SOME recent developments in the retail field... NCE/Compumart decided on Dec. 1 to drop the Bally line. They will fill all back orders and repair units. JS&A no longer shows the Bally in their catalog, and they expect to have their backlog substantially reduced by the end of the year, including repair. There is an article in a recent Business Week that indicates that Bally may drop the production of the Arcade®, but the factory denied this and a retraction is in progress.

THREE TONE MUSIC has been accomplished. I heard a tune the other day, and it is not bad. At the moment, the sounds are created by direct access to the chip via machine language, but we are getting indications of how to do it at the BASIC language level. Seems like &(n)=p yields some results when n varies from 16 to 23

ALONG the musical line, if the tone generator can be replaced by a unit like the one described in the November 73, we'll get realism. The article discusses a unit that varies the harmonic content and envelope of a note, therebye making it representative of an actual instrument.

A new dealer in my neighborhood is the Emporium chain, and also Home Cinema of Santa Clara and San Rafael. The manager of the Santa Clara store invites Arcadians to identify themselves for special consideration. They manufacture a rugged interface switch -the one that goes to the back of the TV- for use with the Bally.

FOUR COLOR SCREEN is now available, as follows: substitute the desired color number for the values n and p.
First off, split the screen with a vertical line by using &(9) = s
where s is 84 if you want the split in the middle
Use BC and FC to give the left side colors as in the book
Use &(2) = n &(3) = n to give the right side foreground
Use &(0) = p &(1) = p to give the right side background

$ gives access to the internal calculator. Not much known about that yet

THE ADD-ON now seems to be scheduled for full marketing for next year's Christmas season. The last inputs will be made after the January show, and production start in April.

APPARENTLY some of the handwritten material in the last issue did not reproduce well, so it has been typed and included here:

TRY THIS:
10 PRINT &(23)
20 GOTO 10
and run the program.
In turn press each key in the left hand row and see the
appropriate numbers appear

&(M)=N appears to set output decimal port #M to the value of N
and
N=&(M) appears to read the value of the M input port into the variable N.

Use this program for checking out your controller joystick operation to see that it functions in all positions

```
10 &(16) = 5x (1)
20 PRINT &(16)
30 GOTO 10
```

HERE are some more secrets from the depths of the Tiny BASIC . I am also reprinting a copy of a technical paper that has a lot of background information about the entire system. Sorry about the legibility in plaŝes.

A couple of comments on string constants. The @ must be used with (). Your example on pg.4 did not show this and may cause confusion for some. The characters produced with TV=@() duplicate for the most part, the ASCII decimal code. My unit starts out with 900 strings available, but decreases one string for every two bytes of program stored. (Therefor 900 strings = 1800 bytes)

Other commands

```
10 FOR A = -10 TO 10
20 B = ABS(A)
30 PRINT A,B
40 NEXT A
50 IF KP = "1" GOTO 70
60 STOP
70 ......
```

TBASIC has two additional commands not indicated in their manual. ABS() produces the absolute value;i.e., it changes a negative number to a positive. A STOP statement does just that, and can be placed anywhere in the program. I use the KP statement to place a temp. halt in the program. In this, depressing a '1' causes a jump to statement 70. Any other character and the program stops.

I've had some interesting results with th CALL () command most not worthy of repeating. Similar in some respects to &(13). One of the most interesting was the display of the large GAME OVER printout that occurs in the games portion.

You might have your experts play with the $. Try the following and other variations.

```
10 FOR A = 1 TO 256
20 $ = A,B,C
30 PRINT A,B,C
40 NEXT A
```

```
10 FOR A = 1 TO 256
20 $ = %(A),B,C
30 PRINT A,B,C
40 NEXT A
```

RECEIVED a question about the hand controller connector. It is an RS-232 connector, with nine contacts. One dealer is JADE Computer Products, of 4901 W. Rosencrans, Hawthorne,CA 90250. Part number DE-9S, at $2.15. In addition they have a cover at 1.25, to finish it off.

