P 11

the AARDVARK JOURNAL

april 1981 vol 2, no.1

IN THIS ISSUE

I decided to publish virtually the entire text of the GUIDE TO PROFESSIONAL PROGRAMMING. Some of it has appeared in the journal previously, but there is new material designed for those who want to write professionally or for the journal. NOTE- most of the GUIDE was written a long time ago and I have myself not followed all the advice that it gives. Please don't bother to write and point that out to me.

We have a good article from tbe British Users Group on machine code. I had just begun working on a similar article myself when this one came in. It is better that what I was doing so we published it. We'll try to get further articles from the BUG but cannot promise what will happen in the future.

THE GUIDE TO PROFESSIONAL PROGRAMMING

I am going to make a number of assumptions about the reader of the document. I will assume that you know the basics of programming. I will assume that you have read OSI's Graphics Manual and Aardvarks "Graphics in BASIC" and that you do not, therefore, need more information on how to move things around the display and how to program special function keys. I will therefore, dwell on how to make the programs run on both C1 and C2-C4 systems and on matters of professional style.

It is important that programs run on both types of video/keyboard systems. At this time the market for programs is about 70% 600 board and 30% 540 or 540B.

A program that runs on only one system, therefore, cuts out almost half of the potential market. It is additionally more expensive for us to handle and much more difficult to place with dealers as it doubles their stocking problems. We, therefore, try not to carry one system programs. We do do so only in exceptional cases. All the programs that I have written for OSI run on both systems and I am willing to convert well written programs-but the commissions are lower for programs I convert and I will only convert programs that are written with conversion in mind.

HOW TO DETECT THE SYSTEM We have tried a number of methods of detecting the system type automatically and have found that the most dependable and simplest method is to PEEK the keyboard. PEEK (57088) will return a number over 128 if the system is a 600 board (C1 or Superboard) and a number under 129 if the system is a 540 or 540B. A typical decision line would read:

```
 100VIDEO=600:IFPEEK (57088) <129  THEN
VIDEO=540
```

The program can then set up and adjust the variables for screen routine and keyboard for the system detected.

DUAL PURPOSE KEYBOARD ROUTINES

There are three kinds of routines you can use to do dual purpose keyboard detection. Which you use depends on the needs of the program.

if you are inputting from joystics or need a lot of keys detected, the best way is brute force. Store all the values for the keyboard in variable names or in an array and set up a couple of lines to change the values according to the system. For instance, if you are writing on a C4P and want to detect the

(1) key, you say-
P1=128:IFVID=600(REM THE C1P)THENP1=127
REM DETECTS TOP ROW EITHER SYS
POKE57088,P1:REM POKEKB WITH CORRECT
VALUE.
Fortunately, there is a simple system to convert the PEEKed values for the alternate system. As the keyboard values are inverted, you write for your system and then subtract the PEEKed value from 255 to get the alternate system - like this.
100P=PEEK(57088):IFVID=600THENP=255-P
REM CONVERTS FOR EITHER SYSTEM. The first system is therefor cumbersome but simple. Check the keyboard for which system you have. Store the system in a variable or flag. Change the POKE values if necessary. POKE the keyboard. Store the PEEK. Subtract the PEEK from 255 if other system.

The second routine works faster, takes less code and is preferable for systems where you can get by with 5 control keys. As explained in our graphics manual, the state of the control keys (shifts, cntrl, rpt, esc) are scanned continuously and stored in 57100. You can detect key presses by PEEKing that location without turning off the control C scan or POKEing to the keyboard. It, therefore, works fast and is simple to convert to dual usage. As with the preceding routine, you write the program to work on your system and then convert to dual usage by subtracting the value you PEEK from 255, with a line similar to the one above except that the POKE is not needed. That speeds up and siplifies things.

The third method is the simplest to use and requires no conversion. As given in our catalog, there is a subroutine in ROM at FE00 that can be addressed by a USR function. For those of you who do not have a copy of our catalog, I repeat.

10POKE11,0:POKE12,253 (sets up USR vector)

20X=USR(x):X=PEEK(531) (Gives ASCII value returned by keyboard) REM SEE AARDVARK CATALOG OR FORMER JOURNALS FOR DISK ROUTINE.
This routine STOPS FOR AN INPUT and cannot be used in real time programming.

DISPLAYS

Displays are important even if you are doing utility or business programs, and vital to games. With some care, your display routines can be written to run easily on both kinds of video displays, and may end up with a program that runs faster as a freebee benefit.

The goal is to end up with a program that can be converted by changing a single line of variable values. With care you can do a complex display that runs easily-Backgammon displays a complete board, moves pieces around, displays dice and messages, and changes from system to system by changing only two variables-the line length and the position of the corner of the board. Everything else is derived from those two values.

There should be no absolute values in any of your displays unless the values will work equally well for each system.

At a minimum you will probably need to assign three variables: Line Length T(L=64/L=32), Corner of the screen or some other common point to start both display, and Width (W-25/W=32 or64) if you plan to center things. You center by adding W/2 to a corner position.

After you have set up the variable values with the statement on the first page, you define the rest of the points you use in terms of the first variables. If you want a point 10 lines down from the corner on both systems, (and assuming you set up corner as one of your variables) you define the second point as :P1=CORNER+10*L. To draw a line across the screen 10 lines down, the code becomes
100FORX=P2toP2+POKEX,161:NEXT

Movement is done the same way. Horizontal movements work the same as a single system program. Add or subtract one. Vertical movements are done by adding or subtracting L rather than 63,65,31,and 33.

I am not going to give a lot of specific examples as our programs are full of them and as you are an expert programmer, you should be able to get the specific routines from them. I will add that you cannot trust the OSI map for 25x25 display on the C1. The display is closer to 25x30 on most systems and is not centered as in the graphics manual. I have included a map of the corners of both visible screens and the print positions for mixing print in graphics.

DON'T SCRIMP ON DISPLAYS

They are the most visible thing that separates us from 101 BASIC GAMES. They are what customizes a program and makes it marketable rather than useable. That also is true for business programs. They sell better if the displays are pleasing rather than just utilitarian.

Remember that your printed statements must be read on both 64 and 25 wide displays. If you put in timing loops too slow down the printed material, you will bore some users, insult others and lose the rest. If you have to divide the instructions into sections, end each section with an input. Try INPUT"READY FOR MORE"; A$. No matter what the user enters, the program will continue. It isn't necessary to check for a negative answer-they won't input anything until they have read the display.

Limit prints to 20-25 character lines at a time.

CONTROLS

One of the most common problems in programs sent in for evaluation is poor control layouts. This is a problem in business programs as well as games, as a well planned business program may contain such things as menu selection, screen scolls, and other special purpose keys.

I have a private and formerly secret belief that the first man to write computer games was a left handed sadist who hated the potential players and who was cross eyed to boot.

There is no other way to explain the huge numbers of games that now use the keys (1) (2) (3), and (4) for controls.- they had to be planned by a sadist. Consider the evidence.

(1) As a majority of the world is right handed, the solitary player has to play the game with his arm crossed in front of his body and his wrist bent back at an eventually painfull angle. If the victim tires or decides to rest his now aching limb, his palms will hit every key on the board, boot the disk, activate the break key, and scare the cat. It is a beautiful plan.

(2) The second piece of evidence is that there is no possible logical connection between the keys and the control functions. A typical game will have movement in 4-8 directions and one or more special functions like "fire", "stop","restart", etc. Thy typical thing is to make (1) move up (2) move right (3) move down (4) move left (5) fire (6) etc. There in normally no logical connection between the keys and the directions or functions, but it does have one "function". You will beat your friends all hollow until they learn the controls - assuming they stay that long.

Alternatives-

You have to keep two things in mind for professional and easy controls.

The first thing is that you have to have is some logical positional relationship between the controls. For instance, if you are going to move a cursor back and forth across the screen, use two keys side by side. The two shift keys are good for this. If you have only one person using the system, let him use both hands. He can do that. He isn't a politician. He uses both hands to drive cars and golf clubs and he can keep them separated easily. It also gives a much more spacious feeling to the game when both hands are used for contol.

Similarly, if you are moving something up and down use two keys like the (esc) and (ctrl) which are up and down from each other. I would also suggest that as some of the world is left handed, the space bar makes an easy to find control key for all users. - Just don't use in in conjuction with the (9) key. Keep in mind that the users have human sized hands the are jointed only in limited manners. They do not have thumbs on both sides of their hands or growing out of the wrists so keep the controls rationally close together.

The second major thing to keep in mind is that the controls must be visually distinct as the players or users will initially be searching visually between uses. That gives good reason for using shift keys (large and distinct) and the (,) and (.) keys (they have arrows printed on them). (Esc), (ctrl), (rpt) and similar keys have the advantage of being on the edge of the board and therefore easy to find. Remember that the user will be looking for the contols under stress and in a hurry. Make them easy to find, easy to understand, and easy to use.

GOOD IDEAS TO LOOK PROFESSIONAL

Do not use "Question (Y/N)" or (Yes=1/NO=2) constructs. OSI supports real string functions and failure to use them looks amateurish. Simply ask the question and if it is a Yes/No type answer, do not specify the format of the answer. You don't tell people you talk too how to answer. No one answers in French. Use the Left$ function to detect a "Y" or "N" and that will give you a yes or no unless the user is very weird.

Drop the formality. Just as you don't tell someone how to say yes, you don't say "wish me to" when you mean "want me to" and you don't say "May I..." when you mean "Can I...". People like to believe their systems think. Help them out by speaking colloquially.

Do not insult the user. The novelty of the computerized insult has worn off for all but the newest user. Do not use any end statments, prompts, or victory announcements that you wouldn't say in person.

