the AARDVARK JOURNAL

AUGUST 1981 VOL 2, NO.3

IN THIS ISSUE

The theme of this issue is goulash. We have included all the neat things that didn't fit in the theme issue.

We started off with the rest of the article on Adventure Writing. We've got two excellent articles on disk modification for the C1P an since we couldn't decide which one to publish, we included both of them. Dave VanHorn also sent us an excellent article on adding Standby Power to 5V systems. Stanley Windes contributed an excellent article on adding the AY3-8910 programmable Sound Generator and True analog Joysticks to the C1P. I should point out that with very slight modification that hardware will also work on C4/8 models.

Dave Edson, the Machine Code wiz kid who wrote Interceptor, Surface Attack, Collide and Monster Maze, sent us some tips on recovering programs when you crash the assembler. We have a couple of C1 programs and tips on Relocating Basic programs, Random numbers, Modifying the C4 break key and other stuff. For the beginner, Mark Guzdial has written a nice article on Line Pointers in BASIC.

ADVENTURE WRITING

Refer to Listing in Vol.2, #2

It is also easy to inventory what the player has. Line 520 to 540 do the inventory. They simply scan the entire list of objects and if the location of the object is zero they print the object's name. That's an often used and special command to do a special search for the word INV meaning inventory in line 230 and jump directly to the 520 subroutine if it is found. We can now wander around in our own little universe and pick up and drop things that we find. Once we are that far, everything else is simply a matter of adding detail after nit-picking detail and logic trick after logic trick to present the problems. For instance, lines 130 through 190 are executed everytime a command is entered. That's because they concern problems which may or may not exist on each turn. We have on the D.S. a flashlight with a limited life. We set a flag (F3=1) whenever the flashlight is on. That adds up the number of times that the light is on and makes it equal to FT. If the flashlight has been on for more than 25 turns we tell you that it is getting dim and if it is on more than 33 then we turn it off again by making F3 equal zero. The thing is that we have to check on each turn to make sure that you can actually see in that turn. We also check to see if you are carrying a bomb or carrying water because those would effect what would happen on the turn no matter what else you did. So, we check flag F4 to see if you have water and C8 to see if you are carryng the bomb. In lines 200 - 230 we expand the vocabulary of the D.S. by assigning synomins. For instance, if the player has put in GET, you then make A$="TA" for take. We found out writing later ADVENTURES that this was less than perfect as it was a lot more memory efficient to simply make W$ longer and add more GOTO's to the statement in line 300.

The next topic is how to make things magically appear and disappear into cabinets, crevases, lockers, and assorted dark and creepy places. If you look at the lines from 730 to 740 in the listing, which hold all of the locations of various objects, you will notice that a number of objects are in locations higher than 18. In spite of the fact that 18 is the highest location in the D.S. That's the secret to the magic appearance and disappearance when you say OPEN LOCKER or put something in a locker. The verb OPEN is handled by the routine in 1240 - 1340. The first few lines handle magic appearance and disappearance of objects. For instance, line 1250 checks to see if the player is in the same location as the locker and then checks to see if the second word was LOCKER and then scans through the entire array of 44 objects. Any object which is shown to be in location 29 locker and therefore not visable to the player is reassigned to be in location 8 where the player is. It doesn't take any additional space to create such hidden crevases as there doesn't need to be an element 29 in the location array and the implied three dimensional quality of the program is greatly inhanced by the use of such techniques.

Line 1300 demonstrates another one of the great advantages of this particular scheme of mapping an environment. It is easy to open up new pathways by simply changing an element in the array of movements. Line 1300 shows how this is done in D.S. There is a door which can only be opened from one side. Line 1300 checks first to see if you said OPEN DOOR and then if you did and you are in location 14 it changes the second location in the array for location 14 to a 12 rather than a 0, changes the name of the door to OPEN DOOR, and opens the corridor from location 12 to location 14 by changing that element in the array. The technique, therefore, allows for easy modification of the adventure map during the play. It is also handy for one way passages which add a lot to the joy and bewilderment of the adventurer. There are passageways that can be closed behind them by simply changing a former room number to a zero in the movement array.

Most of the other supposedly complex logic of an adventure is actually just a simple setting of flags and checking for locations and words. You have to get the idea of flags definitively in mind before you can write this type of program. There are a number of them used in D.S. For the purposes of this article we'll define a flag as being a variable which has two values - on and o Though they are not used often in D.S. as it was one of my earlier attempts, there is a neat construction in BASIC which allows you to check to see if a flag has any value other than zero. The phrase "IF T THEN XXX" will come up as true if T is anything at all and will come up false only if T equals zero. Therefore, if you have a number of critical things an adventurer must do before he can do other things, you simply set a flag each time he completes one step of the change. As an example, the first flag you encounter reading through the D.S. ADVENTURE is in line 150. The flag F3 is the flag for the flashlight being on or off. Whenever the player says LIGHT FLASHLIGHT, we make F3 equal 1, we then store the number of uses of the flashling in FT. When the player unlights the flash, it is time to make F3 equal zero. Therefore if we are in a dark area of the ship, and want to find out if we can print the description of the surroundings, we look at F3 and if it is 1 we print what is there to be seen.

The magic of transmutation is another interesting aspect of adventuring. The adventurer gathers up flour, eggs, milk and a bowl and - POOF - everything disappears and is transformed into a cake. There are a number of methods of doing that. One is to simply change the names of the objects as is done in line 1500 in D.S. In line 1500, the box has just been smashed, something teleported into the room and the description of the box has been changed to SPLINTERS. That only works, of course, if you don't want to pick up the box anymore as the word SPLINTERS is not in the long object string which the program checks for object words. There is a second technique which is usable if you have the space for it and which is a little more versatile. What you do is to add the desired object such as CAKE to the list of objects when you make up the initial listing and then when you want to make a transmution you simply switch the location of the objects. The bowl, flowr, milk and whatever change to a location off screen and the location of the cake is changed to on screen and PRESTO!! it suddenly appears.

For other things it is usually even simplier and I am not going to try to go over the entire program line for line, but in general most problems in an ADVENTURE are a question of do you have the right object to carry out this command? For that you simply check to see if they are in the right location by checking the number of the room, check the location of the objects he has to have by seeing if they are all zeros (on him) and then either complete the command or ignore it.

In general, if you want to look at individual verbs in the D.S. listing, simply look at W$ which appears in line 690. Count over by twos to the first two letters of the verb you want to examine and look at the "ONFGOTO" statement in line 300. That will give you the location of the subroutines that service that verb.

That should get most people off the ground in being able to write simple reality simulation ADVENTURES. We could go into more detail, but that would spoil the fun for anyone who was writing one.

There is one more topic that we need to cover before we close the article. It's a lot easier to code an ADVENTURE, that is to write the BASIC code, than it is to write an ADVENTURE. Coming up with the synario, the plot, the problems, and the map is much more work than the coding and much more important. There are a lot of ADVENTURES coming on the market and some of them are extremely good but a lot of them make some pretty definite and basic mistakes that we can warn new ADVENTURE writers about. First of all, after the first half hour of playing an ADVENTURE, most people do not care about long description. It's great the first time you see a description that begins with 'YOU ARE ON A TALL AND WINDY HILL LOOKING DOWN AT RAIN SPATTERED ROCKS. THE SKY ABOVE YOU IS GLOOMY...etc etc., etc." Unfortunately, during an ADVENTURE that description will probably be read two or three hundred times and after the second time you interpert it as being "MOUNTAINTOP" no matter what the long description is. In general, people prefer many more locations, many more objects and more complex logic to having lots of long pretty descriptions.

The manner in which the problem is solved is probably the most crucial part of the game. None of the ADVENTURES which we publish have any random numbers in them. An ADVENTURE is a thinking type problem. There is no luck involved, it appeals more to chess players than gamblers. That means a number of things. The first thing it means is that the problems must be difficult to handle but reasonably solvable. The adventurer once he sees the solution to the problem and gets through should slap himself on the forehead and say, "My God, why didn't I think of that first." As an example of a very poor ADVENTURE construct, there is a routine in the original ADVENTURE written on a DEC system where in the adventurer is trapped in a corridor by a cyclops. There is no way past the cyclops and no way back. The only way out is to whisper "ULYSSES". At that point the cyclops gets scared, runs through the wall, making a big hole which you follow. The real problem with that particular scenario is that no one in his right mind is ever going to solve that problem. No matter how cute and original the programmer was, that makes very poor gaming. Therefore, the problems presented must be carefully chosen. They must be difficult but solvable, and more importantly, rationally solvable. The ADVENTURE writer is not trying to prove how cute he can be but to present an interesting problem to the adventurer.

With that in mind, before the first line of an adventure is written there should be a complete map of locations the ADVENTURE will take place in and a movement chart and a list of problems that are going to be presented and a list of the objects that are going to be used.

It's kind of like writing a book rather than writing a program. You have to spend a lot of time on that plot. With that in mind every body go out and have fun making your own little universe to amaze, befuddle and astound your friends.

LINE POINTERS IN OSI BASIC-IN-ROM
BY MARK GUZDIAL

In the internal format of OSI Basic-in-Rom, the first three bytes of each program line consist of an initial null (a zero) and a two-byte pointer to the memory address where the next line begins. In this article, I'd like to discuss what this pointer does and where it's used.

The pointer is used for getting from one line to the next more quickly than by wading through each byte looking for the initial null. The pointer to the second line in the program is at $0301, $0302 (Dec. 769,770). Let's say they contain a $10 and a $03, respectively. Since line pointers use the 6502 format of least significant byte (LSB) followed by the most significant byte (MSB) of an address, this means the second line begins at $0310. The third line can then be found at the address contained in $0310,$0311. These jumps are alot easier than just checking each byte between $0300 and $0310 for the initial null of the second line.

This could make insertion of new lines really easy. If Basic just 'tacked' the new program line at the end, it could then just change pointers around to point to the new line. But it doesn't.

Every time that we type in a new line of code, Basic figures out which memory location it belong in, and then moves the whole program in memory-inserting our program line in its correct spot. Then Basic goes through and re-adjusts the pointers. Why go to all that trouble?

Because the main function os Basic, RUNning a Basic program, doesn't use the pointers to go from line-to-line (except for GOTOs and GOSUBs). It wades through the program, byte-by-byte, so that it can execute each byte that it comes to. This means the program lines must be in order.

The main function of the pointers is in the LIST command.

Let me try and prove this. Let's use the pointers as I describe them above, with ($0301,$0302) pointing to $0310. Let's also assume that those first two lines are numbered 10 and 20. If we change the pointer at ($0310,$0311) to point to some impossible address like $0300 (this is the easiest address to change to, just POKE $0310 with a 0). Until we get to a GOTO or GOSUB, the program will RUN correctly (if we use FOR...NEXTs, we'll notice even they run correctly-that's because they use memory addresses, not line numbers). But if we try and LIST the program, well, we'll have to warm start because the computer will start throwing garbage up on the screen and lock-up. If we type LIST 0-100, the program will list up to line 20 and stop.