ARCADIANS
3626 Morrie Dr.
San Jose, CA 95127

First Class

BASIC Zgrass--A Sophisticated Graphics Language for the Bally Home Library Computer

Tom DeFanti, University of Illinois
at Chicago Circle
Jay Fenton, Dave Nutting Associates
Nola Donato, University of Illinois
at Chicago Circle

Abstract

Home computer users are just now discovering computer graphics. Modest extensions to BASIC allow plotting but not much more. The Bally Home Library Computer, however, has hardware to aid implementation of video games. Custom integrated circuits working on a 160X102 pixel (2 bits per pixel) color television screen allow certain forms of animation in real time. To give this power to the user, BASIC Zgrass has been designed and implemented. It is an extension of BASIC that allows parallel processes, picture objects that move, scale and group together as well as several drawing modes. There are also software controls of a three-voice music synthesizer, interactive input devices, a film camera and an IEEE bus interface. We will concentrate mainly on the language design for making it all easy to learn and use.

Content Indicators
1.51, 1.52, 2.12, 3.41, 3.44, 3.80, 4.13, 4.22
Keywords: interactive computer graphics, intepreters, real-time, graphic language, art

Introduction and General Motivation

Zgrass can be called an immodest extension of BASIC (Ted Nelson refers to it as "Super BASIC" [1]). It is an extension in that graphics support and user instruction facilities have been added to the capability of running BASIC programs copied out of a hobbyist computer magazine. It is immodest in that great liberties have been taken to weed out some of BASIC's undesirable programming conventions. Zgrass is actually a video game programming language designed specifically to encourage creation of beautiful animations in short order. It is also designed to teach many of the important concepts of interactive systems and 2-D computer graphics.

The hardware used is the Bally Home Library Computer built around the Bally Arcade unit. It is a Z-80 based machine with special integrated circuits which help the processor manage the 160x102 2-bit-per-pixel color output connected to any standard color tv set. The alphanumerics generated by program output and keyboard input are video mixed over the color graphics so text is not constrained to the 160x102 format. The two bits in each pixel indicate one of four bytes from which an index into a 256-element color map is taken. Thus, four of 256 colors can be on the screen at once. The hardware also includes some tricks which have been found useful in professional video arcade units. Its projected cost, without color tv, is about $750.00, including the arcade unit.

Zgrass itself takes up 16k bytes of ROM storage and has 16k bytes of RAM for user storage and system use. An additional 16k ROM (referred to as the 'extension' below) plugs into the side and provides room for a compiler and other features. Audio cassette storage and modem linkage to other computers is provided. Software for driving optional intelligent floppy disk drives is also built-in. Further hardware in this unit includes four hand controls, a 24-key pocket calculator keyboard, a three-voice sound generator and an IEEE Bus interface. Zgrass is the operating system for this computer as well.

Zgrass design concepts are rather different from those apparently underlying the current batch of home computer systems. Home systems are now trying very hard to be cheap minicomputers for expert users. These users when at home can be likened to the ham radio operators of the nineteen fifties, able to change diodes, violently shake intermittent boards and, in general, understand the innards. These persons can also get gratification from fighting with manuals and the trials of the latest software release, just as we professionals do for a living.

Zgrass, however is designed for the two-hour-a-week user. This type of person is guaranteed to continually forget the syntax and semantics of whatever software exists. Zgrass is designed (certainly at the cost of computer time and memory use) so the user does not have to rely on a manual to decipher everything. In short, this system is trying to be the Model T of the home computer industry, with all that implies. We shall see.

Above all, Zgrass is designed to be as attractive and as fun as pinball but considerably more intriguing and useful.

General Zgrass Concepts

"Right away" is the foremost design concept of Zgrass. Positive experiences in the first two hours of play are essential. When it is not part of your job or intended career, you must be able to do interesting, beautiful things right away, without reading a five-pound manual.

BASIC is now the home computer language. BASIC was designed as a teaching language and shines in its simplicity and easy matrix operations. It is, however, a poor language for the

manipulation of anything but numbers. It has no features that make writing large progams (over 200 statements) easy. Its subroutine capability is archaic. But, it is obviously a success in what the designers intended, otherwise it could not presently be the universal home computer language.