Remember that people want to win games. They want a good fight, but reliable research shows that people will only continue with a game if they win a few at the start. No one either likes to admit that he needs to win or is willing to continue a game that is frustrating. Real success calls for user selectable levels of difficulty. At least one fairly easy version to get him started and then a harder version to keep the interest up. You can get the varying levels by changing timing, scoring structures, and the complexity of the task but note: the levels must have meaningfully different skill requirements. Just rushing a game or making it impossible to beat doesn't count.

Space is limited on a computer screen-particularly on the C1. If you have a border with the main function of keeping things from going off the screen, put the border off the screen. The first non-visible space will do as well and a user limited to a 25 wide area to play a game or read a display does not appreciate sacrificing 10% of it for borders.

THE AARDVARK RULES OF STYLE

If you are like most serious computerists, you have at least one book on BASIC STYLE. That puts you in a unique position to benefit yourself, me, our customers, and the energy crisis. BURN THAT BOOK! Most such books are written by business programmers who use MINI rather than MICRO systems and think that everyone has 48k memory or magazine editors who think that everyone has large tax deductible system at the office.

I read one book recently where the author actually used a full line-72 bytes-to give a simple remark. He suggested that, as the user is blind and cannot read the work REM, you preface the remark with lots of ****** to make sure that it is seen.

Our typical user has 4k of RAM and next most popular size system is 8k. It doesn't take many of those remarks to blow his entire program space. In fact, our typical user, despite his self image as a thinker, is really a man who bought an expensive version of an Atari video game. He is much more interested in features than in style. He would rather have two more targets, one more function, or another level of difficulty rather than a really pretty remark.

Our rules of style then, are based on giving the user the most possible use from his usually limited system.

RULE ONE---DO NOT GO TO REMs

Do not address branches to REM statements. We will document every program, but if room gets tight, the REMs will appear only on the printed version. Even if we don't remove the REMs, the user should have the option of doing so if he needs more room for additions. Therefore never have a GOTO, GOSUB, or THEN where the gone too is a REM. All REMS must be removable by simple erasure.

RULE TWO---USE LOTS OF REMs

This does not contradict rule one. We mean removable REMs. Neither customers or myself are willing to sort out an unexplained program. Explain every variable, identify each routine, and explain any unusual constructs. Don't be afraid to put in more REMs than the system will run with. If you have followed RULE ONE, we can erase them before we run the program and keep them in when we print it.

One REM should always be your name and address and phone number. Put it in as lines 0-5. Your envelope or letter could be lost or separated, but the tape will never be separated from program.

RULE THREE---DO NOT USE SPACES

Spaces do not make the program more readable for any computerist more than a few months into the hobby, and they can burn up as much as a third of your useable space. One extreme example was a "9k" program we received recently. When the author removed the spaces and remarks, the program ran in 4k and the market for it was tripled.

RULE FOUR-VARIABLE NAMES

I have taken a solemn oath that I will burn the next program that I recieve that has more than 9 variable names that start with X or Y. BASIC supports wonderfull variable names like BALL, SERVE, SIDE, TANK, CAR, SHIP, FENCE, SHELL, ROCKET, MONEY, CARD, SUIT, BALANCE, DAILY, MONTHLY, YEARLY, PLAYER and Author. Use them. They make the program readable and easier to understand. Leave X1,X2,X3........X9 and Y1,Y2,....Y adnausea home.

For the best balance between clarity and economy, use the full name first time the variable appears and use only the first two letters after that.

Aardvark customers are used to a few conventions in variable names. These are not required, but could make you program easieir to read.

X,Y,I are used for temporary counters, usually in loops. They store values that will not be reused later. If you need a fourth counter, use N SC or SCORE is used for score. C1 and C2 or E1 and E2 are used for the corners of the screen or edges of the display. The 1 designates upper left, the 1 lower right. TIME for time. AP for target positions. This does not follow rule 4 well. It comes from the fact that our first few games were shooting airplanes (AP). P is usually reserved for values from PEEK statements.(P=PEEK...) K is usually reserved for Keyboard (57088) P1=P9 are used for POKE values for keyboard routines. L is always line length. SHELL (SH) is used for any projectile position.

As it says above, these are very optional. Several of them started when the company was new and would be done different if started now. H X or Y that the value is temporary and that C1 is always a reference point in the upperleft corner of the screen.-unless I decided to break the rule that time-.

RULE FIVE---MULTIPLE LINE STATEMENTS

The worst possible line of code that I can think of is
100PRINT. The next three worst are...

```
100FORX=1TO30
110PRINT
120NEXT
```

Multiple statement lines have several important advantages. They run faster, load faster and save considerable memory on a small system. Normally a responsible programmer writes the rough draft in stretched format (separate lines for FOR and NEXT and so on) and then retypes the sections into compact form as each section is finalized and debugged. It's more work, but that is part of what "professional" means. For NEXT loops on the same line run faster as the system does not have to process line numbers on each loop.

A similar rule applies to short subroutines. Microsoft BASIC allows quite a bit of programming on a line after an IF statement. Do not branch out to a subroutine unless the business cannot be handled on the end of the line or you plan to address the subroutine from several different lines. Branching to a location, search the line number for the subroutine, decode the address from ASCII to Hex, do the subroutine, and branch back. In a business program it is annoyingly slow and in graphics game annoyingly slow and in high speed game it is disasterously slow.

When your program is submitted, it should be precise, clear, documented, labeled with your name and address, and compact.

RULE SIX THE FINAL ONE!

This is the one that every editor includes and always has ignored. Do not waste time and space telling how wonderful your program is, how it will sweep the world, "totally change the use of OSI computers" (a recent line an author used to explain a trivial utility program) or how much fun it is. I'm going to run the program. If it is good I'll probably catch on as my I.Q. is slightly in excess of 90 and I have several years in program marketing.

Use the time and space to explain features, controls, and constructs that may be unusual. If it has a subtle use that is not obvious, explain it briefly please. (No one ever explained anything briefly, but I keep hoping.)

I am not trying to imply that a little enthusiasm for your program is not proper. I tend to get wound up a little when I discuss my own work. However, lets try to keep it down to a few Hurrahs.

One more item has come up since this draft was done. We are receiving more programs now with USR machine code routines in them. Before publication, we need a commented dissassembly of the machine code routines. It need not be fancy, but we will need to explain to users what the funny numbers do.

The BASICs of machine-code
(reprinted by permission of British Users Group)

Over the past year we've had a fair number of comments on the lines of "All this machine-code information you publish is fine: but I don't even know where to start, so it's no use at all". Most people get on reasonably quickly with BASIC--it is, after all, designed to be a beginner's language--but without some basic idea of its concepts and terms, machine-code programming just can't make sense. The standard books on 65002 programming, such as Zaks' Programming the 6502 and Leventhal's excellent 6502 Assembly Language Programming, do assume that you know rather a lot about the subject before they start--or else, like Zaks, tend to confuse rather then help! So for the next few issues we'll be running this series on basic principles and practice of machine-code programming, to be based on, and linked with, OSI's BASIC. We'll assume no knowledge of machine-code at all--so bear with us if you do know the basics!-- but we will assume that you have a working knowledge of programming in BASIC, for games and the like rather than for complex mathematical juggling.

BASIC is described as a 'high-level language' for programming computers. But it's high-level only in the sense that you don't have to think much about where things are -- the language translates things like PRINT statements for you, for example, without you having to know what is being PRINTed to, where it is in the system memory, how it is accessed, or anything of the 'lower' level at all. But precisely because it handles these things for you, the language itself limits the flexibility with which you can use the system. It simplifies programming for some kinds of work--in BASIC's case, for basic mathematical functions--while making others, such as text-matching and fast graphics, well-high impossible. To do these elegantly and, above all, fast, they need to be done at a lower level--the machine-code level.

BASIC itself, of course, is made up of these lower-level machine-code instructions. In ROM BASIC, these can be seen by PEEKing the values stored in addresses 40960 to 49151 (we'll be dealing next issue with the meanings of the values you'll find there). Each BASIC instruction consists internally of a long trail of routines and subroutines--every POKE has to be checked such that its value doesn't exceed 255, or its address exceed 65535, for example--which is why clearing the screen by POKEing it with blanks is so agonizingly slow. A single POKE takes about 15 milliseconds; the equivalent LDA/STA machine-code instructions take about 15 microseconds--significant difference!

Working in machine-code, however, can be more than a little daunting--particularly if your only way of working with it is through OSI's next-to-useless monitor in the standard SYNMON, SYN600 and UK101 monitors. As you'll know, pressing "M" in response to the reset 'D/C/W/M?' prompt gives you a four digit mumber (an address) and a two digit number (the value at the address) in the top left of the screen--and that's your lot! OSI does publish a cassette version of their very good Extended Monitor (ExMon)--supplied as standard with UK101s, otherwise about £8 from most dealers. Otherwise we would recommend the CEGMOM monitor (not least because we wrote it!)--its machine-code monitor is designed to be compatible with ExMon, if somewhat less flexible, and it does have the advantage of being built-in to the ROM monitor rather than a five-minute load from cassette each time. We will deal with the use of the standard SYNMOM monitor, but for simplicity most of our examples will use the ExMon or CEGMON formats.

