5254 C4=F(2)+DC:H4=F(1)+DH
 5299 RETURN
 5500 FORJ=0TOW1-1:FORK=0TOH1-1:POKE8140
+J+K*7,A(J,K):NEXT:NEXT
 5501 GOSUB5005
 5510 POKE11,94:POKE12,29:X=USR(X):POKE1
1,0:POKE12,253
 5685 IFNG=1THENRETURN
 5687 NH=0:NC=0
 5690 FORQ=7553TO7650:IFPEEK(Q)=1THENNH=
NH+1
 5700 IFPEEK(Q)=2THENNC=NC+1
 5710 NEXT
 5730 N=7553:FORX=0TOH1-1:FORY=0TO7:A(Y,
X)=PEEK(N):N=N+1
 5735 IFA(Y,X)=0THENA(Y,X)=3
 5740 NEXTY:NEXTX
 5800 RETURN
 9000 REM
 9020 GH=1:GC=2:B=3:BC=4:TC=5:BH=6:TH=7:
RETURN
 11000 PRINTCHR$(13)"    * T H E    E N D
 *  ";
 11010 GOSUB18500:IFJ=127THENGOSUB12000:
GOTO11000
 11020 PRINT:RUN4
 12000 FORJ=0TOW1-1:FORK=0TOH1-1
 12010 L3=PEEK(8140+J+K*7):POKE8140+J+K*
7,A(J,K):A(J,K)=L3
 12020 NEXT:NEXT
 12025 PRINTCHR$(13)"          OLD BOARD
  ";
 12030 GOSUB300
 12040 FORJ=0TOW1-1:FORK=0TOH1-1
 12050 L3=PEEK(8140+J+K*7):POKE8140+J+K*
7,A(J,K):A(J,K)=L3
 12060 NEXT:NEXT
 12070 X=USR(X):IFPEEK(531)=127THEN12090
 12080 GOTO12070
 12090 PRINTCHR$(13)" ***CURRENT BOARD**
*    ";:GOSUB300:RETURN
 13000 X=0:FORJ=1TO4:X=X+1
 13010 NEXT:STOP
 14000 REM
 14010 GOSUB18900:PRINTCHR$(13)"MOVE, "A
1$".";
 14015 GOSUB100
 14020 CX=HX:CY=HY
 14040 UJ=CX-1:UK=CY-1
 14045 IFA(UJ,UK)<3THENPRINTCHR$(13)"TRY
 AGAIN!!!            ";:GOTO14010
 14050 GOTO5165
 17000 U1=PEEK(129):U2=PEEK(130):POKE129
,0:POKE130,212:U$=" ":FORU=1TO7
 17010 U$=U$+U$+" ":NEXT:POKE129,U1:POKE
130,U2:RETURN
 18000 GOSUB17000
 18010 PRINT"ONE OR TWO PLAYERS? ";
 18020 GOSUB18500:IFJ<49ORJ>50THEN18020
 18025 L5=J-48:PRINTCHR$(J):PRINT:PRINT:
IFL5=1THENCP=1
 18031 PRINT"FREE RUNNING?":GOSUB18500:P
RINT:PRINT:IFJ<>89THEN18035
 18032 A=1:PRINT"MAX. ANIMAL ENTRY?":GOS
UB18500:AA=J-48
 18034 PRINT:PRINT:AX=INT(.5+AA/2)
 18035 IFL5=2THENINPUT"NAME, PLAYER 1";A
1$:INPUT"NAME, PLAYER 2";A2$
 18036 IFA=1THEN18060
 18037 A1$=LEFT$(A1$,7):A2$=LEFT$(A2$,7)
 18040 PRINT:PRINT"Take turns separately
?";:GOSUB18500
 18050 L$="N":IFJ=89THENL$="Y"
 18051 PRINT" "L$
 18052 IFL$="N"ORL5=2THEN18060
 18053 PRINT:PRINT:PRINT"WANT FIRST MOVE
? ";:GOSUB18500
 18055 IFJ=89THEND9=1:PRINT"Y":GOTO18060
 18057 PRINT"N"
 18060 REM
 18070 PRINT:PRINT"Width and Height? ";
 18080 GOSUB18500:IFJ<50ORJ>55THEN18080
 18090 W1=J-48:PRINTCHR$(J)",";
 18100 GOSUB18500:IFJ<50ORJ>55THEN18100
 18110 H1=J-48:PRINTH1
 18200 GOSUB17000:RETURN
 18500 POKE11,0:POKE12,253:X=USR(X):J=PE
EK(531):RETURN
 18900 PRINTCHR$(13)"
    ";:RETURN
 25000 POKE133,0:POKE134,29
 25022 FORQ= 8105 TO 8113 :READM:POKEQ,M
:NEXTQ
 25024 DATA9,0,1,2,8,10,16,17,18
 27004 FORQ= 7424 TO 7545 :READM:POKEQ,M