The LIST-alone crashed because it followed the pointers which started leading to addresses that weren't correct. The LIST-by-line-number worked, somewhat, because it does checks on the pointers to see if they're at all reasonable and stops when they seem to be out of the range specified.

If we tried to insert a line, say 21, right now, we'd find a terrible mess on our hands, totally un-RUNable. That's because Basic would try the initial null of the first line and cause an even bigger mess when it tries to move the memory down to make room for the new line.

But if we tried to insert a line, say 5, not only would it work, but all the pointers would be restored to the right places. This is because we would be inserting BEFORE our now defective pointer.

Finally, if we don't do anything but replace whatever was in $0310 with the value was there before we nulled it out, we'll find our program to be functioning fine again with no problems.

So, line pointers in OSI Basic-in-ROM are used in LISTing and for finding where a new line should be inserted, but they are not used for getting from line-to-line during sequential Basic execution-line pointers are ignored during execution except for finding GOTO and GOSUB line numbers.

I hope that those users who work with Basic internals may find this of interest and use. Hopefully, this article will clear up any confusion over what those first two bytes after the null were doing there.

...MAKING YOUR DISK QUIET...
BY DAVE POMPEA

As I sit here the red light on mv disk drive is on but there is no sound and the head is not loaded. Your disk can be resting too.

When DOS wants to access the disk it has to load the head; seek the track; wait for the index hole; wait for the correct sector and then read the data. All these operations involve accessing the PIA & ACIA at the address COXX. We will use this COXX signal to tell us when to start the disk drive motor. We need to use the Phase 2 line to tell us that the COXX address is valid. So heres how it works:

When the Phase 2 line goes high and the not COXX line is low the one shot is triggered (and can be retriggered) and starts timing. This turns the drive motor on until it times out (retriggering restarts the timing). This motor on pulse starts another oneshot. This one blocks the index hole pulse so that the disk can get up to speed before DOS starts reading or writing.

HERES HOW TO MODIFY YOUR C1P:

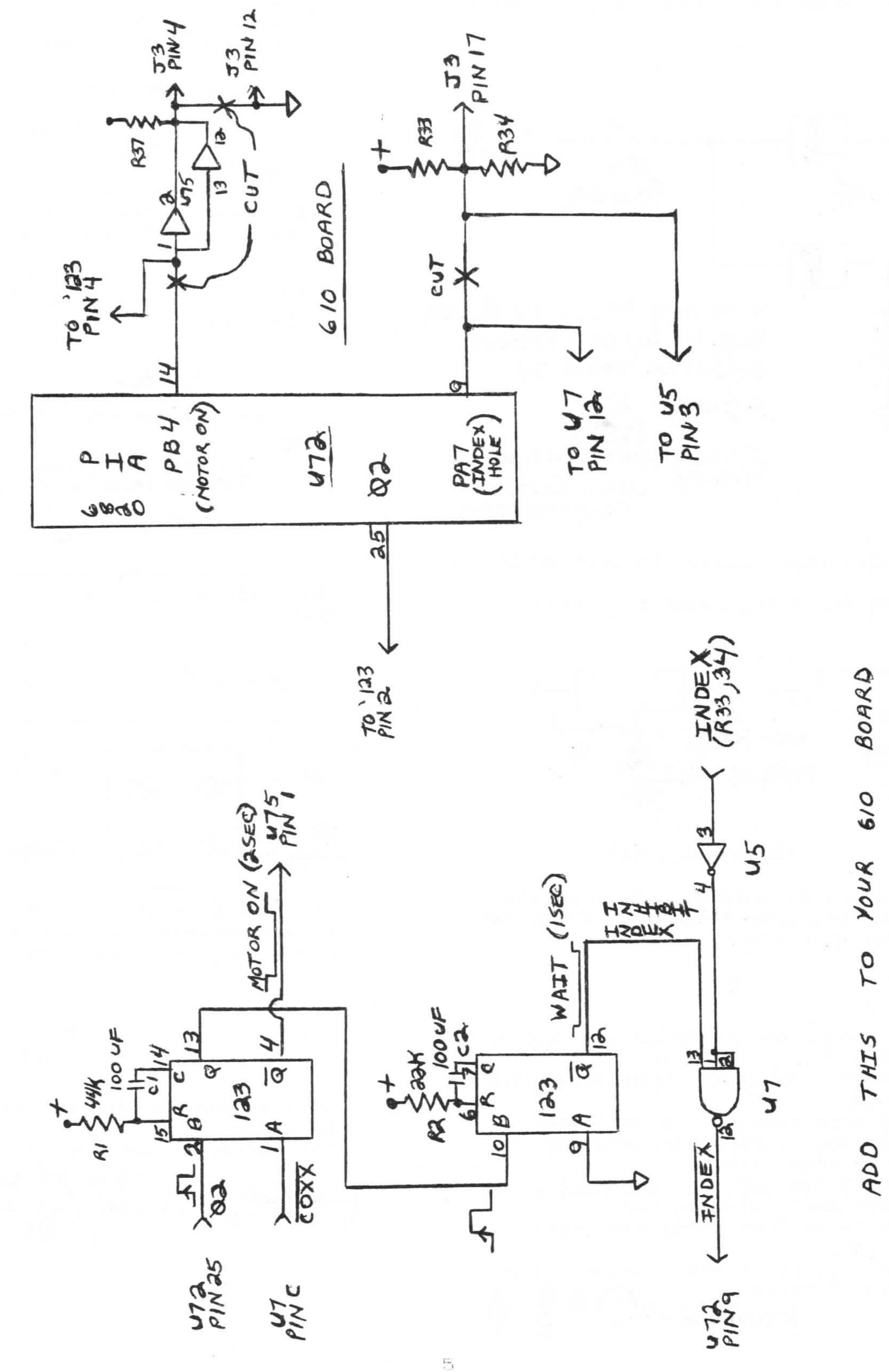
1. On a small perfboard mount and wire the 74123, R1&2, C1&2.
2. Open up your C1P to expose the bottom of the 610 board.
3. Using wire wrap wire (any thing bigger invites burned boards and fingers!) connect the spare 3-input nand gate U7 pins 1 & 2, to the spare inverter U5 Pin 4.
4. Pull off the disk interface board (it's the one with the ribbon cable and look at position 4, that the motor on signal, it's jumpered to position 12 (GND) to keep the motor running all the time. Cut that jumper.
5. Now take an XACTO knife and cut the trace from U72 pin 9 to the bias resistors.
6. On the top of the 610 board cut the trace that comes out of U72 pine 14 (motor on signal).
7. Connect the 74123 pin 2 to U72 pin 25 (the phase 2 line).
8. Connect the 74123 pin 1 to U7 pin 6 (the not COXX line).
9. Connect the 74123 pin 4 to U75 pin 1 (motor on line).
10. Connect the 74123 pin 12 to U7 pin 13 (index inhibit).
11. Connect U7 pin 12 to U72 pin 9 (index to PIA).
12. Connect U5 pin 3 to the index signal bias resistors, R33 & R34 (other side of U72 pin 9 trace cut).

Connect power & GND to the 74123 and that it, your drive motor will turn on when DOS wants to use the disk, wait 1 sec. to let the disk come up to speed (by blocking the index hole signal) and then turn off after about 2 seconds of non-use.

You can also change the jumpers inside your drive to load the head upon motor on (it's already set for load on select). You should keep all wires as short as possible and add A.1 UF bypass capacitor to the 74123 for noise reduction.

To change head load with drive select to load with motor on:
remove the pin 1-14 shunt and make the pin 7-8 shunt at the programming socket inside the drive.

C1P MINI FLOPPY DRIVE FIX

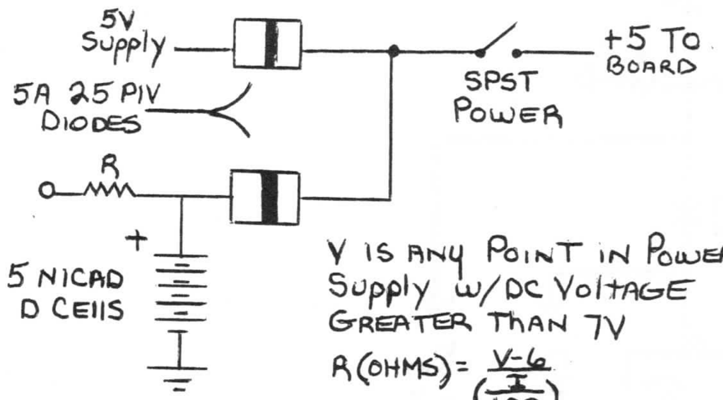


STANDBY POWER SUPPY FOR C1/C4P'S

BY DAVE VAN HORN

If using C1P or 4P with Delcotron T1-8C power supply use sheets A,B.

If using some other power supply - read on!

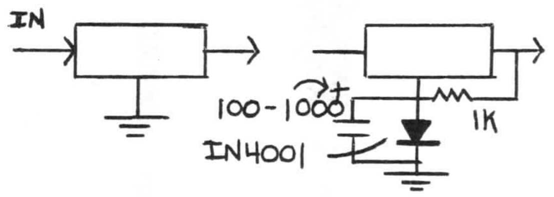


V IS ANY POINT IN POWER Supply w/DC VoltAGE GREATER ThAN 7V

$$R(\text{OHMS}) = \frac{V-6}{\left(\frac{I}{100}\right)}$$

I = CURRENT RATING OF NICADS (4AM Cells RECOMMENDED)

Re adjust power supply to provide 5V to Board.
If using fixed regulator try this:

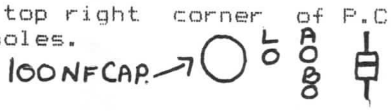


fools regulator

It raises output voltage by .7V. 1K resistor provides feedback. Capacitor provides soft start.

A

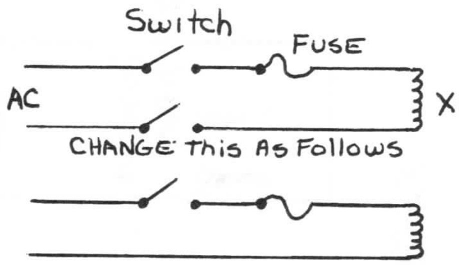
1. Set comp. on table upside down with KB facing you.
2. Remove bottom of case:remove board from case.
3. Make sure computer is unplugged.
4. Unsolder wires to transformer & remove PS from case.
5. Set PS on table & reconnect power cable but don't plug in yet.
6. Turn PS so that large capacitors face toward you.
7. Look at top right corner of P.C. board for 3 holes.