The difficulty with machine-code programming is that while the numbers in that four-digit section are relatively easy to understand--namely a system address--the two-digit numbers can mean anything, dependent upon context. The number 4C (and it is a number, a number in hexadecimal notation) could be a simple number (in other words 76 in decimal); part of a larger number stored in a number of bytes; an index, or part of a look-up table; an address, or part of one; an instruction (JMP--'jump' to the address defined by the following two bytes); the letter 'M' in the ASCII code; or all manner of other things. It all depends on its context; it all depends on where it is. Hence a couple of tools we'll describe in more detail next time: the assembler, which allows you to assemble machine-code instructions into a program with some semblance of clarity; and disassembler, which allows you to disentangle a string of hexadecimal machine-code values into something reasonably decipherable. Both are essential for serious machine-code values work; but for the smaller routines we'll be starting off with they aren't all that important. OSI's assembler, for all systems including the UK101, is available from most dealers at about £20; while there's a disassembler already built into ExMon, and we'll also give a listing for a disassembler in BASIC in the next part of this series.

The other catch is that almost all machine-code work is done in hexadecimal--counting in sixteens--rather than the decimal that BASIC uses. 'Hex' takes a little getting used to, but once you do know it, it will make the layout of your system a lot easier to understand. If you've ever wondered why your screen memory, for example, starts at such an odd number--53248--it's because in hex it is actually a nice round number: D000. Decimal numbers count from 0-9, then A-F (F is thus fifteen in decimal), and only then adds one to the next row.

The largest number a single eight-bit byte can handle is 255, or one less than 256,2 (2 'overflows', leaving an effective 'all-zeroes' in the byte). Describing a full eight-bit byte in decimal is tedious and misleading, especially as the 6502 processor has a separate BCD (binary coded decimal) 'decimal' mode of operation which doesn't quite appear as true decimal on the outside. Counting in whole bytes, in 256's, is hopelessly impractical; but half a byte (nicknamed a 'nibble'!) holds up to one less than 2, or up to 15 in decimal (the processor counts from 0-thus there are actually sixteen possible numbers in a nibble). We can thus represent the contents of any byte by two hexadecimal numbers:AF=175, 4C=76.

To reduce confusion, particularly when using an assembler, the convention for 6502 or 6800 processors is to prefix a hex value by a 'dollars' sign: 55 is fifty-five, while $55 is eighty-five (5x16, +5) in decimal. (Z-80 assemblers use a different convention: hex values have an 'H' suffix, thus 55H). Don't use the $ prefix when working with the machine-code monitors, though;they only know about hex, and will only get confused if you try to tell them otherwise!

Writing in machine-code is rather like writing in a very tightly limited BASIC. You have only three variables which you can use directly--the main registers A,X and Y--although you can operate on the rest of the memory with a very restricted range of functions. You have a very limited set of commands and functions--no large-scale functions like * or MID$, only the ability to copy or change the contents of the registers or of single memory locations. So programs have to be worked out in far more detail than they would in BASIC.

However, it's often easier to plan a machine-code program by writing it in a kind of 'dummy-BASIC'. The system's A register, or Accumulator, is used by almost everything-- a kind of high-speed messenger. The other two main registers, X and Y, tend to be used as counters or pointers--hence their description as 'index registers'. If we treat these as

```
Machine-code instruction...BASIC
     LDA...........A=...
     LDX...........X=...
     LDY...........Y=...
     TAX...........X=A
     TYA...........A=Y
     INX...........X=X+1
     STA...........=AorPOKE...A
```

We then get into the subtle game of the 6502's 'addressing modes'. Unlike the Z-80, which has different instructions for everything, the 6502 and 6800 processors use the same instruction in different ways, to increase programming flexibility while (in principle at least) maintaining some semblance of comprehensibility Thus the LDA instruction has a number of variations:

```
LDA#$20.....'immediate'........A=32
LDA$20......'zero-page'..........A=PEEK(32)
LDA$0220......'absolute'.........A=PEEK(2*256+32*1)
LDA$20,X......'zero-page indexed'A=PEEK(32+X)
LDA$2020,Y....'absolute indexed'.A=PEEK(32*256+32*1+Y)
LDA($20,X)....'indirect indexed'.A=PEEK(PEEK(32+X)
LDA($20),Y....'indexed indirect'.A=PEEK(PEEK(32)+Y)
```

The last two may seem unnecessarily complicated, but they are used a great deal to simplify the handling of look-up tables and the like. We'll be using a 'STA indexed indirect' for a fast screen-clear routine at the end of this article.

Most programming uses loops, conditional loops and branches. BASIC does these with FOR:NEXT, IF/THEN,ON/GOTOorON/GOSUB, GOTO andGOSUB. In machine code there are equivalents, but they sometimes work in a slightly different way. In particular, the IF/THEN tests are done against particular bits or 'flags' in a separate 'status register', referred to as the 'P' register; the much-used BEQ and BNE (branch if equal/not equal) tests check if the top bit in that register is set or clear respectively, for that bit is set if the last LD..(or some other 'move' instruction like ROL or INC) resulted in a zero value.

```
JMP$0020.........GOTO32
JSR$0020.........GOSUB32
BNE..............IF (last value) <>0 THEN GOTO...
BEQ..............IF(last value) =0THEN GOTO
JMP($0020) .......('jump indirect') GOTOPEEK(32)
```

The last one is something you can't do in BASIC! It's used by the SYN600 and CEGMON monitors to pass BASIC's input and output 'vectors' through RAM, to allow you to drive your own devices directly from BASIC--but more on that in a later part of this series.

The 'branches'--BEQ,BNE,BPL,BMI,BCS,BCC,BVS,BVC--we'll mostly leave till later, but there is one point about them which is important.

A JMP or JSR point to anywhere in memory; a branch to forward 129, to be precise. The branch instruction is followed by a one-byte 'displacement'--which can only have a value as a 'sign' flag, top-bit set meaning '--' or back. The maximum range is -128 to +127; but this is from the next byte ,so the effective range is -126 to +129 from the branch op code itself. It takes a little getting used to!

For demonstration, we'll show one of the most called-for routines-- a fast screen-clear. In OSI'S BASIC, you have two 'official' options:scrolling the screen with FOR X=1 TO 32: PRINT : NEXT-- which is messy and inelegant; or 'POKE' the screen with blanks'-- which is slow. For a C2 the shortest version of the latter would be+

FOR X = 53248 TO 55295 : POKE X, 32 : NEXT which, at a standard 1MHz clock speed, takes a good half-minute. The BASIC version of the fast machine-code screen-clear is rather more tangled:

```
10......A=208
20......Y=0
30......HI=255 : POKE HI,A
40......LO=254 : POKE LO,Y
50......X=4  : REM on 2k  screen memory
(C2/4, C1-E, etc) X=8
60......A=32
70......POKE  (PEEK(LO)+256 *
PEEK(HI)+Y,A
80......Y=Y+1:IF Y=256 THEN Y=0
90......IF Y<>0 THEN GOTO 70
100.....POKE HI, PEEK(HI)+1
110.....X=X--1
120.....IF X<>0THEN GOTO 70
130.....RETURN
```

Try running this as a BASIC program--you'll find it will take nearly twice as long as the simple 'POKE the screen with blanks'. The machine code is as follows:

```
A9D0......LDA#$D0......;  A000......LDY#$00......;zero  index  for
later 85FF......STAHI........;point  the LO(HI  pair
84FE......STYLO........;to  the  beginning of screen
A204......LDX#$04......;08  for  2K screens
A920......LDA#$20......;ASCII  'space'  91FE......STA(LO),Y....;blank
the  current location C8........INY..........;bump up current location
D0FB......BNE-5........;and  loop back until Y 'overflows'
E6FF......INCHI........;bump up to point to  next  'page'
CA........DEX..........;check last page not  done
D0F6......BNE-10.......;and clear next page if not
60........RTS..........;else return
```

This gives you a subroutine that will clear the screen in about one hundreth of a second--around 500 times faster. The catch is that you have to be able to get at it somehow! Although the routine itself is relocatable--it has no absolute addresses within it--it has to be put somewhere. On the assumption that it will be used with BASIC, we can place it in the 'free RAM' area in page 2(around 576 (b10)-760(b10) --$0240 is probably the safest all around.

Using the standard SYNMON/SYN600 monitor, type. (for address) 0240/(to input data as hex values) amd then each hex pair, separated by 'RETURN' (< >):A9< >D0< >A0< >00< >85 and so on to D0<>F6<>60<>. Each of these values should 'roll' into the two-digit part of the display; the two-digit area, to be replaced by your new value. After you've pressed 'RETURN' after the last byte, the 60, the address counter should read 0257.

With EXMON, type 0240, then each value separated by LINE-FEED. With CEGMON, type.0240 then type each value separated by, (comma), LINE-FEED or RETURN--it's probably simplest to type them is as in the assembly listing, such as:

A9,D0
A0,00

Since this is stored in that 'free RAM', it isn't wiped out by BASIC when you cold-start. But it is, of course, lost when you turn the power off. We can presume that you don't want to have to type it in with the monitor each time you start your system! But BASIC can't use hex--it only knows decimal--so we have to go about it another way. One is to convert the whole lot to decimal DATA statements, and POKE them in at the right addresses: FOR X = 576 TO 598 : READ Y : POKE X,Y : NEXT DATA 169, 208, 160, 0, 133, 255, 132, 254 DATA 162, 4, 169, 32, 145, 254, 200, 208, 251 DATA 230, 255, 202, 208, 246, 96

A final problem, for this time, is to connect this routine to BASIC in such a way that it can be called direct as a subroutine within BASIC. There is a provision for this, in the USR(X) function. As usual, OSI's description of this 'manual' is both garbled and wrong!