In particular, though, BASIC is a poor language for animation. To be sure, plotter art and decaying sine curves are well suited to BASIC. But our almost eight years of experience with the GRASS language[2] and Dan Sandin's color video Image Processor[3] as well as being directly or indirectly involved with roughly half of the most popular arcade video games, has given us insight into the potential of a color tv set, and you just cannot do a tv set justice in BASIC.

The next several pages will give details on syntax of Zgrass, but first we will further consider the "right away" criterion. Anyone who already knows BASIC should be able to write Zgrass programs immediately. For those persons in America who do not religiously read "BYTE" or "Creative Computing" (a sizable part of the population), we have been developing self-paced instructional software for Zgrass. The prototype system now functions quite well in GRASS (see paper by Towle and DeFanti, these proceedings). Both GRASS and Zgrass have enough supervisory functions and error trapping features to allow a good programmer/educator to write programs which execute and verify instructional programs written by beginning students. We feel that the chief problem in teaching programming is that it, as an activity, is poorly simulated by the examples and the Backus-Naur-type syntax information usually found in manuals. Normal programmers have fellow workers, consultants or at least other students around to help out. The home user has no such recourse.

The essence of the teaching problem stems from the fact that novices have tremendous problems with meaningless (to them) error messages. You really have to know nearly everything about a system before you can start knowing why what you have typed does not work. This is absolutely not an overstatement of the problem.

All this commentary leads directly to computer aided instruction. While CAI has not quite lived up to its expectations for teaching other subjects, it can be used to teach people how to program. It is relatively straightforward to lead the user through commands, interactively teach looping concepts and verify the results of carefully chosen problem sets. All possible errors can be trapped and explained in detail. Our experience with this type of teaching is very supportive--non-programmers (primarily art students and a few university officials) can be doing fascinating graphics of great beauty in half an hour. The whole process has the feeling of a game and is consistently rewarding. Providing internal system support for these teaching programs is no doubt the second most important design concept of Zgrass, after "right away" of course.

Note that few (if any) programming languages allow you to write programs to interactively teach programming. Smalltalk[4] and Logo[5] teach by user experimentation in surroundings not lacking professional help. Plato[6] has no student program or storage space (only authors can write and store TUTOR programs). One could probably design some CAI programs in Snobol or Lisp but these are hardly languages for novices and they are not known for their lucid error message handling, or availability on micros. Perhaps a graphics-extended APL might do. It would be wonderful if the experience with Zgrass encouraged others to design extensions to languages with teaching in mind.

Zgrass Technical Details

Zgrass is an interpreter of commands and assignment statements stored internally as ASCII strings. These strings may be entered and executed line by line, formed into programs (called "macros"), edited, and even built and pulled apart by string manipulation primitives. (A compiler which eliminates most of the interpretive overhead and which takes better advantage of the resident hardware floating point unit is part of the 16k extension ROM.)

We are determined to maintain compatibility with at least TINY BASIC (it is hard to say what "standard" BASIC really is). Since BASIC has line numbers, Zgrass labels must start with a number (e.g. 100, 1obekenobe, 707crash, etc.). A BASIC program copied out of "BYTE" would simply have a lot of extraneous labels. Labels in Zgrass are obviously not used to order and edit the statements as in BASIC since one would hardly like to have alphabetically ordered labels and Zgrass has a good on-screen editor anyway. Our definition of compatibility is restricted to executing perfect programs written in BASIC. Getting them in, editing, executing and debugging them is done differently.

Commands are made up of a keyword followed by zero or more operands. Examples of commands are:

```
goto 4jail
move deathweapon,x1,y1
clear
input dea,fbi,c
print  beep,"who loves ya, baby?"
```

(some of the more idiosyncratic BASIC commands, notably "if" and "for" have their peculiar syntax retained for compatibility).

Commands are gentle to users. If not enough arguments are supplied, if an incorrect argument is given or the argument is non-existent, a special error fixup routine is entered. This routine prints out the command in error, points at the argument in error and says, for example, "NO! this command wants a variable name here." The user can then type in a correct argument and the command goes on. He can also elect to enter command mode to create a missing name, for example, and then resume the above process. All this will happen whether commands are entered line-by-line or executed as part of a macro.