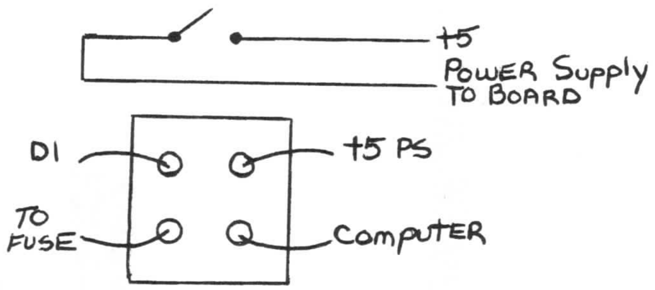
A,B, and C should be connected to same foil. Plug in PS and meter voltage at A, B or C. Should be in 11-14V.

8. If V <= 13.8 then use 10 nicad cells in series.
If V < 13.8 then use 1 12V gell cell.
I = current rating of standby battery.
If nicad then charge resistor $\frac{V-12.2}{\left(\frac{I}{100}\right)}\Omega$
Use nearest higher standard resistor.
If gell then charge resistor. $\frac{V-13.8}{\left(\frac{I}{100}\right)}\Omega$

9. Put 5A 25 pin diode in hole B with band facing you. Put resistor in Hole A.
If you want fast charge connect A high power 2 ener. (see B) in hole C with band(s) toward you.
Connect all free leads together & attach 18" 209A wire to battery or jack - of batter goes to case.



This allows you to turn of computer with switch.



The advantage of this method is that the output voltage does not change appreciably (< .01V) when on standby.

B
POWER 2 ENER.

This is made by connecting 1N4001 diodes in series.
The number is determined by

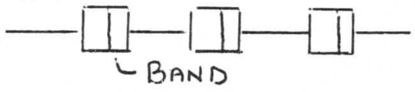
$$\left(\frac{V-12.2}{.7}\right) = N$$

IF N NOT INT. USE NEXT HIGHER INT. 4.2 = 5

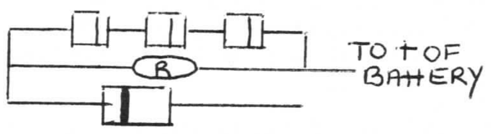
Example:

$$\frac{13.4 - 12.2}{.7} = 16:1$$

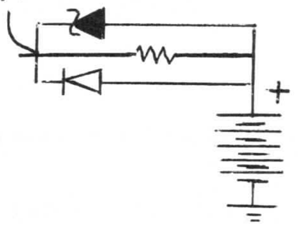
Connect Like this



Whole LKT is



V+ Schematic



BOB McCULLA, EVERETTE, WASHINGTON

I have just received Issue no.6 and enjoyed it quite a bit (as usual). I was a bit shocked to see you mention Assembly programming, knowing your abhorance of this mode. I really can't understand your feelings towards machine language but, different strokes etc. One way to make Assembly Programming a little easier is to keep a notebook, or disk file, of often used subroutines. Like, for instance: One hex byte to two ASCII converter, (OHTA) as it is know. A few others - (TAOH) or even 4 ASCII to 2 Hex bytes converter (FATH). I even have a standard input text buffer that I use all the time to input data from the keyboard. It can accept anything from 4 characters (ASCII) to a full screen full. The idea here is to make sure that I don't keep re-inventing my wheel. A little black or red, book of selected short programs or subroutines, can really help save some time. Why not ask your readers to contribute some of their short subs for the rest of us to file away. I would like to start this off with a little goodie that I like to use now and then.

This little sub is used to convert numbers that I enter through the terminal, (in decimal) to Hex code. This sub is used after the (TAOH) converter and allows the person using the program to input 94(dec) instead of 5E(hex). The original author of this sub is Mr. Jack Odell the best controls programmer in the world, or at the very least in my company. One more item, you might want to know that many useful subs are available for use in 65D, BASIC in ROM or even their monitor program 65A-65V-65F, etc. as long as it ends in $60(RTS). Use it!!

```
 10  0000              ;JACK ODELL'S SUPER
DEC TO BIN CONVERTER
 20 0000            ;CONVERT 8 BIT DECIMAL
BYTE IN ACCUMULATOR
 30 0000            ;TO HEX DEC EQUIVELENT
IN ACCUMULATOR AFTER
 40 0000              ;CONVERSION.  USES 2
TEMP. STORAGE LOCATIONS,
 50 0000            ;"SAVE 1" AND "BYTE"
 60  0000                  ;LATEST VERSION
03,31,1980
 70 0000            ;
 80 4000            *=$4000
 90 4000            ;
100  4000 48         DECBIN PHA          SAVE
BYTE
110 4001 290F                AND #$0F   GET
RID OF MS 1/2 BYTE
120 4003 8D1840               STA BYTE   LS
1/2 BYTE
130 4006 68                  PLA        GET
BACK SAVED BYTE
140 4007 29F0                AND #$F0   GET
RID OF LS 1/2 BYTE
150 4009 4A                LSR A   2
160 400A 8D1740            STA SAVE1
170 400D 4A                LSR A   4
180 400E 4A                LSR A   8
190 400F 18                CLC
200 4010 6D1740            ADC SAVE1
210 4013 6D1840            ADC BYTE
220 4016 60                RTS
230 4017 00         SAVE1   .BYTE 00
240 4018 00         BYTE    .BYTE 00
250 4019            .END
  24 BYTES
```

CURT BRODERICK, CLINTON, MASSACHUSETTES

Here is my program. It is for the C1P. The one thing that is unique, is the timer (I think!). One round of the clock and the game is over. Long play is two times as long as short play. The object of the game is to kill a high percentage of the Wampi (plural of Wampus). The only control keys in the game are the 1 key, 2 key, and the 7 key. By using the 1 and 2 key; 1 is to go left, 2 is to go right, you position yourself under the evasive Wampus (CHR$(4)). Once under it, hit the fire (7) key and see if you hit it. Score is shown at the end of each game.

FOR SALE
BASE 2 PRINTER, 64 TO 132 CHARACTERS PER LINE. 500 CHARACTER BUFFER, BI-DIRECTIONAL, MULTIPLE CHARACTER SETS AND FULL GRAPHICS. $450.00 PHONE CHUCK SCOTT (313) 528-0088 OR WRITE 3216 BELLE COURT, ROYAL OAK, MI 48072

For C1P

```
 5 :::::::::::::::::::::
 10 REM CURTIS BRODERICK
 15 REM C1P - KILL A WAMPUS
 40 REM
 45 ::::::::::::::::::::
 55 GOSUB1200:GOSUB1100
 60 CL=120:CT=125:PRINT
 65 PRINT"(1) LONG PLAY":INPUT"(2) SHORT
PLAY";I
 70 CL=120:CT=125:PRINT:IFI=1THENCT=250
 75 PRINT"(1) EXPERT"
 80 PRINT"(2) INTERMEDIATE"
 85 INPUT"(3) BEGINNER";I
 90 IFI=1THENH=17:SH=7
 95 IFI=2THENH=26:SH=5
 100 IFI=3THENH=45:SH=3
 105 IFI<1ORI>3THENPRINT:GOTO75
 110 GOSUB1200:GOSUB900
 115 C=54159+INT(RND(99)*24+1)
 120 POKE530,1:K=57088:A=248:POKEK,127
 125 X=54159:POKEX,A
 130 IFPEEK(K)=127THEN190
 135 IFPEEK(K)=191THEN200
 140 IFPEEK(K)=253THEN400
 145 IFP>1THENP=P+1
 150 CL=CL+1:GOSUB750:GOTO130
 190 POKEX,32:X=X-1:POKEX,A:GOSUB300:GOT
O130
 200 POKEX,32:X=X+1:POKEX,A:GOSUB300:GOT
O130
 300 IFP=0THENW=53571+INT(RND(99)*24+1)
 302 IFP=0ANDABS(C-W)<SHTHEN300
 305 C=W
 307 POKEW,4
 310 P=P+1
 315 CL=CL+1
 317 GOSUB750
 320 IFP>HTHEN350
 330 RETURN
 350 POKEW,32:P=0:Z=0:GOTO300
 400 POKEX,32
 405 X=X-32
 407 POKEX,A
 410 IFX=WTHEN 450
 415 IFX<53550THENPOKEX,32:S1=S1+1:GOTO1
20
 417 POKEX,32
 420 FORT=1TO4
 425 GOTO400
 450 POKEW-1,4:POKEW+1,4
 455 FORT=1TO1000:NEXT
 460 FORT=-1TO1:POKEW+T,32:NEXTT
 465 S=S+1:GOTO120
 500 GOSUB1200:PRINT"  TIME IS UP !!!"
 510 PRINT:PRINT"OF"S+S1"SHOTS YOU GOT"S
 520 L$="BEGINNER":M$="AN":IFI=1THENL$="
EXPERT"
 530 IFI=2THENL$="INTERMEDIATE"
 540 IFI=3THENM$="A"
 550 PRINT:PRINT"AS "M$" "L$
 560 PRINT:PRINT"YOU HIT"INT(S/(S+S1)*10
00)/10"% OF THEM
 570 FORT=1TO7:PRINT:NEXT:INPUT"PLAY AGA
IN";M$
 580 IFASC(M$)=89THENRUN
 590 PRINT:PRINT"BYE-":END
 750 IFCL=CTTHENCL=0:GOTO800
 760 RETURN
 800 POKEM+AM,32
 805 IFMC=5THENAM=32
 810 IFMC=9THENAM=33
 815 IFMC=13THENAM=1
 825 READ CA
 830 IFCA=1THEN500
 835 POKEM+AM,CA
 840 MC=MC+1
 845 RETURN
 900 M=53423
 910 POKEM-33,221:POKEM-30,222:POKEM+63,
220:POKEM+66,223
 930 POKEM-32,131:POKEM-31,131
 935 POKEM+64,132:POKEM+65,132
 940 POKEM-1,140:POKEM+31,140
 945 POKEM+2,139:POKEM+34,139
 950 POKEM-65,84:POKEM-64,105:POKEM-63,1
09:POKEM-62,101
 970 RETURN
 990 DATA143,199,190,198,128,195,189,201
,143
 995 DATA200,190,197,135,196,189,202,136
,1
 1000 RETURN
 1100 FORT=1TO8:PRINT:NEXT:G$="KILL A WA
MPUS"
 1110 FORT=5TO18:PRINTTAB(T);MID$(G$,T-4
,1):FORTI=1TO56:NEXTTI,T
 1120 FORT=1TO500:NEXT
 1130 FORT=54120TO53608STEP-32:POKET,16:
FORTI=1TO60:NEXT:POKET,32:NEXTT
 1140 FORT=53608TO54070STEP33:POKET,19:F
ORTI=1TO99:NEXT:POKET,32:NEXT
 1150 POKE54070,240:FORT=1TO999:NEXT
 1200 A=PEEK(129):B=PEEK(130):POKE129,0:
POKE130,212:S$=" ":FORC=1TO7
 1205 S$=S$+S$+" ":NEXT:POKE129,A:POKE13
0,B:RETURN
 1500 RETURN
```

DALE MEYERS, LANSING, MI 48910

The following programs are for making the random number generator have a random sequence (I believe) for BASIC-IN-ROM (disk BASIC should be able to be changed similarly). The first program (listing 1) pokes a machine language program (without having to go to the Monitor mode) that jumps the character parser into a routine on page 0 (listing 2) which checks for the RND keyword, $B4. If the keyword is found, the system decrements $F7 and if zero sets $F7 at $D9, decrements $F8 and puts the $F8 value in $D6, the third value of the random seed. The program adds 6 clock periods for every character and keyword examined by the BASIC interpreter. Of course the program resides entirely in page 0, so it doesn't conflict with the program or all those machine language programs in page 2 that you have stored away. Also the program doesn't use the USR vector (this is the only place that I know of that BASIC's operation can be changed while it is running).