USR(X) is used as a means not just calling a machine-code routine, but of transferring values to it if required. We don't need that part at the moment; the only thing we need to know is that the BASIC statement X=USR(X) jumps to a machine-code subroutine pointed to by two locations. In the BASIC manual we are told that these are 23E and 23F--which appears to be true for disc systems under 65D, but not for BASIC in ROM. The right locations are (decimal) 11 and 12. They need a two-byte hex address, in the right order and, of course, converted to decimal--just to make things easier? The right order is the low byte first; a peculiarity of the 6502 which apparently allows it to run faster. If the start address of our screen-clear is a 576(b10), convert it to hex:$0240. The high byte is $02, and goes in 12(b10); the low byte is $40, or 64(b10), and goes in 11. Thus POKE 11, 64 : POKE 12,2 sets up the USR function to point to our screen-clear at $0240, 576(b10). Note, though, that although the screen then--in fact to point at the FC(function call) error--and will need to reset after cold-start.

To set up the screen-clear:

```
10  FOR X =576 TO 598 : READY : POKE X,Y
: NEXT
20 DATA 169, 208, 160, 0, 133, 255, 132,
254
30  DATA 162, 4, 169, 32, 145, 254, 200,
208, 251
40 DATA 230, 255, 202, 208, 246, 96
50 POKE 11,64 : POKE 12,2
```

Thereafter, to call the routine:

```
X=USR(X)
```

C1P HEAD LOAD/UNLOAD MODIFICATION
BY EDWARD J. KEATING

One of the many questions that OSI owners have asked relates to the operation of the mini disc drive. Why doesn't the disc spin down or the disc head unload from the disc.

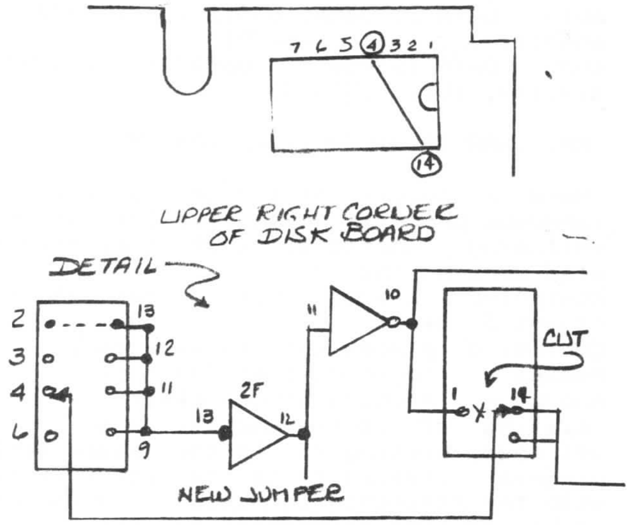
As to the question of drive spin up or down, in the interest of increased throughput OSI has decided that the disc should remain spinning because it takes

1.0 seconds to insure that the drive is up to speed. (300 rpm +/- 1.5%) The line from the disc drive controlling motor on is always grounded at the interface connector board that plugs into the 610 board. One of the problems that exists in using motor on commands is that no obvious feedback is available from the drive to determine the disc speed, other than counting index holes against a processor timing loop. Additionally there is also no indication that the drive is even powered up, hence the reason that no 'ERROR 6' will be seen by OS65D. (the status line from the P1A is grounded to always return a true signal.) The life of the motor as rated by MPI, the makes of the drive, is quoted as approximately 2500 hours can be obtained. The cost of the replacement motor from MPI is $30.00 but labor may approximate that. (meantime to repair is quoted at .5/hour)

One modification that can increase the life of the drive is exceedingly minor. After I received a mini disc, 610 board and copy of OS65D to add to a 'naked' superboard, one of my first undertakings was a complete disassembly listing of the operating system. One of the interesting routines found was inside the floppy disc driver module. This routine was responsible for timing the head setting time and toggling a line from the PIA PB7 (pin 17) which ends up on connector J3 pin 1 after going through a 7417. (U75) The cable connects this signal to pin 14 of the mini disc interface which is identified as DRIVE SEL 3 or DS2. The subroutine in the driver was being faithfully called to turn on the bit after a seek request , but before and read or write request. The other half of the routine was just as faithfully called after the read or write to turn the bit off (ground true signal). As the C1P only 2 disc drives, I wondered if this might not be a superfluous carryover from an 8' floppy disc driver. Since the C1 has access to just about all bootstraps, the 5 and 8' bootstraps are both in the monitor rom area even if addressed at a non-runnable location. Consulting the MP1 interface specification, the disc head can be loaded by either drive select or motor on but not an external source. Examining the schematic of the disc drive resulted in the modification. In essence, you must break the connection between pin 14 and pin 1 of the dip header plus (which was head load with DRIVE SEL 3). I did this on a separate dip header plug and replaced the entire shunt. After this modification the DS2 or DRIVE SEL 3 line will now load or unload the head and the drive will make clicking sounds. Also remember that no increase in delays are necessary because the routine always did exist in the system for head settling time anyway.

On a 2 drive system this modification may require an additional 'AND' gate to mask DS2 with the current drive select, otherwise both disc heads will load or unload at the same time. This extraneous system, but may become annoying after a time.

This modification has been done to several C1P's with no apparent problems. In fact it allows easier removal and re-insertion of diskettes without bumping the disc head causing the ERROR 5.



WE GET LETTERS

CARL M. KING, SARASOTA, FLA.

I have picked up some excellent suggestions from the JOURNAL. In my major program, which I call COGO Surveying, I have occasion to tabulate the results of a series of calculations in a table of itemized coordinate pairs. Earlier I had decided that the tabulation would look much nicer and be more readable, if we could justify the decimal point location in each column, much as it would appear on an adding machine tape. Well, I worked out a slow and byte consuming routine for doing it, but I was not entirely happy with it. Isn't BASIC amazing! You can obtain the same result in more than one way. My method was to calculate the common log of each number, and subtract the characteristic from the TAB spacing. It worked, but it was slow and expensive in program steps.

I was thrilled with the suggestion in the first issue of the JOURNAL on page 5 - "STRINGS FOR DIGITAL MANIPULATION."
So now I have a neat new routine, which your readers might find helpful and instructive, and I am pleased to contribute the following sample program:

```
300    N=N+1:PRINT N; :READ X:Y=10:GOSUB
5000
310      READ X:Y=28: GOSUB5000: PRINT:
GOTO300
5000   W=ABS(X):  Z=LEN (STR$(INT(W)):IF
W<1 AND W=>.01 THEN Z=1
5010  IF Z>7 THEN Z=7
5020  PRINTTAB(Y-Z) X;: RETURN
6000   DATA 123456,  .0654321, 12345.6,
.654321, 1234.56, 6.54321
6010  DATA 123.456, 65.4321, 12.3456,
654.321, 1.23456, 6543.21
6020   DATA .123456, 65432.1,  .0123456,
654321, .00123456, 6543210
6030   DATA 1234560, .00654321, 3.14159,
314.159, 10.1, 3333.33
```

MR. CHARLES STEWART, ADRIAN, MI

Here a couple of simple machine language programs which give the C1P the following functions. (note they should also work on the C4)
Control-L places unit in load return, clears as usual
Control-S places unit in save mode
Escape -   performs list function
Rubout -   machine screen clear
Listing #2 is the machine code utilized, Listing #1 is the basic POKE program. Listing #3 is the same routine with the reference to RUBOUT removed and input routine referenced to the beginning of the C1P cursor control program. If listing 3 is loaded with cursor control it gives all cursor functions plus the control functions.

LISTING #1

```
 60 FORX=218TO249:READA:POKEX,A:NEXT
 70 DATA32,186,255,201,12,208,3,32,139,2,55,201
 80 DATA19,208,3,32,150,255,201,127,208,3,76
 90 DATA34,2,201,27,208,3,76,181,164,96
100 FOR X=546TO567:READA:POKEX,A:NEXT
110 DATA72,169,32,162,0,157,0,208,157,0,209,157,0,210
120 DATA157,0,211,232,208,241,104,96
150 POKE11,34:POKE12,2:POKE536,218,:POKE537,0
160 X=USR(X):PRINT"*CONTROL VERSION #1":PRINT"*BY CHARLES A. STEWART"
165 PRINT:PRINT
170 PRINT"ESC LISTS":PRINT"RUBOUT GIVES SCREEN CLEAR"
180 PRINT"CONTROL S = SAVE:PRINT"CONTROL L = LOAD"
200 NEW
```

LISTING #2

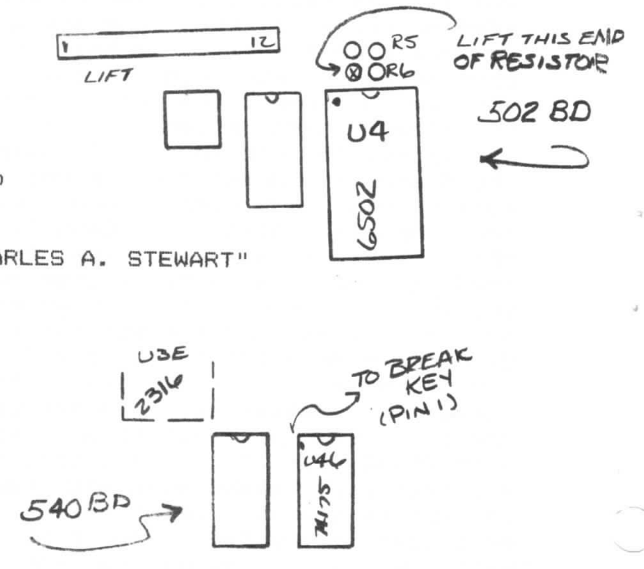
```
00DA.....JSR$FFBA   ; input subroutine
00DD.....CMP #$0C...; compare to Control L
00DF.....BNE HERE...; branch if not equal
00E1.....JSR $FF8B..; execute load
00E4 HERE CMP #$13..; cmp to control S
00E6.....BNE HERE1..;
00E8.....JSR $FF96..; execute save
00EB.HERE1 CMP #$7F.; cmp to rubout
00ED.....BNE HERE2..;
00EF.....JMP $0022..; execute screen clear
00F3.HERE2 CMP #$1B.; cmp to escape
00F5.....BNE END...;
00F7.....JMP $A0B5.; execute list
00FA END RTS.......;


0222.....pHA.......; push A to stack
0223.....LDA#$20...; load space character
0225.....LDX#$00...; load accx with 0
0227.CON.STA $D000,X; store on screen
022A.....STA $D100,X; "
022D.....STA $D200,X; "
0230.....STA $D300,X; "
0233.....INX.......; increment x
0234.....BNE CON....; branch back to $0227
0236.....PLA........; pull A from stack
0237.....RTS........; return from subroutine
```