Note that all commands are more or less self documenting. You can type the command name and it asks you for the operands. This error facility allows one to get by trivial syntax errors without constant re-editing. It is also a sloppy way to get input to macros, although there are several conceptually clearer (to computer folk) ways.

Many commands have options indicated by postfixing the command name with a hyphen plus a modifier (e.g. "input-string" which can be shortened to "in-str" or even "i-s"). The hyphenated option construction is more English-like than

single-character switches and it helps keep down proliferation of command names.

Since every command has an internally stored list of what argument types it wants, the "help" command can easily print these out with options (there are about 20 different argument types in Zgrass, like number, string, expression, picture prototype, array and so on). Further syntax and semantic information is available in the manual but the information you need most often is at your fingertips "right away."

A few more details about variables are necessary. Variable names (macros are actually string variables) can be any length and must start with an alphabetic character. Variables used as macros may not have names which conflict with system names. Global variables start with lower case letters and local variables start with upper case letters. The system decides whether a variable is a numeric or string variable by examining the context in which it is first used.

(Commands are terminated by semicolons or carriage returns. The alphanumeric generator which is video mixed over the color graphics or routed to a separate monitor puts up sixteen 32-character lines. The alphanumeric handler automatically folds lines over 32 characters long for display purposes but does not insert a carriage return. A special character indicates folded lines.)

Arithmetic statements are similar in format to assignment statements in BASIC or FORTRAN, with the exception of the left arrow used. The parser automatically changes equals signs (which are used for conditionals) to left arrows to maintain compatibility with BASIC yet allow a sophisticated expression evaluator to operate unambiguously by not having to deal with multi-purpose operators. Examples of arithmetic assignment statements are:

```
abc←sin(arg1)+cos(arg2)
babarum←1.2
c←whodunit(f,g,huh)
```

where the last example contains a user-defined function (a macro, of course).

Numeric variables are kept in fixed or floating point by the system, switching mode as necessary without user knowledge. The luxury of a floating point arithmetic unit helps calculation speed considerably. There are also reserved system variables which hold the values of the hand controls, keypad keys, external I/O strings and other special things.

Strings are assigned as follows:

```
tom←"this is a single line"
sam←tom&tom&babarum                ;.concatenat
cr←'
'                    ;.to enter a carriage return
mymacro←<print tom,cr,tom," again"
        stuff←<hello>    ;. stuff gets "hello"
        for a=1 to 10
        print stuff,a
        next a>
```

The last example shows nested assignments and a way to create a macro. Note that acceptable string delimiters are ",',<,>,[, and], the last four of which must be balanced. String decomposition is handled by a variety of string commands in the 16k extension.

Macros

Any string can be executed in Zgrass. If it contains meaningful commands, it can be used as a program. Again, programs in Zgrass are called "macros." Macros can call other macros, call themselves, execute in foreground or in parallel with other macros in the background, supervise other macros and interact with macros running on other Zgrass machines.

Zgrass differs from BASIC greatly when it comes to subroutine linkage. Zgrass has a very convenient and conceptually clear way of passing arguments--you simply make believe the macro is a command and use standard command syntax. Furthermore, the "input" command (for numbers) and the "input-name" and "input-string" commands (for strings) fetch the arguments passed in a way that automatically request user input from the terminal if not enough arguments are passed. This linkage is discussed in more detail below.

Looping and control transfer is done with "goto," "for/next" and "gosub." (Gosub is retained for compatibility.) A version of goto called "skip" has the option of jumping relative a number of lines, a good feature for those quick loops in which you forgot to put the label. There is also a "return" command which can pass an argument back if it is a function call like "whodunnit" above.

When a macro is asked to execute (by typing its name as the first thing on a line, like a command), the system builds a macro invocation block (MIB). The MIB holds information about local variables, for/next blocks, argument and data lists and so on. When the macro is done, the MIB is deleted along with all the pieces hanging off it. The storage allocation/reclamation algorithms used are again similar to GRASS's.