The second program (listing 3) does the same thing as the first program but as a subroutine in BASIC, which if called only a few times in a real long program, before use of the random function might be faster than the machine language program. I have checked these programs and have found no sequence in over 400,000 uses of RND(8), after initialization of over 200,000 uses of the random function, and distribution is good.

LISTING 1

```
      10000
FORX=216TO238:READR:POKEX,R:NEXT
      10005  REM Listing 2 will now be in
page 0
      10010  POKE200,15:NEW:REM BASIC has
now been changed to go
            to listing 2
      10020
DATA201,180,208,16,198,247,208,10,169,217,133,247

      10030
DATA230,248,165,248,133,214,169,180,201,58,96
```

LISTING 2

```
      00C8....0F           Jumps to random
check
      00D8....CMP ##$B4  compare to RND
      00DA....BNE $00EC  branch if not
equal
      00DC....DEC $F7    decrease $F7
      00DE....BNE $00EC  branch  if  not
equal to zero
      00E0....LDA ##$D9  load A with $D9
      00E2....STA $F7    reset $F7
      00E4....DEC $F8    decrease $F8
      00E6....LDA  $F8    load A with the
value of $F8
      00E8....STA $D6    store at $D6
      00EA....LDA  ##$B4  reload A with
RND(B4)
      00EC....CMP ##$3A  reset flags
      00ED....RTS
```

LISTING 3

```
      10 X=PEEK(247):X=X-1:POKE247,X
      11
IFXTHENPOKE247,218:F8=F8+1+255*(F8=255):POKE214,
F8
      12 RETURN
```

KARL SEIDLER, WEST GERMANY

This is a little program which I've written. I found the idea to do it in a German computer magazine called "CHIP". There was an article about a game for the APPLE II which was called to write a similar game for the C1P or C4P. The only difficulty was that I didn't have a listing of the program. But I think the description was enough, and the game is good enough (I hope).

The object of the game is to let your snake eat as many meatballs as possible and to let it cover the longest possible way. You can steer it with the L-SHIFT, R-SHIFT, ESC-and CTRL keys (explained in the game). I added selectable difficulty levels (-0=slow, 0=fast) for the speed of the snake. Sometimes a new meatball will appear on the screen or one will disappear. If a new one appears, the number of the offered food will increase by one.

Everytime the snake has eaten a meatball, the computer will print out the covered way and how many percents of the offered food the snake has eaten. If you get more than 35% you're good. It is possible to get a poisoned meatball. Then the snake will split and appear at any other place of the screen, or you get the message: "GAME OVER - SNAKE DEAD".

In line 760 I set the variable "F". It gives the length and width of the area. If your system is a C4P or a modified C1P you can change it to 32 (now it's at 22).

```
 30 REM  RATTLESNAKE 40 REM  COPYRIGHT F
OR C1P 18.5.1981 BY:
 50 REM KARL-HDINZ SEIDLER            For C1P
 60 REM ELTZERSTRASSE 3
 70 REM 3176 MEINERSEN-OHOF
 80 REM W.-GERMANY
 90 REM TEL. (05372) 1213
 100 GOSUB760:KB=57088:V=-1:FO=50:D=INT(
RND(1)*4+1):ONDGOTO120,130,140
 110 D=32:GOTO150
 120 D=-32:GOTO150
 130 D=-1:GOTO150
 140 D=1
 150 C=53348:L=32:IFPEEK(KB)<129THENC=53
317:L=64:V=0
 160 F=22
 170 PRINTTAB(6)"RATTLESNAKE":PRINT:PRIN
T:PRINT:PRINT:PRINT
 180 INPUT" INSTRUCTIONS";A$:IFASC(A$)<>
78THENGOSUB640
 190 INPUT"DIFFICULTY (1-10)";H:H=H*10:G
OSUB760
 200 REM  SET UP AREA
 210 FORX=1TOF:POKEC+X,154:POKEC+X*L,157
:POKEC+F+1+X*L,156
 220 POKEC+(F+1)*L+X,155:NEXT:POKEC,165:
POKEC+X,167:POKEC+X*L,166
 230 POKEC+X*L+X,168:S=C+11*L+11
 240 FORX=1TOFO:POKEC+INT(RND(1)*22+1)*L
+INT(RND(1)*22+1),24:NEXT
 250 GOTO330
 260 REM CHECK IF KEY IS PRESSED; CHANGE
 DIRECTION
 270 K=PEEK(KB):IFNOTVTHENK=255-K
 280 IFK=252THEND=1:GOTO320
 290 IFK=255THEND=32:GOTO320
 300 IFK=250THEND=-1:GOTO320
 310 IFK=222THEND=-32
 320 S=S+D:IFPEEK(S)<>32THEN410
 330 POKES,226:POKES-D,187::WAY=WAY+1
 340 FORX=1TOH:NEXT:IFRND(1)>.6THEN270
 350 REM  SET OR EREASE MEATBALLS
 360 P=C+(RND(1)*F+1)*L+RND(1)*F+1:IFPEE
K(P)>32THEN270
 370 POKEP,24:FO=FO+1
 380 P=C+(RND(1)*F+1)*L+RND(1)*F+1:IFPEE
K(P)>24THEN270
 390 POKEP,32:GOTO270
 400 REM  SEE IF THERE IS ANY OBSTACLE
 410 IFPEEK(S)>32THEN540
 420 REM  EAT THE MEATBALL
 430 FORX=1TO50:POKES,62:POKES,25:NEXT:M
E=ME+1
 440 PRINTCHR$(13)"WAY";WAY;" %=";ME*100
/FO;
 450 IFRND(1)>.1THEN330
 460 REM  MEATBALL WAS POISONED!
 470 PRINTCHR$(13)SPC(22);
 480 PRINTCHR$(13)" POISON!!";:FORA=1TO1
500:NEXT
 490 D=1:PRINTCHR$(13)SPC(22);:S=C+(RND(
1)*F+1)*L+RND(1)*F+1
 500 REM  FIND A NEW PLACE FOR THE SNAKE
 510 IFPEEK(S+1)=32ANDPEEK(S+2)=32ANDPEE
K(S+3)=32THENS=S+D:GOTO330
 520 GOTO490
 530 REM  WALL - GAME IS OVER
 540 PRINTCHR$(13)"GAME OVER - SNAKE DEA
D";:FORA=1TO3000:NEXT
 550 PRINT:GOSUB760:PRINT"YOU HAVE KILLE
D YOUR    SNAKE!!"
 560 PRINT"IT HAS COVERED A WAY":PRINT"O
F";WAY;"METERS AND HAS"
 570 PRINT"EATEN";ME;"MEATBALLS
 580 PRINT"THIS ARE";ME*100/FO;"% OF
 590 PRINT"THE OFFERED FOOD
 600 FORX=0TO5:PRINT:NEXT
 610 INPUT"ANOTHER SNAKE";A$:IFASC(A$)<>
78THENWA=0:ME=0:FO=50:GOTO190
 620 GOSUB760:END
 630 REM  INSTRUCTIONS
 640 FORS=1TO6:PRINT:NEXT
 650 PRINT"YOUR SNAKE MUST EAT AS  MANY
MEATBALLS AS POS-
 660 PRINT"SIBLE. YOU CAN STEER IT WITH
THESE KEYS:":PRINT
 670 PRINT" ESC = UP":PRINT" CTRL = DOWN
 680 PRINT" L-SHIFT = LEFT":PRINT" R-SHI
FT = RIGHT":PRINT
 690 PRINT"TRY TO CATCH THE BALLS  WITHO
UT TOUCHING YOUR
 700 PRINT"SNAKE'S BODY OR THE WALL
 710 PRINT"BUT IF YOU GET A POI-   SENED
ONE THE SNAKE
 720 PRINT"WILL SPLITT ITSELF AND  APEAR
AGAIN ON ANOTHER
 730 PRINT"PLACE OF THEN SCREEN.":PRINT
 740 INPUT"  READY";A$:PRINT:PRINT:PRINT
:RETURN
 750 REM  CLEAR SCREEN
 760 ZY=PEEK(129):ZX=PEEK(130):POKE129,0
:POKE130,212:S$=" "
 770 FORX=1TO7:S$=S$+S$+" ":NEXT:POKE129
,ZY:POKE130,ZX:RETURN
 780 : D=DIRECTION; L=LINELENGTH; C=UPPE
R-LEFT CORNER; V=SYSTEM
 790 : S=POSITION OF SNAKE'S HEAD; WAY=W
AY COVERED IN METERS;
 800 : KB=KEYBOARD; K=PEEK(KEYBOARD);
 810 : MEAT=NUMBER OF MEATBALLS SNAKE HA
S EATEN
```

MOTOR CONTROL FOR THE C1P DISK DRIVE

BY JAY L. CROSS

After trying Edward Keating's head load modification described in last month's Journal, I decided there must be a way to control the head load soleniod and the drive motor on the C1P disk systems.

Some checking in the manual for the MBI B-51 drive (used by OSI) shows pin 16 on the card edge connector in the drive is called Unfortunately, OSI had hardwired this line to ground, so the motor runs continuously. This ground connection is on the small circuit card at the "610 board" end of the interface cable. What is needed is a way to switch this line from low to high and back when desired.

There are several ways to do this. The simplest is to put a switch in the line and manually turn the motor on and off. However, I like to do things under program control, and, with a little effort and one IC, this can be done also.

My thanks to C.W. Aigledinger who suggested (also in the April AARDVARK Journal) using address strobes decoded by the AARDVARK sound generator board. With slight modification, you can control your drive with his circuit. Of course, the first thing to do is rewire the address lines on the sound board to be compatale with the expansion connector on the 610 board instead of the 600 board for which it was originally set up. This is a very minor change, as some of the lines are the same on both the 600 and the 610.

With a 74LS74 wired as shown here, connect the "CLOCK" (pin 11) to an unused decoder output on the sound board (we'll use Y7 - pin 7 as an example). POKEing anything (less than 256) into address 40959 will change the output state of the flip-flop at pin 9. This same output will be connected to the drive's "MOTOR ON" line.