Listing #3

```
240 FORX=218TO249:READA:POKEX,A:NEXT
250 DATA32,42,2,201,12,208,3,32,139,255,201
260 DATA19,208,3,32,150,255,234,234,234,234,234
270 DATA234,234,201,27,208,3,76.181.164.96
280 PRINT"*CONTROL VERSION #1"
285 PRINT:PRINT
290 PRINT"ESC LISTS":PRINT"RUBOUT GIVES SCREEN CLEAR"
295 PRINT"CONTROL S=SAVE":PRINT"CONTROL L=LOAD"
300 POKE536,218:POKE537,0
```

DAVID RODENBERGER, COLUMBUS GROVE, OH

I have used my OSI C4P for about 6 months now, and have become very tired of poking 56832,1 every time I bring up my system from a cold start. So I decided to do something about it. After looking over the schematics, I found that the problem would be very easy to solve. All that is needed is to connect a wire from one of the integrated circuits to the break key. Then when you bring the system up and hit break, the color and sound is automatically reset to off.



The first thing that must be done is to remove the six screws from the bottom plate. Then revome the ground wire from the bottom plate. Next turn the computer upside down with the keyboard to your left. Remove the bolt from the lower left corner of the 502 board. Now remove the two connectors from the lower right and remove the 502 board. Set it aside. Now refer to the IC layout for the 540 board. Locate IC U4G. Attach a wire about 18 inches long to pin one of the IC U4G and route it over to the break Key. Attach the wire to the pin closest to the front of the key board, on the break key. This pin goes to the reset pin on the microprocessor.

The next step is to reinstall the 502 board being careful not to bend the pins or the board. You could damage it. Locate the microprocessor on the 502 board. This is the largest IC on the board, 40 pins. Locate the two 4,700 ohm resistors at one end of the IC. Remove the longest wire of the resistor, closest to the microprocessor, from the board. This finishes the modification. We remove the resistor because there is already a pull up resistor on IC U4G pin one, so you don't need two. Also, it isn't good practice to have two pull up resistors on the same pin.

The chip you hooked the wire to is a flip-flop and pin one is the reset. So when you bring this pin low by hitting break, you reset it along with the processor. Have fun!

GEORGE FISHER, ARLINGTON, VA

Received my order and the back issues of the Aardvark Journal". Was most pleased with the Journal, as I am a newcomer to the "info-gap" world of OSI. My experience began about a month ago when I acquired a Superboard II. However, this turned out to be the new production board (series 2 or revision D, 1980) and it has a number of changes/features different from its predecessor. As you may not have encountered this yet, I will try to give you a summary of the differences (as far as I have been able to go). The documentation is not one of the improvements.

1. New circuits have been added to provide an automatic reset on power-up. Also, this new circuit is designed so that the <BRK> key must be held down about 3 seconds to initiate a reset. (A big help for fat fingered typists.)
2. The board comes with the RS-232C circuit fully populated and wired.
3. The connector J4 has been changed to a 2-pin molex which is used for output (noise) of the DAC. The DAC is now 8 bits, and has the components (resistors and diodes) installed and must be enabled via the program. (Poking various values into location 55296) The keyboard busses which had been brought out to the J4 connector are no longer available at a connector.
4. There are a few other new IC's on the board and three new empty sockets-to accommodate color video expansion-one a 8T28, one a 2114, and the third is a connector for color board interconnections.
5. Two pre-recorded tapes in addition to the demo tape are also included. One is called Video Swap, which changes the video drive routine and gives an optional 48 character x 12 line video output. The other, is a program which permits the computer to be used as a (almost dumb) terminal. Now for what didn't come with the computer. There were no schematic diagrams or part layout diagrams. (I called OSI and asked for "Customer Service", but I was told by operators all such calls or inquiries must be made to their dealers....). A helpful dealer did send me Xerox copies of the diagrams, a bit hard to read but better than nothing. Also, there are no instructions on how to connect up the RS-232C or modem outputs in the manual. The connector J2 (which has the various in & out data connections) has a plug on it with three jumpers. Yet no info is provided on what these leads do or should go to for other configurations.

C.W. AIGELDINGER, AMHERST, VA

Not only does the addition of the Aardvark Sound Gen. Board open new possibilities of generating sounds and computer generated sound effects for programs etc., but the board also has special treat.

The board decodes addresses 9FF8 and 9FF9 off the address buss using 2-7430's and one 74LS138. These addresses are dedicated to enable the sound generator and its many functions. Well the 74LS138 (3 of 8 decoder) also generates 6 more distinct addresses. These addresses are generated, but they are not used by the sound board and can be used for any pupose where 1 out of 65536 addressed are needed.

One example of using these addresses: This is a sample CIP Modification which will give the user programmable full screen This is a sample CIP reverse video under keyboard or program control. On power-up the modification is disabled (normal video) and will only switch to reverse when prompted to do so by keyboard or program. It requires minor modification to the CIP and minimum cost when used with the Aardvark Sound Board.

This modification requires the cutting of the runs going to pins 4 and 5 of U70 on the CIP. Once this is done then short wires can be run to a second board containing the 74LS75 and 74LS09.

Once the modification is installed it works like this:

On power-up the 74LS74 flip-flop is preset to a hi condition on its Q output which is the controlling input to essentially and exclusive or package formed by the 74LS00 and U70. (when Q is hi normal video is displayed. When Q is lo reverse video is displayed.)

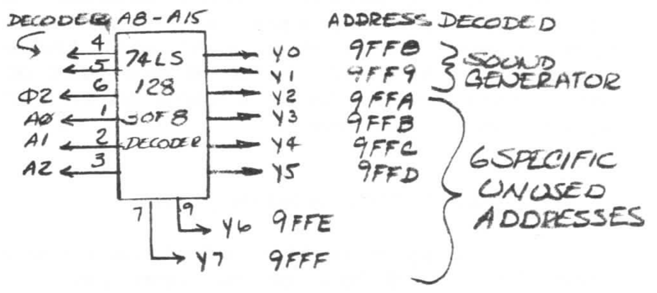
When reverse video is desired all that is done is to use the command: POKE 40958,00

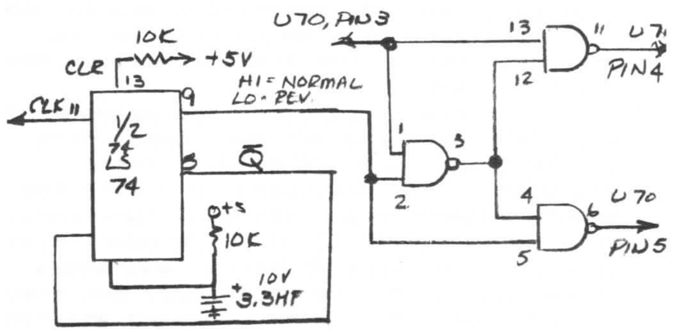
Either in immediate mode or from program control and the display will reverse to restore the display to normal the same command is used again and the display will return to normal. Any data from 0 to 255 (decimal) can be poked into 40958 dec. (9FFE Hex) Since the data is not used (only the actual enabling of the address is necessary)

If the command is used in a for next loop:

```
10 FOR R= 1TO20
20 POKE 40958,00
30 NEXT
```

within another program then a striking attention getting display follows. This is useful to bring attention to any part of a program (errors etc.).





DICK SWEET, ST LOUIS, MO

Tack this short sub-routine at the end of any documented program, turn your cassette on to record, and type RUN63000. When the program is finished, load the new tape on top of your original program and have a condensed version ready to run.

Two cautions: Please make a tape of your well documented program before running at 63000, and never make your REM statements start at anyplace but at the beginning of a program line.

REM ERASER

```
63000 I=PEEK(124) *256+ (PEEK(123)):
SAVE: FORX=768TOI: REM FIND THE END OF
THE PROGRAM AND LOOP TO IT
63010 IF PEEK(X)=142 THEN ?PEEK(X-2)+
(256*PEEK(X-1)) :REM LOOK FOR REM
STATEMENTS AND SAVE ON THE TAPE THE LINE
NUMBERS THEY APPEAR ON
63020 NEXT: ?63000: ?63010: ?63020:REM
ERASES THE REM ERASER WHEN THE NEW TAPE
IS LOADED.
```

TODD BAILEY, GREENVILLE, OH

To eliminate the annoying little bug that you get when combining the MINOS game with the C1E chip (the maze takes forever to come up and sometimes never does), add the following lines to the program:

```
97 POKE560,H+6: REM H=HORIZONAL
SIZE OF MAZE
1207 LO=0:REM SET NEW COUNTER
1275 LO=LO+1:IFLO = PEEK(560) THEN
RETURN: REM INCREMENT COUNTER, COMPARE
COUNTER TO SIZE OF MAZE AND KICK OUT OF
LOOP WHEN READY.
```

Also C1E (and C2E) owners can enhance the AARDVARK adventures by taking advantage of the machine code save to preserve games in process. PEEK memory location 133 and 134 (DEC) to get the LO and HI bytes of the top of your program, including all variables and strings. If you are saving an ADVENTURE running in 8K RAM, just use $00 as the low byte and $20 as the high byte. This saves everything up to 8K. When you get to the point that you want to save your program, hit the BREAK key and answer 'M' to the D/C/W/M prompt to go into the monitor mode. Hit 'S' to start the SAVE mode. Enter your starting address ($0300 for the base of a BASIC program), then enter the end address, hi byte, low byte, (remember to change the DEC PEEKs from BASIC back to HEX to use with the monitor). Enter your restart address (in this case use $FE70 to jump back to the load mode. After you have saved this onto tape, enter 'S' again to go back to the save mode. This time use a starting address of $0000 and an end address of $0086 with a restart address of $0000 and save this on the next section of your tape. The program is now saved. To load it back in, cold start your system in BASIC. Then BREAK and go to the monitor mode, hit jump back into the load mode to laod your pointers, and then jump to a warm start. After the 'OK' message, run the program with a GOTO(line number of the "your command" line. Do not use the RUN command as it will overwrite the variable you just loaded. You are now right back where you left off.