Numeric arguments are passed using the "input" command. It is similar to "input" in BASIC but it first checks the argument list. If there is an argument, it is grabbed. Otherwise, the user is requested to enter the value. Input works in concert with "print" which supresses output when arguments are present. (Of course, there are options to input and print which force terminal I/O.) Thus, a macro that has lots of prompting information can be called by another macro without all sorts of editing to remove the prompts, but the macro will wake up and start asking questions if not enough arguments are supplied. One can create self-documenting macros which are efficient, yet help out when necessary. (How many times have you forgotten the arguments to a subroutine?)

The "input-name" command functions similarly for strings and is used to get names passed as arguments. You use the "@" operator to indirectly reference strings. For example:

```
getandmove←<1again print "gimme a pixname"
        inp-name gettemp          ;.get name
        if gettemp='',return      ;.if null, return
        get-tape @gettemp         ;.get it from tape
        move @gettemp,x3,y3       ;.attach control3
        display @gettemp          ;.and show it
        goto 1again>
        .
        .
        .
        getandmove apple,witch,titles,,
```

where the last line is the call and the lines above are the macro definition presumably entered before the call. Comments are lines or sub-lines which start with a "."

The "input-string" command expects string delimiters around arguments passed so that whole commands can be arguments (you cannot pass a comma with the input-name option). This is important for the construction of teaching and verification programs, among other things. If the argument is not there, input-string will require input from the keyboard, but without the string delimiters, just like input-name.

The principle here is to make macros look like system-defined commands as much as possible, a rather loose definition of extensibility, but one that is meaningful when speaking about interpreters (a conclusion adapted from comments in [7]).

Picture and Pattern Drawing

Zgrass has several predefined variables which cause drawing on the screen when they are written into. To display a point, you set variables xs and ys to the x and y coordinate. When you set variable cs to a value from one to four, the point is displayed with the color value indicated by the value of cs. There are also the "line," "box," and "circle" commands which draw vectors, and filled rectangles and ellipses on the screen.

Picture Animation

Once a picture is drawn on the screen, all or part of it may be stored as a picture prototype with the "snap" command. Picture prototypes are pixel lists in this case and are kept in user 16k RAM rather than in the screen 4k RAM. So, a picture is something you see on the screen and a picture prototype (or "prototype" for short) is its representation in user memory.

One then attaches the prototype to a variable, time-based variable (a user-defined special variable whose value varies automatically over a given time period), hand control, or user-defined function. The prototype is displayed with the "display" command. After that, anytime the variable or function changes, the prototype will be erased with an "exclusive or" write and rewritten with its updated translation or other transformation with an "exclusive or" write. This technique lets pictures of one color pass over pictures of another color without leaving holes (using standard bit plane raster scan graphics techniques). There is an interrupt level routine which manages movements of prototypes and a host of other details, along the lines of a conventional refresh graphics driver. The user has a simple way to indicate what maximum percentage of time should be allocated to interrupt level updating, the rest being left for command processing.

The same interrupt routine also manages picture prototype lists made up of lists of points, vector endpoints, box and circle drawing information. With the 16k extension, rotation and scaling of these lists is possible. Prototype lists, of course, have to be built up rather than snapped using options to the line, box and circle commands. The user can access individual endpoints and manipulate the prototype as a whole. Of course, there can be many prototypes floating around at once.

Again, the user's aesthetics and perceptions are essential. Since everything is real-time (or close to it) the user can easily see the effect of various commands on the screen, the implications of varying amounts of interrupt processing time, the ways of using different colors, and so on.

Prototype lists and patterns can be grouped into a tree structure which allows concatenation of transformations. Moreover, the "select" command specifies a sequence of pictures to be put up and erased in round-robin fashion (imagine several views of a walking "man" being switched to provide the illusion of walking). A simple Super 8 camera hookup allows more complex, synchronized sequences to be filmed, if desired.

Conclusions

Zgrass is designed to be a first programming language which encourages both novices and experts to learn about color graphics, generate meaningful and pretty displays, possibly make Super 8 movies, and perhaps even access governmental databases and control electric train sets. The software is designed to be multi-leveled and rich with feedback. Considerable research into the teaching aspects has been folded into the design.