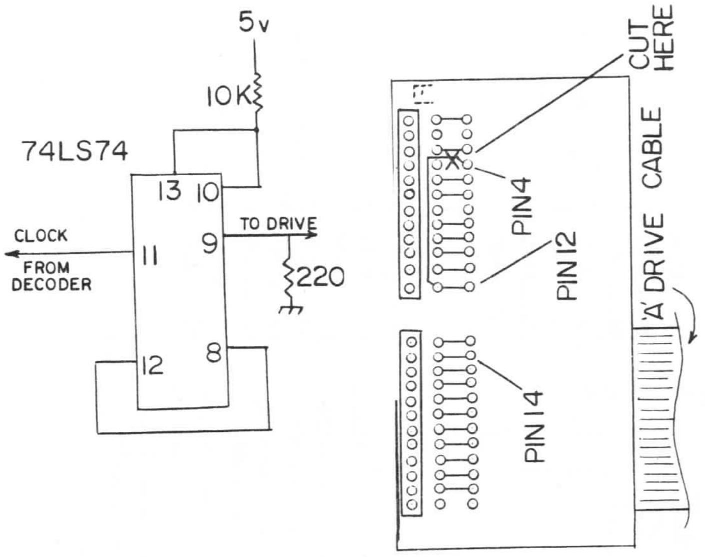
The easiest way to do this is to locate the circuit run connecting pin 4 and pin 12 on the small circuit board at the marked "OSI A-13 REV. B". Cut this run as shown in the sketch. Now solder a wire from this through-hole to the hole at pin 14 (OSI left 14, 15 and 16 unused).

On the 610 board solder a wire to pin 14 of the floppy cable connector. Pin 24 is at the edge of the board. Count backward from there. Connect this wire to pin 9 of the 74LS74, which can be located on a small piece of perf board or anywhere else desired.

If you used Y7 of the sound generator board's decode, typing will turn it back on. Putting this line into a program will also control the motor, but you must add a timing loop after this line to allow the motor speed to stabalize before reading or writing the disk.

One more thing and you are finished. If you remove the jumper plug at the upper right corner of the drive board (just below the edge connector) and rearrange it so that pin 2 is jumpered to 13 and pin 7 is jumpered to 8, then turning on the motor will load the head and turning the motor off will unload the head.

If you do not have the AARDVARK sound generator board, you can still do the mod using an unused address line on the 600 board. Pins 9 and 10 on U-20 (74LS138) are not used on the Model I Superboard II or C1P. Either or these could be connected to pin 11 of the added 74LS74. POKE 54272 for pin 10, and POKE 55296 for pin 9. The Model II version (with 48 X 12 display) makes use of these pins on U-20, so new board owners will have to look elsewhere.



SUPERBOARD SUPER SOUND AND OTHER GOODIES
BY STANLEY WINDES

I have made several additions to my Superboard which have greatly extended its usefullness and the fun I get out of it. Those of you who enjoy the hardware aspect of personal computers will be especially interested.

1. A terrific arcade-type sound generator.

2. A fast and easy to use Joy-Stick interface.

Lets start the discussion with the Sound Generator circuit. I use the G.I. AY3-8910 (Programmable Sound Generator PSG), the most versatile sound chip on the market. It was written up in the Dec. '80 issue of Kilobaud, so I won't discuss the chip's structure and programming, only how I am using it.

Figure one shows the AY3-8910 and its associated circuitry. I show buffering of the address lines which may not be necessary since the circuit adds only one additional LS input. I do use buffering by way of a 74LS241 in my particular set-up, because I added this circuit to an existing expansion board. I chose to locate the sound chip at memory locations 63232 thru 63235 (F700 through F703) because they were free and simple to decode with a single 74LS30. (see figure 3) The LM386 output amplifier provides plenty of volume even though it is operated from only a 5V supply. The clock input on Pin 22 of the PSG can use the standard (1mc) phase-two clock, but to be able to use the frequency constants from the data sheets, it should be connected instead to two times phase two clock from pin 14 of U30. The programming is well explained in the data sheets that come with the PSG. The proceedure to load a value into one of the PSG's registers is: first select a register (1 to 16) by POKEing a value into location 63235 (F703) then loading the data by POKEing to 63233 (F701). The following shows how simple it is to produce a ball-bounce sound and an explosion which I use in one of my games.

```
5  R=63235:D=63233:REM REG  AND  DATA
ADDRESS DEFINITION

1100  POKER,1:POKED,10:REM SUB TO  MAKE
BALL BOUNCE SOUND
1120 POKER,7:POKED,56
1130 POKER,8:POKED,16
1140 POKER,12:POKED,10
1150 POKER,13:POKED,0
1170 RETURN
1200  POKER,6:POKED,15:REM SUB  TO MAKE
EXPLOSION SOUND
1210 POKER,7:POKED,7
1220 POKER,8:POKED,16
1230 POKER,13:POKED,0
1240 RETURN
```

The Joystick input is shown in figure 2. The chip that does all the work is the AD7581 (Analog Devices Inc.) and is available for approximately 12-20 dollars, depending on type and quantity. There are 8 Analog inputs to this chip corresponding to 8 memory locations. The chip continuously up-dates the memory locations with a vaue from 0 to 255 scaled to input voltage for each Analog input. All 8 inputs are updated in 640 clock cycles; which, for the Superboard translates to approximately once each MS.

The address of the input locations are from 63264 to 63292 (F720 to F73C) in steps of four. (Again a hardware convenience) (see figure 3) therefore to find a Joystick position just PEEK at the address of the proper input channel.

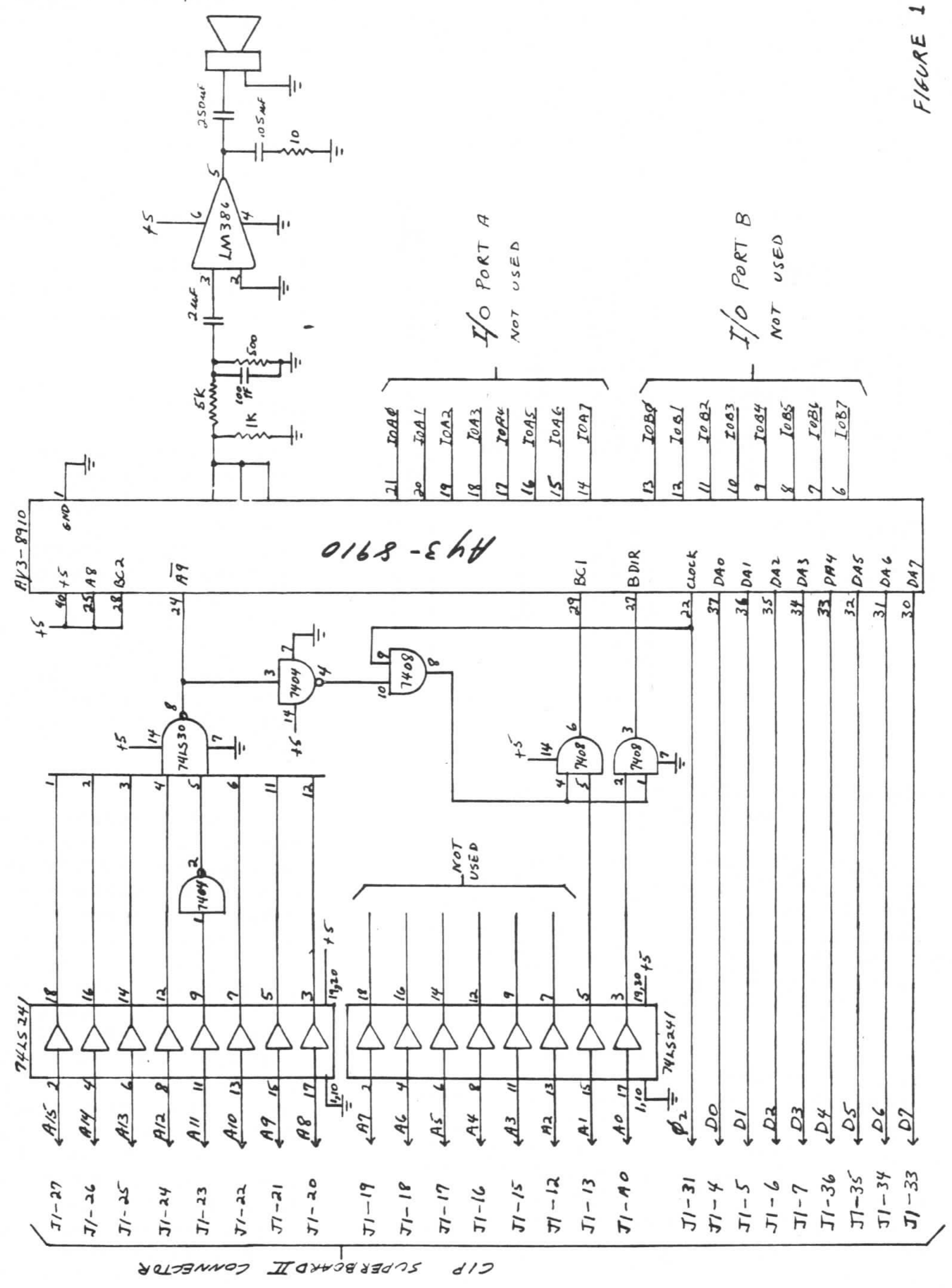
The following is the routine I used in a program like BREAKTHRU. The fire button is used to serve the ball.

```
100
GOSUB500:IFPEEK(63284)<200THEN100:REM
WAITING TO SERVE
500  IP=INT((PEEK(63288)/255)*19):REM
JOYSTICK OFFSET POSITION
501 POKEPP,32:REM ERASE OLD PADDLE
502 PP=53894+IP:REM NEW PADDLE POSITION
503 POKEPP,135:REM DISPLAY NEW PADDLE
```

The power requirements for this chip are +5V and -10V. The acceptable range of analog inputs is 0 to 10V. The wiring diagram for the Joysticks is in figure 4. Note that it is possible to connect other analog devices to the inputs if you wish.

One last word about the Interdependenies of the circuits presented: they were separated to emphasize the fact that all need not be implemented at once. Figure 1, the PSG, can be implemented alone as shown. When the analog input is added, as per figure two, then connections to Pin 25 of the PSG change as shown. This turns off the PSG when the analog input circuit is addressed.