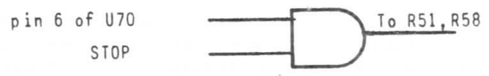
GARY WHEELER, SAN DIEGO, CA

I was quite pleased to see the C1P video mod in your December journal. I did however have a little bit of trouble building the mod.

First of all, the resistor of the '123' is unlabelled. When I called Aardvark, either Jane or Judy told me it was 200 ohms. This worked fine, although I had to replace the 47pf capacitor with a .001 uf capacitor.

Secondly, I had some stability problems. This didn't bother me until I shortened the resistor leads to a bare minimum with no improvement. To work, the circuit requires some capacitive coupling between the HS and the STOP lines. Lack of this gives grounds for the "fuzzies". To cure this, disconnect pin 1 of the '123 from the RC network and from pin 2 and connect it to pin 5, the STOP line. This locks the video clock to the HS line and cures the fuzzy character problems.

Finally, I had some wrap-around problems. Although this is a quirk of my monitor, others may have the same trouble. Its symptom is having the white bands, resulting when you stop the clock, reflected halfway across the screen. The cure is a simple one, change the white bands to blanks. This simple circuit does that:



Unfortunately, the Superboard doesn't have a spare AND gate so you must use your own.

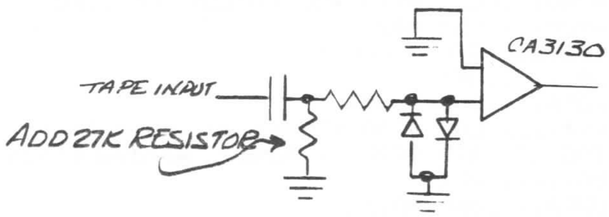
Although, I had a couple of minor problems, the mod works great and I am very happy with the results.

While the 32x32 display created by this display is good, a 32x64 display is ultimately needed for some applications i.e. text editing etc. Wouldn't this scheme work? Connect two more 2114's to the VA and VD lines of the video circuitry. Connect pin 8 of both new rams to the unused multiplexer of U55. Supply the appropriate inputs to the multiplexer and then connect pin 10 of both rams to the appropriate pin of U20. With this installed, adjust the video clock to a frequency twice that needed for the 32x32 display. This should give a 32x64 display to Superboard owners.
EARL MORRIS, MIDLAND, MI

The February issue of the Aardvark Journal contained an article on reducing cassette loading errors. I would like to suggest a hardware change and a more scientific manner of making the timing adjustment.

Below is shown the input circuit for OSI's cassette tape interface. This circuit has two problems: The input to the amplifier has no D.C. path to ground and the input is coupled by a capacitor but since the amplifier has a very high input impedence, response extends to low frequencies. A cheap and simple fix is to install a $.05 resistor from the input capacitor to ground as shown. I used a value of 27k ohms. This now gives the amplifier a D.C. path to ground and creates a high pass filter with a corner frequency of 600 cps. Noise and hum at 60 cps are attenuated a factor of 10. After making this modification, the volume control setting on your recorder will be much less critical.

The second suggestion regards adjusting the timing of the one-shot. Do not confuse this with the baud rate adjustment which is set at 4800 cps as per the previous article. The one-shot circuit determines whether the input is at 1200 or 2400 cps. The trip point of this circuit should be near the geometric mean (1700 cps) of the two frequencies. Correct adjustment requires that an audio oscillator be connected to the cassette input port. If a voltmeter is connected to the 7474 flip-flop of the cassette circuit, the meter will change from 5 volts to zero as the frequency of the oscillator passes through the trip point. If the frequency is much defferent than 1700 cps, the one-shot trimpot must be adjusted. Anywhere between 1600 and 1800 cps should be good.



KERRY LOURASH, DECATUR, ILL
SCROLLING' DOWN THE AVENUE

Here's a short ML routine, relocatable anywhere in memory, that scrolls the screen down instead of up. It has the exact opposite effect of a PRINT that doesn't print anything.

When I was a novice (not long ago!), the screen scroll was a fascinating mystery. Now that I know how it works, I still think it's a neat trick. Let's take the C1P as an example. The video display is organized as 32 rows of 32 columns. If you look at the table in the back of the OSI graphics manual, you see that each memory location from $D000 to $D3FF is assigned one small area of the display. Those of you who have done some graphic experiments know that, to make a character move straight up, you subtract 32 from its present address in Video memory and put the character in that position.

The scroll routine in BASIC does the same thing, but on a larger scale. Starting at the top of video memory, it transfers the contents of each memory location to the row above. When the scroll is completed, the home line, the line the cursor occupies, is cleared. Only the home line must be erased, since each byte that is moved takes the place of the old byte in that location. The scroll routine is part of the video output routine located at $BF2D.

My scroll routine moves each byte down, not up. Zero-page loctions $50-51 hold the target address and $52-53 hold the source address. The Y register is used as a counter for the individual video addresses and the X register counts the

DOWN SCROLL ROUTINE

```
ADDRESS CODE      MNEMONICS     *-CHANGE FOR CLIP
 0130 A9BF....LDA #$BF *DF load start address
    2 8550....STA $50
    4 A9FF....LDA #$FF
    6 8552....STA $52
    8 A9D7....LDA #$D7 *D3
    A 8551....STA $51
    C 8553....STA $53
    E A208....LDX #08 *4 ; page counter
 0140 A000....LDY #00    ; byte counter
    2 b150....LDA (50),Y ; scroll 1 byte
    4 9152....STA (52),Y
    6 88......DEY
    7 D0F9....BNE $0142
    9 C651....DEC $51.....;next page
    B C653....DEC $53
    D CA......DEX.........;decrement page counter
    E 10F2....BPL $0142
 0150 A240....LDX #$40 *20 ;erase top line
    2 A920....LDA #$20
    4 9D00D0..STA $D000,X
    7 CA......DEX
    8 D0FA....BNE $0154
    A A240....LDX #$40 *20 ;blank home line
    C 9DC0D7..STA $D7C0,X *D360, Aardvark-D340
    F CA......DEX
 0160 D0FA....BNE $015C
    2 60......RTS
```

pages (1 page equals 256 bytes). Since there is nothing to scroll into the top line, it must be cleared of the garbage loaded into it. By the way, this garbage is the result of trying to read non-existant memory. The home line is also cleared, although this feature is optional. Turn it off with a POKE346,96 and restore it with POKE346,162. The routine is meant to be used as a USR routine. Set the USR vector with a POKE11,48:POKE12,1. Be warned that the routine is located in the stack and could be partially destroyed under some conditions. To relocate the routine, copy the bytes in a different section of memory with the monitor or change the values of X in line 10 of the BASIC load program. Just for fun, see the effect of changing location 305(normally $BF or 191 decimal). When using this program with the Aardvark cursor control program, the address of the home line should be changed. For the C1P, change the first data element in line50(96) to 64. I'm not sure about the C2P, but I think it should be (192) to 128.

BASIC LOAD-C2P

```
10 FORX=304 TO 354 :READA: POKEX,A: NEXT
20 DATA169,191,133,80,169,255,133,82,169,215,133,81,133,83,62
30 DATA8,160,0,177,80,145,82,136,208,249,198,81,198,83,202
40 DATA16,242,162,64,169,32,157,0,208,202,208,250,162,64,157
50 DATA192,215,202,208,250,96
```

BASIC LOAD-C1P

```
10 FORX=304 TO 354 :READA: POKE,A: NEXT
20 DATA169,223,133,80,169,255,133,82,169,211,133,81,133,83,162
30 DATA4,160,0,177,80,145,82,136,208,249,198,81,198,83,202
40 DATA16,242,162,32,169,32,157,0,208,202,208,250,162,32,157
50 DATA96,211,202,208,250,96
```

DEMO PROGRAM

```
100 FOR Z=1TO32: PRINT: NEXT
110 POKE11,48: POKE12,1:FORY=1TO9:PRINT"A("Y")";:INPUTA(Y):
   X=USR(X):NEXT
120 POKE346,96:POKE305,PEEK(305)-1:DATAR,E,V,E,R,S,E
130 FORI=1TO7:READA$:PRINTA$:X=USR(X):PRINT:NEXT
140 POKE346,162:POKE305,PEEK(305)+1
```

The AARDVARK JOURNAL is published six times a year by AARDVARK TECHNICAL SERVICES, LTD., 2352 South Commerce, Walled Lake, MI 48088. Subscription rates: $9.00 per year ($14.00 overseas). Copyright 1981, AARDVARK TECHNICAL SERVICES, LTD. Bulk rate postage (permit #5) paid at Walled Lake, MI, 48088

NEW AT AARDVARK

The single stepper/monitor is now available on tape for $19.95. Dave Edson has come through with a couple of new machine code goodies for the C1P. We now have Interceptor, Collide ($9.95), and Monster Maze ($12.95) for the C1P.