Continuing developmental effort along these lines now concerns higher resolution displays, much faster microprocessors, parallel programming techniques and connection to videodisks and other television equipment. Zgrass is one way you can control the amount of sex and violence on your tv set.

References

[1] Nelson, Ted, The Home Computer Revolution, 1977, p. 79.

[2] DeFanti, T.A., "The Digital Component of the Circle Graphics Habitat," Proc. NCC, 1976.

[3] DeFanti, T.A., Sandin, D.J., and Nelson, T.H., "Computer Graphics as a Way of Life," Computers & Graphics, Vol. 1, No. 1, May 1975.

[4] Goldberg, A. and Kay, A., Smalltalk-72 Instruction Manual, Xerox PARC #ssl76-6, March 1976.

[5] Papert, Seymour, A Computer Laboratory for Elementary Schools, Logo Memo 1, MIT Artificial Intelligence Lab, October, 1971.

[6] Alpert, D., and Bitzer, D., "Advances in Computer-Based Education, Science, Vol. 167, March 1970.

[7] Feldman, J.A., "Proceedings of the Extensible Languages Symposium," SIGPLAN NOTICES, Vol. 4., No. 8., August 1969.

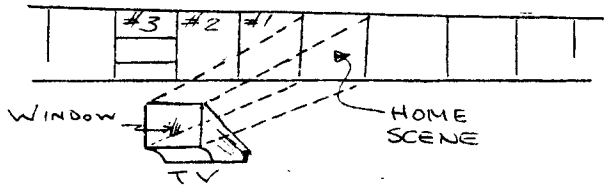
Addendum: Zgrass Command List (in addition to BASIC)

(The commands beginning with a '*' are in the 16k extension.) (Not all options are indicated.)

IN DEPTH NON-PROFESSIONAL VIEW OF &(9)

If you set up a loop FOR A= 1 TO 200; &(9)=A; NEXT A you will see a vertical line wiping the screen from left to right, color changes, and sometimes a blanked-off area at the top and bottom. I have been doing a little research with the following results.

The world of &(9) can be envisioned as a panorama of scenes laid side by side, with a "Home Scene" in the center. The Home Scene is the one we see thru the TV "Window" when we normally use the TBASIC. Spreading to each side are a number of scenes, the first few of which will be described. Each scene has the same dimensions as the TV Window, something like:



If you set &(9)=Ø, you will view the first scene, with the background in color. In this scene, the color controls FC and BC do not work. Instead, &(11) gives the background and &(Ø) gives the foreground. This is #1 above.

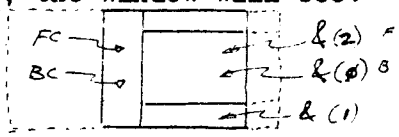
If &(9)=37, you will see another scene, #2 above, but it is now like the Home Scene in that FC and BC again operate. These types of scenes alternate as a general statement. The next in line, #3 above, is set at &(9)=63, but there is now a border above and below the text area of the screen - and &(2) provides the foreground color while both &(Ø) and &(11) will give the background, and &(1) controls the border.

We can keep moving sideways, jumping from scene to scene with the following generalities: The width of a scene is either 26 or 38 "numbers" wide. The numbers are the values of A, so that setting A equal to 63,1Ø2, 127, etc, will give you a full view of the various scenes. Actually there seems to be a tolerance of ± 1. The scenes alternate between white backgrounds and colored backgrounds. The white backgrounds have the 26 number wide scenes. And the same kind of story takes place in the -A direction.

Color control in the 'odd' scenes is inconsistent, as the following table shows:

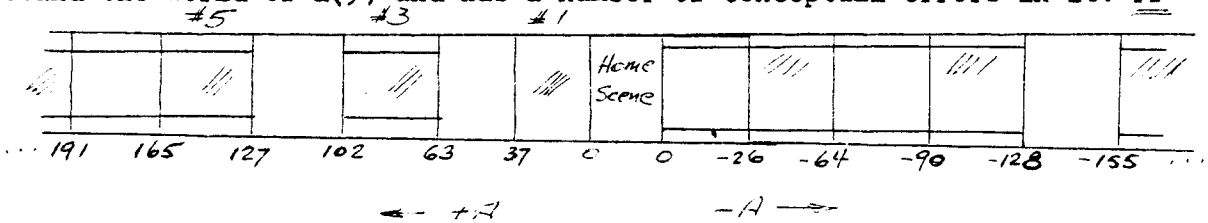
SCENE:	#1	#3	#5
FOREGROUND	&(Ø)	&(2)	&(2)
BACKGROUND	&(11)	&(11) or &(Ø)	&(11) or &(Ø)
BORDER	n/a	&(1)	&(2)

If &(9) is set to some intermediate number, say &(9)=70, you will see parts of two scenes at once, and since adjacent scenes have a different color control scheme, you can get multicolors on the TV Window. With the above example, the Window will see:



The words written on there are written on the Window, overlaying the split screen.

Caveat - the description of scenes is a personal attempt to understand the world of &(9) and has a number of conceptual errors in it. rf



EXECUTIVE SOFTWARE data is now available from me at $1. to cover printing and postage. This is of value only to those conversant with assembly language. It was developed by Tom Wood and provides internal locations of most of the data , subroutines, etc., that programmer would find useful.

HERE is a portion of the ASCII code, as it pertains to the Bally. Use @(n) where n is the number in the Decimal column.

Character	Binary Bit 7 to Bit 0	Octal	Decimal	Hexadecimal
SP	00100000	040	032	20
!	00100001	041	033	21
"	00100010	042	034	22
#	00100011	043	035	23
$	00100100	044	036	24
%	00100101	045	037	25
&	00100110	046	038	26
'	00100111	047	039	27
(	00101000	050	040	28
)	00101001	051	041	29
*	00101010	052	042	2A
+	00101011	053	043	2B
,	00101100	054	044	2C
-	00101101	055	045	2D
.	00101110	056	046	2E
/	00101111	057	047	2F
0	00110000	060	048	30
1	00110001	061	049	31
2	00110010	062	050	32
3	00110011	063	051	33
4	00110100	064	052	34
5	00110101	065	053	35
6	00110110	066	054	36
7	00110111	067	055	37
8	00111000	070	056	38
9	00111001	071	057	39
:	00111010	072	058	3A
;	00111011	073	059	3B
<	00111100	074	060	3C
=	00111101	075	061	3D
>	00111110	076	062	3E
?	00111111	077	063	3F
@	01000000	100	064	40
A	01000001	101	065	41
B	01000010	102	066	42
C	01000011	103	067	43
D	01000100	104	068	44
E	01000101	105	069	45
F	01000110	106	070	46
G	01000111	107	071	47
H	01001000	110	072	48
I	01001001	111	073	49
J	01001010	112	074	4A
K	01001011	113	075	4B
L	01001100	114	076	4C
M	01001101	115	077	4D
N	01001110	116	078	4E
O	01001111	117	079	4F
P	01010000	120	080	50
Q	01010001	121	081	51
R	01010010	122	082	52
S	01010011	123	083	53
T	01010100	124	084	54
U	01010101	125	085	55
V	01010110	126	086	56
W	01010111	127	087	57
X	01011000	130	088	58
Y	01011001	131	089	59
Z	01011010	132	090	5A
[	01011011	133	091	5B
\	01011100	134	092	5C
]	01011101	135	093	5D
^	01011110	136	094	5E

SERVICE MANUAL PA-1 by Bally is now available from me at $1. to cover mailing. Not too detailed but it does have the schematic, a parts list, and a parts layout.

CURRENT PRICING of the Videocades is as follows:

19.95 - 2001	24.95 - 2003
2002	2005
2004	3001
3003	3002
3004	4002
4001	5002
5001	
5003	49.95 - 4003

Note that the number of games has no relation to the price.

LIST of all characters and commands is generated by this:

```
FOR A = 0 TO 120
TV=A
@(10)=A
NEXT A
```

The characters at positions 100 to 103 have no meaning and are used to fill unused spaces.

ARCADIAN

Robert Fabris, Staff
3626 Morrie Dr.
San Jose, CA 95127

FIRST CLASS