GOOD LUCK TO ALL



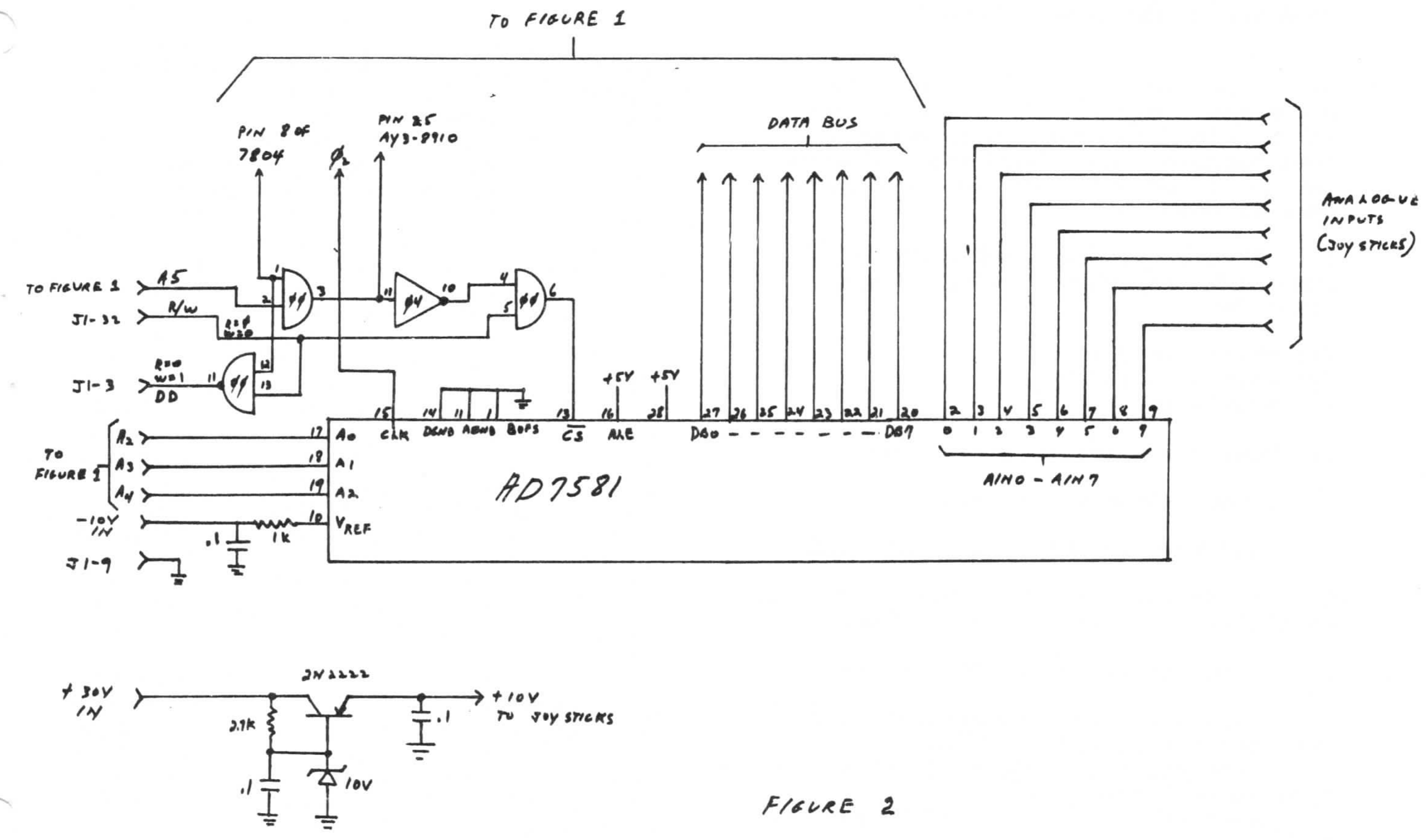


FIGURE 2

FIGURE 3

SOUND

A0 BDIR	A1 BCI	ADDRESS	INSTRUCTION
0	0	F700 63232	POKE 63232,0
1	0	F701 63233	POKE 63233,Data
0	1	F702 63234	PEEK (Read Purt)
1	1	F703 63235	POKE 63235,Reg Add.

JOYSTICKS

ANALOGUE INPUT	ADDRESS	INSTRUCTION
AIN 0	F720 63264	PEEK ADDRESS
AIN 1	F724 63268	
AIN 2	F728 63272	
AIN 3	F72C 63276	
AIN 4	F730 63280	
AIN 5	F734 63284	
AIN 6	F738 63288	
AIN 7	F73C 63292	

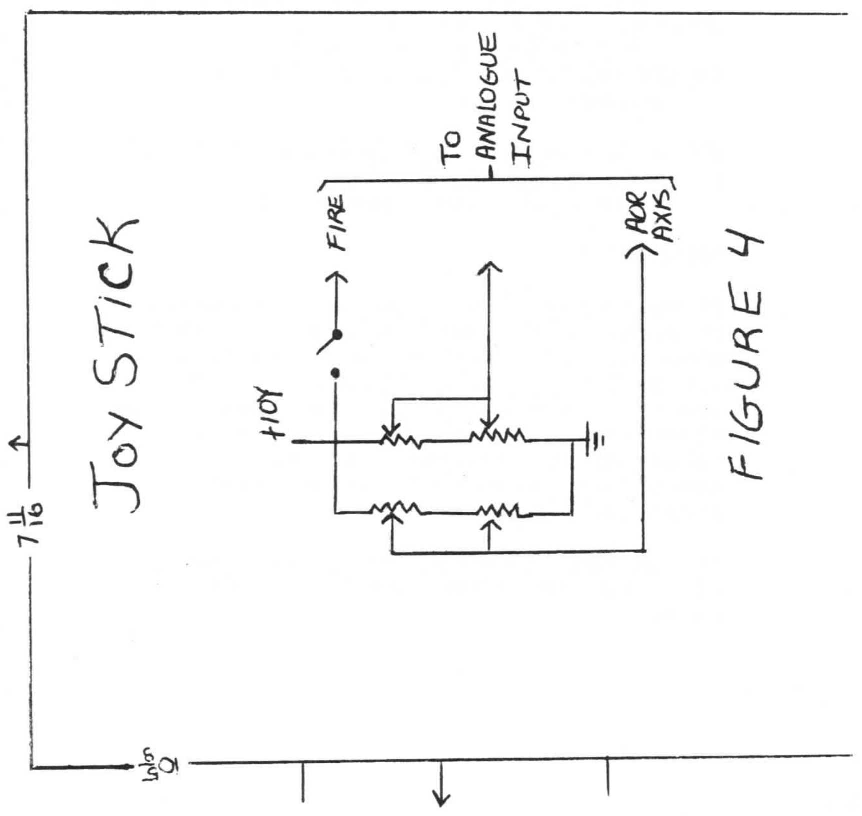


FIGURE 4

DAVE EDSON, MERIDIAN, IDAHO

As I am a serious machine code programmer myself, I realize the need for a method to save that program you lost because you crashed the assembler (which I do about 10 times per game!). Here is how-----

1. BEFORE RUNNING THE PROGRAM, WRITE DOWN THE FOLLOWING THREE MEMORY LOCATIONS:

(1) What's in $12CC
(2) What's in $12FE
(3) What's in $12FF

2. Now, crash the assembler (Ha-Ha)

3. DO NOT BREAK C WHEN YOU CRASH

4. Load the Assembler back into memory

5. Reset these three locations back to what they were before you crashed.

6. DO NOT INIZ. Type "P", return.

7. You will see more garbage than you ever thought possible. Then after about 15 seconds of this, your program will start, at about the 11th line. You lose lines 1-10.

8. Type "S", return, "P", return.

9. As the garbage is saving out, do not start the tape (Unless you like garbage). In some spots of the garbage, there will be nothing but nulls. This is normal-where the assembler filters cassette output.

10. Once you see your program, with the first 10 lines lost, press record, and save your program.

NOTE

If your crash was a loop out of control, it might have looped through your source program. If this is the case, you're out of luck--it's gone forever. This saver is meant mainly for when you do something like store variables in the assembler or assemble in the assembler-things that crash the assembler ONLY.

If you don't understand any of this, call me at (208) 888-4707. Ask for Dave.

BASIC BLOCK DELETE

This 82 byte machine language program allows C1P owners to delete any number of BASIC lines with a single command. A word of warning: this is a quick and dirty routine without error checking. Any deviation from the command format may produce unpredictable results. To delete a block of lines, type: first line, last line CTRL/. For example, to delete lines 100 to 400, type: 100,400 CTRL/ (CTRL / appears as a lower case letter o). When the deletion is done, the "OK" message will appear. The BBD may take 10 to 40 seconds to delete a large number of lines.

Basically, what the BBD does is to set the delete flag (very appropriately loc. $00), read the delete line numbers, and convert them to two-byte binary. BBD reads the first number, throws it away, reads the second number, stores it in loc. $30,31, goes back and rereads the first number, and stores it in loc. $11,12. All this so you can type the lower line number first, instead of last!

A null is stored in the start of the input buffer (loc. $13) to simulate a null line entry. Next, BBD jumps to the line delete routine at $A29D. This routine is not a subroutine: it goes to the start of the input routine (A$27D) when done. Luckily, the C1P input routine jumps through RAM and the delete routine can be intercepted on its way back to the immediate mode.

The delete flag is checked, and if it is set, the line number in loc. $11,12 is incremented. Next, the number in $11,12 is compared to the number in $30,31. If $11,12 is less than $30,31 the BBD jumps to the erase routine again. If $11,12 is greater than $30,31 the BBD is done. It turns off the delete flag and exits ti the immediate mode.

You can load the BBD with the monitor, as shown in a previous issue of the AARDVARK Journal. Once you have double-checked the code, patch BBD into the input routine with a POKE 536,34:POKE 537,2. Although BBD is shown at $0222, it can be stored in any part of RAM without modification. The only change would be in the two POKEs.

A very neat way to load the BBD would be to save it as a series of data lines. You could design a BASIC program to lower the HIMEM pointer by eighty-two bytes, POKE the BBD just above the new HIMEM and set the input vector to the start of BBD. By setting the delete flag and POKEing the right numbers into $11,12 and $30,31 you could erase the BBD loading program!