We have a new 16k ram board for the C1P. It comes with instructions to adapt it to the C2/C4/C8P's also. The bare board is available now for $39.95. An assembled version of both the PROM Burner and the Memory boards is in the works and both should be available before the next issue comes out.

Chuck Scotts' super super disk is now available. It has a 14 character file name capability, the new machine code editor, and a disk manager with capabilities like disk packing. It is a great 65D executive - and we are selling it as Superdisk II for $39.95

How about this for a great diagnostic. We have a disk speed analyser on disk. It reads out disk speed accurate to 1/2 %. It sells for $8.95.

CLASSIFIED ADDS

C1P MINIFLOPPY (MODEL 1) 24K RAM. $895. THIS IS AARDVARKS workhorse C1P. It has been used a lot but is still in good shape. Can be had with 32k ram installed for $950.

RADIO SHACK QUICK PRINTER II AND 4 ROLLS OF PAPER. $140 Todd Bailey. (513) 782 5083 or (513) 548 1312 after 5.30 p.m.

ABOUT THIS MONTH'S PROGRAMS.

I am going to include Battlefleet and Slashball as this months software. The ostensible reason is that they show good control setups and that Battlefleet demonstrates the mixing of PRINT and POKE graphics on the same screen. Battlefleet also demonstrates that you can pack a lot of data and do a lot of programming in 4K. I spent a lot of time getting it in what was then a standard sized system.

Those are the official reasons. The real reason is that they are two fun games that everyone likes when they play them and which no one buys because they don't describe well in a catalog. I want to show them off and have people enjoy them and since no one has bought them, we have to print them here.

Slashball is diabolical. It looks simple, but gets very hard after the first few rounds. The controls on that one are simple but take real skill to master.

BATTLEFLEET was one of the first programs I ever wrote and one of the main reasons I purchased a computer. It is a very complex form of Battleship that I once played with friends. Unfortunately, the bookkeeping involved made it impractical to play without a computer. There is virtually no luck involved in the game!

The computer will hide 3 ships in the grid. They are two dimensional, 2x6 squares and may be horizontal, vertical or on a 45 degree slant from lower left to upper right. They will not overlap but may touch. The really hard part is that you will have to fire volleys of 6 shots at a time and will be told only how many hit - not which ones or on what ships. That you will have to test to determine.

You fire with a gunsight that initially appears in the upper left corner of the grid. You move the sight with (<) and (>) and the (U) and (D) keys. (guess which is which). You fire at the grid point under the sight with the (F) key. At the end of six shots. the system will display the number of hits. When you have sank all 3 ships (or run out of shells) the system will displays the grid image. To help you sort out volleys, there is a Review command (R). To review, press (R) and then tell the system which volley you want to start with.