:NEXTQ
 27006 REM12,29  11,94
 27008 DATA169,100,141,167,31,141,168,31
,162,3,169,0,157,163,31,202
 27010 DATA208,250,160,9,190,168,31,189,
234,30,240,18,170,202,208,5
 27012 DATA238,164,31,208,6,202,208,6,23
8,165,31,238,166,31,136,208
 27014 DATA227,170,240,22,201,200,240,50
,173,166,31,201,3,240,42,201
 27016 DATA4,240,38,222,166,31,169,0,240
,32,173,166,31,201,3,208
 27018 DATA245,162,2,173,164,31,205,165,
31,48,1,202,208,8,169,80
 27020 DATA141,24,29,76,0,29,254,166,31,
138,172,24,29,153,128,29
 27022 DATA206,24,29,240,3,76,8,29,96,36
 27024 RETURN
```

NOTE CLASSIFIED ADS ARE $7.95 FOR UP TO 4 40 CHARACTER LINES. $2.50 FOR EACH ADDITIONAL LINE.

FOR SALE OSI C2P, 8K MEMORY INCLUDING CASSETTE. PROGRAMS ALSO INCLUDED. $350.00 CALL (313) 624-3480

FOR SALE C4P IN EXCELLENT COND. IMPROVED CASSETTE INPUT (100% LOAD AND SAVE WITH NO NOISE ON THE MONITOR). SWITCH SELECTABLE BAUD RATE. ALSO INCLUDED ARE: AARDVARK 8K BARE BOARD PLENTY OF FREE SOFTWARE, ALL BACK ISSUES OF THE AARDVARK JOURNAL. ALL FOR ONLY $750
CALL GARY VOGEL (313) 288-5517

FOR SALE: OSI C2-4P (4K) IN "LIKE NEW" COND.;COMPLETE WITH ALL DOCUMENTATION AND SEND INQUIRIES TO: D. MONTALVO 215 E. 197TH ST #3J, NEW YORK, NY 10458

FOR SALE: 505 BOARD, CPU WITH FLOPPY INTERFACE AND CASSETTE INSTRUCTIONS. UPGRADE TO DISK AT HALF THE COST $150 OR BEST OFFER. CONTACT LES CAIN, 1319 N. 16TH GRAND JUNCTION, CO, 80501 OR (303)243-4536

FOR SALE: C1P MODEL 1 W/8K. RS232 INSTALLED ON BOARD. $300. PHONE J. HYLAND (703)938-7218

FOR SALE: C2-8K, 12" MONITOR, TAPE RECORDER, GAMES, BOOKS $425.00 PHONE: J. BAKKER (313) 669-8477, BELLEVILLE MICHIGAN

FOR SALE: C4P CASSETTE COMPUTER IN ORIGINAL CARTON. AUDIO AMPLIFIER AND A FEW PROGRAMS INCLUDED. WILL SELL WITH OLD B&W MONITOR $625 OR W/O $575, PHONE: DAVID KOCH (216) 492-4461

FOR SALE: USED CASSETTE TAPES C-10 AND C-20 (NO CHOICE) $.35 EACH. NO REFUNDS. CONTACT AARDVARK.

FOR SALE: OSI 573 BOARD (EPROM BURNER) ASSEMBLED, USED. FOR 48-PIN BACKPANE (C2/4/8P). SOFTWARE ON 5" DISK. $100.00 CONTACT AARDVARK.

WARNING WHEN PHONING IN ORDERS AFTER 4:00 P.M., THERE IS NO GUARANTEE IT WILL BE PROCESSED OR EVEN RECEIVED. SOMETIMES THE ANSWERING MACHINE DOES NOT RECEIVE THE FULL MESSAGE, AND SOMETIMES WE CANNOT READ RODGER'S WRITING! SO, PLEASE CALL BETWEEN THE HOURS OF 8:00 A.M. AND 4:00 P.M.. TO INSURE CORRECT ORDER PLACEMENT. THANK YOU JUDY & CYNDI.

Judy & Cyndi (313) 669-3110
EASTERN STANDARD TIME

AARDVARK
TECHNICAL SERVICES
2352 South Commerce
Walled Lake, MI 48088