One last word: the BBD does not seem to be operational when the Extended Monitor is in memory even if you are not using the X-Monitor at the time (it took me about three hours to learn this).

```
10 0000                  10 GETCHR=$00BC
20 0000                  20 GETNUM=$A77F
30 0000                  30 ERASE=$A29D
40 0000                  40 INPUT=$FFBA
50 0222                  50         *=$0222
60 0222 C96F             60         CMP #$6F
70 0224 F00B             70         BEQ START
80 0226 48               80         PHA
90 0227 A500             90         LDA $00
100 0229 C94D           100         CMP #$4D
110 022B F033           110         BEQ DO
120 022D 68             120         PLA
130 022E 4CBAFF         130         JMP INPUT
140 0231 E600           140START  INC $00
150 0233 A912           150         LDA #$12
160 0235 85C3           160         STA $C3
170 0237 A900           170         LDA #0
180 0239 85C4           180         STA $C4
190 023B 20BC00         190         JSR GETCHR
200 023E 207FA7         200         JSR GETNUM
210 0241 20BC00         210         JSR GETCHR
220 0244 207FA7         220         JSR GETNUM
230 0247 A511           230         LDA $11
240 0249 8530           240         STA $30
250 024B A512           250         LDA $12
260 024D 8531           260         STA $31
270 024F A912           270         LDA #$12
280 0251 85C3           280         STA $C3
290 0253 20BC00         290         JSR GETCHR
300 0256 207FA7         300         JSR GETNUM
310 0259 A900           310         LDA #0
320 025B 8513           320         STA $13
330 025D 4C9DA2         330DELETE JMP ERASE
340 0260 E611           340 DO     INC $11
350 0262 D002           350         BME D1
360 0264 E612           360         INC $12
370 0266 A530           370 D1     LDA $30
380 0268 C511           380         CMP $!1
390 026A A531           390         LDA $31
400 026C E512           400         SBC $12
410 026E B0ED           410         BCS DELETE
420 0270 C600           420         DEC $00
430 0272 4C0000         430         JMP $0000
```

TODD BAILEY, GREENVILLE, OHIO

In my letter which you published in the April Journal I gave a fix for the MINOS game for the C1E. The value of H+6 given in line 97 may move the maze off the top of the screen on some monitors. A value of H+2 or 3 may work better. Some experimenting may be required for best results.

If you are using the High Speed Tape Load program on a system with the C1E Monitor ROM, the normal LOAD or SAVE function will not work. And it locks the system up requiring a "BREAK". This is because the SAVE and LOAD routines are at new locations on the C1E ROM. When the program sees a normal LOAD or SAVE it tries to jump to the OLD locations and comes up with garbage. To fix this change data line 30250 to read:

"30250 DATA12,76,112,254,165,20,201,
34,240,6,76,123,254,76,35,31"

I have also included a BASIC POKE program for Kerry Lourash's E.Z. Lister program. One is for systems using OSI'S Monitor ROM, the other is for systems using the C1E Monitor ROM.

```
LIST

10 REM-E.Z. LISTER FORC1P WITH OS1 ROM-
20 FORX=546TO603:READY:POKEX,Y:NEXTX
25 FORX=1TO30:PRINT:NEXTX
30  PRINT"POKE538,34:POKE539,2  TO USE
E.Z. LISTER"
35 PRINT:PRINT:PRINT
40 PRINT"POKE538,105:POKE539,255 TO  USE
NORMAL LISTER"
100  DATA141,2,2,72,138,72,152,72,
173,56,2,166,19
110 DATA224,153,208,6,160,8,132,
20,230,19,224,154,208
120 DATA17.201,13,208,13,198,20,
208,9,32,0,253,201
130 DATA32,208,10,198,19,104,168,
104,170,104,76,105,255
140 DATA162,254,154,76,116,162
150 NEW
OK
LIST

10 REM-E.Z. LISTER FORC1P WITH C1E ROM-
20 FORX=576TO633:READY:POKEX,Y:NEXTX
25 FORX=1TO30:PRINT:NEXTX
30 PRINT"POKE538,64:POKE539,2 TO USE
E.Z. LISTER"
35 PRINT:PRINT:PRINT
40  PRINT"POKE538,155:POKE539,255 TO USE
NORMAL LISTER"
100  DATA141,56,2,72,138,72,152,72,
173,56,2,166,19
110  DATA224,153,108,6,160,8,132,20,
230,19,224,154,208
120  DATA17,201,13,208,13,198,20,208,
9,32,70,251,201
130  DATA32,208,10,198,19,104,168,104,
170,104,76,155,255
140 DATA162,254,154,76,116,162
150 NEW
OK
```

WILLIAM BROOKS, SAN JOSE, CALIFORNIA

I noted in your last Aardvark Journal that you have successfully interfaced an Epson MX-80 printer to OSI. I bought a MX-80 at the West Coast Computer Fair to hook up to my C2-8P. I bought the #8141 RS232C serial interface board for the printer too. My C2-8P has the OSI #500 board in it with RS232C hooked up. However, so far the printer does not work. Could you please give me description of what you did to make the MX-80 work?

DEAR MR. BROOKS:

THE RIBBON CARTRIDGE IS INSTALLED. THE RS232 INTERFACE CARD IS INSTALLED AND THE DIP SWITCHES SET FOR OSI USE.

1,2,3-OFF, 4,5-ON, 6,7-OFF, 8-ON

YOU WILL NEED A "CABLE" TO HOOK THE PRINTER TO THE COMPUTER. THIS IS NOT PROVIDED BY EPSON. YOU WILL NEED 2-25 PIN-MOLE CONNECTORS AND 3 LENGTHS OF WIRE LONG ENOUGH TO REACH FROM YOUR COMPUTER TO THE PRINTER.

WIRE #1 GOES FROM PRINTER PIN #3 TO COMPUTER PIN #3.
WIRE #2 GOES FROM PRINTER PIN #7 TO COMPUTER PIN #7.
WIRE #3 GOES FROM COMPUTER PIN #5 TO PRINTER PIN #20.
ON COMPUTER CONNECTOR JUMPER 9 & 10 TOGETHER.

JOE ENNIS, NICEVILLE, FLORIDA

This program is to relocate a BASIC program anywhere in memory. It was originally written as part of a larger project. Namely writing a Forth Compiler for the OSI that will work with either cassette or disk systems and still be under 4K. Since I wrote the relocator, a girl who works in my ofice who has an OSI disk system needed a way to do overlays like she is used to with big main frames. (That is have an exective in BASIC and roll in the other BASIC programs from the disk and pass the Global variables down on the high end of page 2.) The BASIC programs are partitioned so that several can be in RAM memory at the same time and are kind of like big subroutines. This program is a little utility for for setting the partitions as well as used to locate my assembler in high memory so I can assemble Forth in low memory.

BASIC RELOCATOR PROGRAM.

```
 1 GOTO 10
 2 HI=INT(N/256):LO=N-256*HI:RETURN
10
AD=577:FORI=0TO27:READD:POKEAD+I,D:NEXT
20 POKE11,65:POKE12,2
30 PRINT:PRINT"CHANGE MEMORY
PARTITION?":
   PRINT"(ANYTHING  OTHER
THAN":PRINT"ZERO
   WILL CHANGE":INPUT"PARTITION)";N
40
IFN>OTHENGOSUB2:POKE133,LO:POKE134,HI:
   POKE129,LO:POKE130,HI
50 PRINT:PRINT"NEW START LOCATION FOR":
   INPUT"BASIC
ROUTINE";N:GOSUB2:POKEN,0:

LO=LO+1:IFLO>255THENHI=HI+1:LO=LO-256
60 POKE582,LO:POKE586,HI:LO=LO+2:IFLO>
   255THENHI=HI+1:LO=LO-256
70 POKE224,LO:POKE225,HI:POKE226,123:
   POKE227,0
80 X=USR(X)
90 DATA162,003,160,000,169,001,133,121
91 DATA169,003,133,122,165,224,145,226
92 DATA200,165,225,145,226,200,202,016
93 DATA243,076,000,000
```

OPERATORS INSTRUCTIONS
LOAD and RUN. Answer the prompts, answers must be in decimal after you INPUT is made to "NEW START LOCATION" the response of the program will be OK or ready depending on which Monitor ROM you have. Now type NEW. That is all. The change will have been made, and will stay in until the next Cold Reset (C after Break).

PROGRAMMER INSTRUCTIONS

Whats happening

At NEW START
POKE a zero

Set to NEW START+1
POKE 121(lo) and 122(hi), Start of BASIC Pointer

Set to NEW START+3
POKE 123(lo) and 124(hi), Start of Simple Variable Storage
POKE 125(lo) and 126(hi), Start of Array Variable Space
POKE 127(lo) and 128(hi), Start of Array Space

Set to New Top of Memory
POKE 129(10) and 130(hi), String Bottom (counts down)
POKE 131(lo) and 132(hi), String Top
POKE 133(lo) and 134(hi), Memory Size

ASSEMBLY LISTING OF MACHINE CODE OF BASIC RELOCATOR PROGRAM

```
$0241 A203 LDX#03
 0243 A000 LDY#00
 0245 A9XX LDA#LO
 0247 8579 STA$79 Z page
 0249 A9XX LDA#HI
 024B 857A STA$7A Z page
 024D A5E0 LDA$E0 Z page  LOOP
 024F 91E2 STA($E2),Y
 0251 C8   INY
 0252 A5E1 LDA$E1 Z page
 0254 91E2 STA($E2),Y
 0256 C8   INY
 0257 CA   DEX
 0258 10F3 BNE     LOOP
 025A 4C0000 JMP$0000
```

DUST COVER FOR THE C4P.

I made a dust cover for my C4P without sewing a stitch. Get a piece of heavy guae vinyl that has a cloth backing. Cut it to fit across the top of the computer, and between the Walnut side pieces (my cover is approximately 16-1/4" by 24" long). Not only is it functional and simple, but it lets you see the walnut wood's beauty. I roll it up in a very tight roll and store it in the space between the keyboard and the of the unit, and tuck under the fron when the computer is turned off. Why didn't OSI include a dust cover?

E-Z BREAK MOD STEPS FOR ANYBODY:

BOB SABIA.

1) For the C4P: turn the computer upside down on a clear work space and remove the six black screws on the bottom side with the correct size allen wrench. This step would have been easier for me if I had rested the top surface of the computer on a set of books, instead of on the walnut wood sides, but no real problem.

2) Carefully lift the bottom and sides off of the unit, but don't go far with it. You must remove the black ground wire from the bottom piece by unscrewing the bolt with a standard slot-head screw driver. Set the bottom/sides somewhere and concentrate on what is left.

3) I used SAMS Servicing Data book to verify the red wire we are going to cut and solder the cut ends to the switch. You shouldn't need any publications. The keyboard rests on whats called a 542 board. The board you first saw with all the chips on it is called the 502 board on the C4P (or the 505 board on the C4PMF). Look for two wires, one red and one black twisted around each other coming from the hidden side of the 542 board and going to a four pin plug about 3 inches away on the 502 board from where the BREAK key is, and closest to the "1" key.

4) We are going to unplug the red and black wires from the 502 board, but note that the red wire is to the left of the black wire as viewed from the front of the chassis. The four pin plug should read from left to right; pin #1 not connected, pin #2 red wire pin #3 black wire, and pin #4 not connected. Now, unplug the four-pin connector so as to prevent any damage to the chips while soldering and to make things easier to work on. Make sure you know exactly where you unplugged the wires from and which side the red wire goes on when you reconnect them.

5) I selected a miniature red button, momentary closed, spring loaded to the open-position switch (Radio Schack sells a mini-switch, one red/one black, in a package). Any similar switch will do, but the smaller the switch--the smaller the hole you will have to drill. I put my button switch one-half inch from my ESC key. Cut the red wire and strip the insulation back enough to solder the two ends to the switch terminals. I shaved the insulation first, wrapped the bare wire around both terminals, made my solder connections and then cut the wire to seperate the terminals. Whatever is easiest for you, just be neat and don't let the insulation shavings fall behind the board.