Just a reminder - with the office now located away from our home, we can only answer questions on orders (or anything else that requires us to consult the files) during the hours of 8 AM to 3 PM (Mon - Fri) when the office staff (namely Judy) is here. After 3, we can take orders (or questions if you don't mind calling back the next day for an answer), but that's about the limit. We're still getting phone calls at strange hours (2 AM and so on). Rodger's schedule varies a lot now as administrative duties take more and more of his time, so we're often asleep then, and as the phone rings at the house.......

```
10 PRINT:PRINT"BATTLEFLEET":PRINT"COPYRIGHT 1978 R. OLSEN":PRINT:PRINT
15 PRINT"HIT SHIFT TO START"
17 IFPEEK(57088)=1ORPEEK(57088)=254THENR=RND(89):GOTO17
20 LL=64:GL=54083:IFPEEK(57088)>127THENLL=32:GL=53608
30 DIMST(180),H(30),Q(12),A(12),B(12),C(12)
32 REM DIFFICULTY DETERMINES NUMBER OF SHELLS.
35 INPUT"HOW GOOD ARE YOU (1-10)";X:SK=179-6*X
36 REM MAKE SOME SHIPS
40 POKE56900,0:P=P+1:PRINT"WORKING ON SHIP "P
50 L=1:M=INT(9*RND(8)+1):N=INT(9*RND(8)+1)
60 ONRND(8)*3+1GOTO70,110,90
69 REM HORIZONTAL SHIPS
70 S=LL*N+M:FORX=0TO5:FORY=0TO1:Q(L)=S+LL*Y+X
80 L=L+1:NEXTY,X:GOTO140
89 REM VERTICAL SHIP
90 S=LL*M+N:FORX=0TO1:FORY=0TO5
100 Q(L)=S+LL*Y+X:L=L+1:NEXTY,X:GOTO140
109 REM SLANT SHIP
110 M=INT(9*RND(8)+1):N=INT(9*RND(8)+6):S=LL*M+N
120 FORX=0TO(5*(LL-1))STEP(LL-1):FORY=0TO1
130 Q(L)=INT(S+LL*Y+X):L=L+1:NEXTY,X
140 ONPGOTO150,160,180
149 REM STORE SHIP A
150 FORM=1TO12:A(M)=Q(M):NEXT:GOTO40
159 REM CHECK FOR OVERLAP AND STORE SHIP B
160 FORM=1TO12:B(M)=Q(M):FORN=1TO12:IFB(M)=A(N)THEN50
170 NEXTN,M:GOTO40
179 REM STORE SHIPC : SHIP POSITIONS ARE A(X),B(X),C(X)
180 FORM=1TO12:C(M)=Q(M):FORN=1TO12:IFC(M)=A(N)ORC(M)=B(N)THEN50
190 NEXTN,M:W=1:POKE8955,43:POKE8956,37:GOSUB430
191 REM FOR BASIC IN ROM REPLACE 8955 AND 8956 POKES WITH
192 REM WITH POKE11,0:POKE12,253
195 REM TO SEE SHIPS, THIS LINE IS 195GOSUB 470 (DELAY): GOTO30
200 PN=GL+LL:POKEPN,43:FORM=1TO6:P=PEEK(PN)
209 REM POLL KEYBOARD Y=ASCII OF KEY PRESSED.
210 X=USR(X):Y=PEEK(9815)
211 REM FOR TAPE SYSTEMS Y=PEEK(513): FOR C1PMF Y=PEEK(9834)
215 IFY=70ANDP=61THENPOKEPN,42:P=42:Q(M)=PN-GL:GOTO280
219 REM REVIEWS TO SEE IF ANY SHOTS FIRED AND (R) PRESSED.
220 IFM=1ANDY=82ANDI>1THENGOSUB550:PRINT"READY?":X=USR(X):GOTO410
230 POKEPN,P:IFY=44THENPN=PN-1
240 POKEGL+LL,32:IFY=46THENPN=PN+1
250 IFY=68THENPN=PN+LL
260 IFY=85THENPN=PN-LL:X=RND(8)
270 P=PEEK(PN):POKEPN,43:GOTO210
280 NEXTM:I=I+1
289 REM CHECK FOR HITS Q(W)=CURRENT SHOT. H(I)=TOTAL HITS
290 FORM=1TO6:FORN=1TO12
300 IFQ(M)=A(N)THENHA=HA+1:H(I)=H(I)+1
310 IFQ(M)=B(N)THENHB=HB+1:H(I)=H(I)+1
320 IFQ(M)=C(N)THENHC=HC+1:H(I)=H(I)+1
329 REM HA,HB,HC ARE HITS ON SHIPS A,B, AND C.
330 NEXTN:NEXTM
339 REM COUNT UP SUNK SHIPS
340 M=0:IFHB>11THENM=M+1
350 IFHC>11THENM=M+1
360 IFHA>11THENM=M+1
```

```
370 PRINT"WE HAVE SUNK"M"ENEMY SHIPS"
380 PRINT" WITH"HA+HB+HC" HITS"
390 IFM>2THEN500
395 IFW+5>SKTHENPRINT"SORRY, WE'RE OUT OF AMMO"
397 IFW+5>SKTHENPRINT"HERE'S WHERE THEY WERE":GOSUB470:GOTO530
399 REM ADD SHOTS TO STORAGE (ST=STORAGE W=TOTAL SHOTS)
400 FORM=1TO6:ST(W)=Q(M):W=W+1:NEXTM
409 REM READS OUT EACH SHOT SO FAR
410 C=1:N=1:GOSUB430:GOSUB590:GOTO200
420 GOSUB590:GOTO200
430 PRINTSK-W+2"SHELLS LEFT
433 REM PRINTS BACKGROUND
435 PRINT"0   123456789012345"
440 FORM=1TO9:PRINTM" ===============":NEXTM
450 FORM=10TO15:PRINTM"===============":NEXTM
460 RETURN
469 REM POKES IN SHIP LOCATIONS
470 GOSUB430:S=GL:FORL=1TO12
480 POKES+A(L),49:POKES+B(L),50:POKES+C(L),51:NEXTL
490 RETURN
500 PRINT"CONGRATULATIONS YOU":
510 PRINT"SANK ALL THREE SHIPS WITH "W" SHOTS
520 GOSUB470
529 REM CLEAR VARIABLES FOR NEW GAME
530 FORX=1TOW:ST(X)=0:NEXT:W=0:HA=0:HB=0:HC=0
540 INPUT"READY TO START AGAIN?";A$:P=0:RUN20
550 R=1:FORX=1TO30:PRINT:NEXT:PRINT"WE HAVE FIRED "I" VOLLEYS
560 INPUT" WHICH ONE DO YOU WANT TO START WITH";N
570 IFN>ITHENPRINT"NOT FIRED YET":PRINT:GOTO550
579 REM POKE UP PAST VOLLEYS FOR REVIEW OR NEW TURN
580 C=N*6-5:GOSUB430
590 FORX=CTOC+5:POKEGL+ST(X),H(N)+48:NEXT
600 IFR=1THENX=USR(X):GOTO620
610 FORX=1TO700:NEXT
619 REM LOOPS TO NEXT VOLLEH UNLES C>=W (TOTAL SHOTS FIRED)
620 IFC+6<WTHENC=C+6:N=N+1:GOTO590
630 R=0:RETURN
```

```
10 PRINT:PRINT:PRINT"SLASHBALL":PRINT
20 PRINT"COPYRIGHT R. OLSEN 1979":PRINT:PRINT:PRINT
30 W=64:L=W:C1=53248:C2=55230:KB=57088
40 VB=600:IFPEEK(57088)<128THENVB=540:GOTO70
50 W=25:L=32:C1=53315:C2=54205
60 SP=57089
70 INPUT"DO YOU WANT INSTRUCTIONS";A$:IFLEFT$(A$,1)="Y"THEN900
80 PRINT:PRINT:INPUT" HOW HARD DO YOU WANT IT (1-11)";X:D=(13-X)*100
85 ADT=16:B=26:A$=""
90 FORX=1TO32:PRINT:NEXT:FORX=C2-6*LTOC2:POKEX,32:NEXT
95 OF=4096:FORX=C1+OFTOC2+OF:POKEX,4:NEXT
100 TU=0:R1=0:R2=0
105 REM DRAW SCREEN
106 POKE56832,7
110 FORX=1TOW:POKEC1+X,88:POKEC2-X,88:NEXT
120 FORX=1TO32:POKEC1+X*L,88:POKEC2-X*L,88:NEXT
130 AP=(C2-C1)/2+C1:IFVB=600THENAP=53743
145 A$="":FORX=1TO20:A$=A$+CHR$(PEEK(AP-10+X)):NEXT
150 POKEAP,161:POKEAP-1,161:POKEAP+1,161:IFVB=540THENPOKEAP+2,161
```

```
160 ADD=16:IFPEEK(KB)<>129ANDPEEK(KB)<>126THEN160
165 PN=INT(55106+56*RND(8)):MF=-L
167 DE=D/25
170 POKEPN,B
175 FORX=1TODE:NEXT:DE=DE-1
180 PS=PEEK(PN+MF):IFPN<C1THEN165
185 TU=TU+1:IFTU>DTHEN772
200 IFPS=88THENGOSUB5000:MF=-MF:GOTO300
230 IFPS=92THEN330:REM \
240 IFPS=47THEN400:REM /
250 IFPS=161THEN540:REM TARGET
260 P=PEEK(KB):IFVB=600THENP=255-P:REMSCAN KB
265 IFP=3ORP=5THENADD=ADD-1:IFADD<0THENADD=0
270 IFP=3THENPOKEPN+MF,92:FORX=1TO60:NEXTX:GOTO180
280 IFP=5THENPOKEPN+MF,47:FORX=1TO60:NEXTX:GOTO180
300 POKEPN,32:PN=PN+MF:POKEPN,B:GOTO170
330 POKESP,90:POKEPN,32
340 PN=PN+MF:REM SERVICE \
350 IFMF=1THENMF=L:GOTO460
360 IFMF=-1THENMF=-L:GOTO460
370 IFMF=LTHENMF=1:GOTO460
380 IFMF=-LTHENMF=-1:GOTO460
390 GOTO460
400 POKESP,99:POKEPN,32
410 PN=PN+MF
420 IFMF=1THENMF=-L:GOTO460
430 IFMF=-1THENMF=L:GOTO460
440 IFMF=LTHENMF=-1:GOTO460
450 IFMF=-LTHENMF=1
460 PS=PEEK(PN+MF)
461 POKESP,0
470 IFPS=92THEN340
480 IFPS=47THENGOTO410
490 IFPS=88THEN200
500 IFPS=161THEN540
510 POKEPN+MF,B:PN=PN+MF:GOTO170
539 REM END OF TURN DISPLAY AND SCORE
540 C=PEEK(PN):IFC<>47ANDC<>92THENPOKEPN,32
541 POKESP,100
545 IFAD=16THENB$="  SERVE AGAIN   ":GOSUB830:GOSUB840:GOTO580
550 TU=0:A$="":FORX=1TO20:A$=A$+CHR$(PEEK(AP-10+X)):NEXT
551 POKESP,190
552 B$="      SCORE      ":GOSUB830
553 GOSUB840:IFB=26THENR1=R1+ADD
554 B$="PLAYER ONE"+STR$(R1):GOSUB830:GOSUB840
555 POKESP,150:IFB=226THENR2=R2+ADD
556 B$=" PLAYER TWO"+STR$(R2):GOSUB830:GOSUB840
558 GOSUB840
559 POKESP,255
560 IFB=226THENB=26:B$=" PLAYER ONE UP ":GOTO575
565 POKESP,199
570 B=226:B$=" PLAYER TWO UP "
573 POKESP,220
575 GOSUB830:GOSUB840:BT$=B$:B$="      SERVE      ":GOSUB830:GOSUB840
576 POKESP,150
577 B$=BT$:GOSUB830:GOSUB840
580 POKESP,0
585 FORX=1TO20:POKEAP-10+X,ASC(MID$(A$,X,1)):NEXT
590 GOTO150
772 IFB=26THENR2=R2+25:GOTO780
774 R1=R1+25
780 IFR2>R1THENFORX=1TO30:PRINTTAB(X)"PLAYER TWO WINS":NEXT:GOTO800
790 IFR1>R2THENFORX=1TO30:PRINTTAB(X)"PLAYER  ONE WINS":NEXT:GOTO800
795 FORX=1TO30:PRINTTAB(X)"A TIE GAME!!!"::NEXT
800 PRINT" SCORE PLAYER ONE "R1:PRINT" PLAYER TWO "R2
810 R1=0:R2=0:TU=0:GOTO80
```

```
830 FORX=1TOLEN(B$):POKEAP-7+X,ASC(MID$(B$,X,1)):NEXT:RETURN
840 FORX=1TO600:NEXT:RETURN
900 FORX=1TO27:PRINT:NEXT
910 PRINT"I WILL PUT A TARGET":PRINT"IN THE CENTER OF THE
920 PRINT"SCREEN":PRINT"YOU SERVE THE BALL":PRINT"WITH THE REPEAT KEY
930 PRINT"THE LEFT SHIFT WILL PUT":PRINT"A / IN FRONT OF THE BALL  AND
940 PRINT"THE RIGHT SHIFT WILL PUT"
950 PRINT"A \ IN FRONT OF IT":PRINT"YOU STEER THE BALL BY
960 PRINT"BOUNCING IT OFF OF THE":PRINT"BARRIERS":
970 PRINT"THE IDEA IS TO HIT THE":PRINT"TARGET IN THE CENTER OF
980 PRINT"THE SCREEN":PRINT"ALL SLASHES EXCEPT
985 PRINT"A FEW HARD AGAINST THE":PRINT"SIDE ARE PERMANENT
987 INPUT"READY TO DISCUSS SCORE";A$:PRINT:PRINT:PRINT
990 PRINT:PRINT"YOU GET 16 POINTS MINUS":PRINT"THE NUMBER OF SLASHES
1000 PRINT"YOU USE TO HIT THE TARGET
1010 PRINT"BEST SCORE FROM A ROUND":PRINT"IS 15 POINTS AS THE
1020 PRINT"GAME DOESN'T COUNT":PRINT"ACCIDENTAL HITS FROM
1030 PRINT"LUCKY SERVES. IF YOU GET":PRINT"ONE YOU HAVE TO
1040 PRINT"SERVE AGAIN
1050 PRINT"THERE IS A 25 POINT":PRINT"BONUS FOR THE LAST HIT.
1060 PRINT"THE HIGHER THE DIFFICULTY":PRINT"THE LESS TIME YOU HAVE
1070 PRINT"TO SCORE":PRINT:GOTO80
1200 C1&C2=CORNERS:B=BALL:PN=POSITION NOW:PS=PEEK NEXT PN:MF=MOVE
1210 ADD=SCORE:R1&R2=PLAYERS SCORE:BT$TEMPORARY:AP=TARGET:W=WIDTH:L=LI
5000 POKESP,100:FORX=1TO15:NEXT:POKESP,0:RETURN
```

GIFT CERTIFICATES
(or see what they got!)

Limited space in last month's journal kept us from printing who was awarded gift certificates for what articles, so here goes ---

$60 certificates went to:
- Thomas Owens for video pokes
- Bob Woodward for the reverse scroll program
- Tim Walkenhorst for the disk buffer article
- N. Feliss for the parallel printer article
- Kerry Lourash for the E.Z. LISTER

$30 certificates went to:
- Charles Stewart
- Russ Terrell
- Curtis Preston

$15 certicates for the other letters that came up with hints, programs, or bright ideas.

This month, $60 certificates are going to:
- Edward Keating
- C.W. Aigeldinger
- Charles Stewart
- Todd Bailey

$30 certificates go to:
- Gary Wheeler
- Earl Morris
- Carl King
- Kerry Lourash
- Dick Sweet

O.K. We're trying again! On the last issue the response to the new format ran from "I love it" to "take pity on my poor bi-focals, so here is a semi-new format. We've kept the column format, but the print is done on our new EPSON and reduced about 20%. With the white background (the COMPRINT uses silver paper), the printer should be able to shoot a little darker so it will be easier to read. We would appreciate opinions on this style - did we hit it right this time?????

AARDVARK TECHNICAL SERVICES, LTD.
2352 South Commerce
Walled Lake, MI 48088
(313) 669 - 3110

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