6) Drilling the hole for the switch must be done very neatly. All those metal chips and flakes could ruin your whole day. I used 3-inch wide masking tape and lightly taped over the metal to be drilled on (both sides to prevent marring the finish), overlap taped the keyboard, and then made a little box out of masking tape (sticky side toward where the hole will be) to catch the metal chips. Make sure the switch will reach where you want to put it, drill clear of the 542 board, put the keyboard right-side up, and do all drilling someplace other than where you will be reassembling it. You must use a punch to get a small size drill started, and then switch to a larger size drill. Take every precaution to catch every chip, flake, sliver, or particle of metal from the drill. I built sticky masking-tape fences around the drilling, dressed the hole very carefully and disposed of all newspaper and masking tape immediately. Take your time; this step is irreversible!

7) Return to your clean work area, install the switch in the drilled hold replug the four-pin plug (reverse step #4) hook the ground wire back up to the bottom of the case (reverse step #2), and reinstall the bottom cover (reverse step #1).

DENNIS SMITH, SALINA, KANSAS

HINT:

For new OSI Disk users; the OS65D manual talks about the CREATE utility and a need for a password to use that utility, but no where in the book does it say what that password is. After trying "OSI", "65D", I guessed at "PASS", and it worked!!

QUESTIONS

How do I change a password on OS65D?

After running a disk, pressing BREAK, then "D" my machine just quits. I can coldstart, warmstart, use the monitor or run PICO DOS, but none of my disks with 65D will run unless I turn the computer off and start from scratch (this includes "Nuclear Sub") Any ideas as to what causes this?

Dear Mr. Smith:

That silly password is simply contained in a line in the Bexec*. There is an input statement that says if the word that you input is not "PASS" then "NEW" or then "RUN", or something similar. I strongly recommend eliminating that line totally. Passwords are foolish unless you arre putting up a rather large scale turnkey operation for use by office girls.

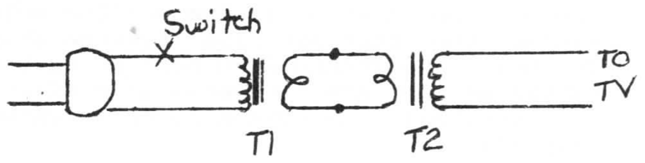
I have never heard of a system refusing to accept a "D" command after a break. I would suggest that you look for a hardware problem. There is, by the way, no practical way to Warm start or use the monitor to restart a 65D system.

JIM ARENS, ABBOTSFORD B.C., CANADA

Here's a way of showing 754 characters on a C1P. (29 lines of 26 characters.) And also get away with using a hot chassis T.V.

Purchase 2 Radio Shack (or similar) 25V 3Amp transformers. Hook the secondaries together which give you an isolation transformer and also drops the voltage to the TV, cutting down overscan.

Then (with AARDVARK C1S)
POKE546,4:POKE547,29:POKE548,2:POKE549,30.
On my Sanyo TV I get all those locations on the screen.



CARL M. KING, SARASOTA, FLORIDA

I have learned another trick from my OS-65D V.3 USER'S MANUAL Preliminary Copy, October, 1978, page 21. That is the loose leaf manual.

When you type EXIT, the DOS prompt tells you the number of tracks and gives you an A*. Then if you type RE BA (RETURN BASIC), it returns you to BASIC, and your program is intact.

In the C4P USERS MANUAL they tell you about EXIT but they fail to mention the RE BA command. If you type just BA it returns alright, but you lose your program.

How about that? The method for measuring the required number of tracks was there all the time, but I was reading the wrong book!

When writing a program to the disk, it is important to have a named region of disk large enough to contain the program. If for example, you have created a one-track file in anticipation of your program, and if it turns out that your program requires more than one track, and if you try to put the program into that file, then you will have the unhappy experience of fouling up your prepared track. The track becomes unusable.

What we need is a handy way to measure the length of a program in the work space before we commit it to the disk. We don't want to use more tracks than necessary just becase we are ignorant of the number of bytes.

Therefore, I have developed a short program routine that I tack onto the end of every program that I develop, as follows:

```
      50000 X=FRE(X):IFX<0THENX=X+65536
      50010 X=36214-X:PRINTX,X/2048
```

In practice then I type: RUN50000 <RETURN> , and the computer prints the number of bytes in my program and the number of tracks needed. Once I have measured the length of my program, I can remove the two steps, if I wish, and cut 50 bytes off the program.

A word of explanation about my routine: The FRE(X) command in the OS65D3 operating system returns a negative number whenever the free bytes exceed 32K. That is why the formula for X is so complicated. The number 36214 is the initial number of free bytes in my work space (less 7 bytes to store X).

Using this little routine has removed one element of doubt from my mind when preparing to put a program onto the disk.

JERRY MELE, BRUNSWICK, OHIO

This subroutine (30-50) will convert a number T in the dollar mode as a string. If you wish to round off in the same operation change A to 1.005.

```
      10 A=1.001
      30
DOLLAR=INT(T):CENTS=T-DO+A:CE$=STR$(CE)
      40 CE$=MID$(CE$,3,3):DO$=STR$(DO)
      50
DO$=RIGHT$(DO$,LEN(DO$)-1):DO$=DO$+CE$:RETURN
```

DO$ now contains T in dollar mode.

We have used this in the OS65U payroll since $R & L$ would not allow close spacing of numbers on the check stubs.

DAVID SUGAR, SKOKIE, ILLINOIS

Heres a neat list of POKEs that I have discovered while doing some dissasembly work. These POKEs are for users of OS65D Version 3.0/3.1, on a C1-PMF, and should make life a little easier for their users.

The first curious POKE is also a NEW screen clear (yes another one is possible). The following instruction typed in the immediate mode should demonstrate how it works:

```
POKE9803,33:PRINT:POKE9803,0
```

The last POKE is used to restore. Also anything may be printed to the screen between the two POKEs but will be cleared with the rest of the screen.

Location 9682 controls the cursor character, and may be changed from its current.

Location 9781 can be used to create a background on the screen, instead of having a blank. Best of all this background will stick with the system until it is changed. To demonstrate try this instruction:

```
POKE9781,161:FORI=1TO30:PRINT:NEXT
```

9800 is a most fascinating location. This is normally 32 and controls the amount the screen scrolls by. To get an idea how this one works, just try POKE9800,1 and hit the carriage return a few times!

9811 and 9791 may be used to specify scroll windows. 9811 points to the bottom of the screen (normally page 211) and 9791 points to the top page of the scroll window. It may be noted that the location the page starts at may be offset by POKEing a low order address in 9810 and 9790 as long as the value of the two match up.

Finally, there are the pairs 9666,9667,9676,9677 and 9684, 9685. These addresses are in low byte (6502) first format. They control where the Cursor, print to screen, and cursor "input" reference address are located. These normally point to the HEX address of D300*X. Relocating this address to match up to a relocated scroll window provides true versitile screen formatting.

For those more interested on where these POKEs came from and why they work, I suggest that you dissasemble the areas from HEX 2599 to 25F4 and the scroll routine at HEX2635+ with the use of the extended monitor, like I did.

FOR SALE
C4P CASSETTE COMPUTER W/RS232, CAT 103D MODEM.
SAMS PHOTOFAX, MICRO VERTER, AND COMPLETE
DOCUMENTATION INCLUDED. USED ONLY 30 MINUTES, LIKE NEW. PLUS $100 WORTH OF
AARDVARK SOFTWARE. $780.00
WRITE: SHANS BASITI
BOX 1605
GRAND PRAIRE, TX 75051

FOR SALE
C1P MODEL I WITH 8K.
RS232 INSTALLED ON BOARD. $300.00
PHONE J. HYLAND
(703) 938-7218

PATRICK TOWNSON, CHICAGO, ILLINOIS

For those of us beginners who like to "Putter Around" with machine language, an excellent new book is available called The 6502 Instructio Handbook. Published by SCELBI Computer Consulting, Inc., 20 Hurlbut Street, Elmwood, CT 06110, this 45 page booklet describes the entire instruction set in great detail. It includes several appendices, with the Mneumonics, Machine Codes, and Addressing Codes. A Hexidecimal to Decimal lookup table is a handy addition also. I bought my copy at a local computer shop for $4.50, but it could probably be ordered direct from the publisher. Much of the material included was written by Robert Findley for his Gourmet Guide and Cookbook, published in 1979.

IAN DAVIDSON, AUSTRALIA

DO YOUR RANDOM NUMBERS REPEAT TOO OFTEN?
Try this:

```
     1000
R=RND(8):FORW=3TO5:T(W)=T(W)+1
     1010 IFT(W)>WTHENT(W)=0:R=RND(8)
     1020 NEXT:RETURN
```

This will not repeat in less than 223,320 numbers. If it is still not good enough use FORW=3TO7 and they will not repeat in under 12 million numbers! (Note: A "repeat" is a sequence of 5 or more random numbers in the same sequence as previously).

WHAT'S NEW AT AARDVARK

Tiny Compiler is starting to grow up to be a Compiler. The FOR,NEXT,USR(X) have been installed. We also have optional routines that you can add for PRINT and INPUT. The bad news is that we are going to raise the price to $19.95 for either tape or disk. We are still hoping that someone will submit a routine to add a subscripted variable.

We have some neat new games again. Dave Edson has submitted SURFACE ATTACK (which looks an awful lot like Defender) and THIEF (which looks an awful lot like Ripoff). They go for $14.95 a piece. The team that put together the original Minos has provided us with CAGE. This is a cute little $8.95 game in BASIC and NIGHT DRIVER which is a hybrid BASIC/Assembler game which puts you on a twisting road at night and gives an excellent simulation of high speed driving. It is one of the authors best efforts and sells for $14.95.

For the first time we have some bad news from AARDVARK. The Business programs which we advertised over a month ago are still not being delivered. Besides a number of internal problems which delayed documentation, we also ran up against errors in the DOS which occasionally scramble files in full scale use. We have found the error in the OSI DOS and are now working on the correction.

The printer promises to have the Source Code for BASIC done by next week. This 100 page commented source listing is for Microsoft BASIC as it appears on the 8" 65D disk. It is virtually identical to the 5 1/4" BASIC and very similar to the BASIC-IN-ROM system. It will sell for $24.95. To forstall any questions on the topic -THE SOURCE IS NOT AVAILABLE IN MACHINE READABLE FORM -. Obviously we have a machine readable form and have assembled BASIC to check the source. However, our intention is to provide documentation to help those who have purchased legal copies of MICROSOFT BASIC and not to help pirates. We will therefor sell the documentation and listing but not at any time the machine readable and assembleable form.

AARDVARK
TECHNICAL SERVICES
2352 South Commerce
Walled Lake, MI 48088

BULK RATE
U. S. POSTAGE
PAID
WALLED LK.,MI.
Permit No. 5