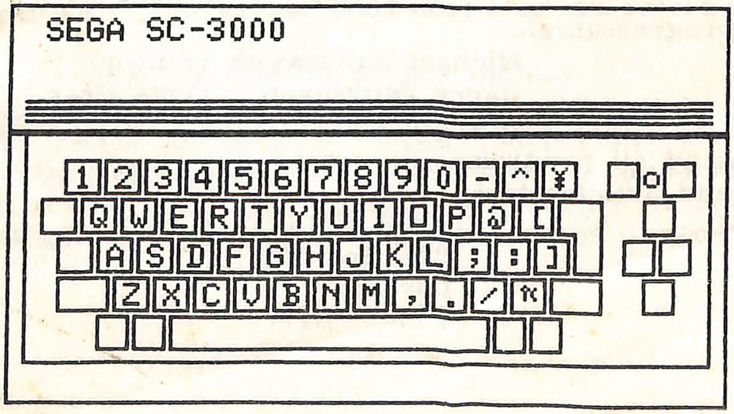
SEGA ADDICT

The Official Magazine of
the Vic Sega Users Group

Vol. 4 No. 5



INKEY$:

Hello and welcome to another ***SEGA ADDICT***. In this issue you will find another of Gwen Woods delightful musical, graphics programs. If you enjoyed Gwen's version of 'At the Ballet' then you'll love her three 'Russian Dancer's'!! Also inside you will find an article from the ever-present Julius Guest. Julius takes the mystery out of a neat little program called "ADDRESS.BAS" which is supplied with your origional Disk BASIC. For those of you who may be interested about what is inside your Sega's black plastic case, take a squizz at this month's *WORKSHOP* for an anatomical view of your beloved computer.

Disks containing all of the programs that have appeared within the pages of *SEGA ADDICT* (right back to Vol. 1, No. 1) are now available at the regular V.S.U.G. monthly meetings. These disks may be copied at the meetings for a small charge (you will need to bring along 2 blank disks). This service will save you the effort of entering and debugging the fantastic listings that are found inside your favorite mag. There is also a version of modified Disk BASIC (containing all of the extra control codes and SMART-KEY) available at the meetings.

PROGRAMS PROGRAMS PROGRAMS PROGRAMS PROGRAMS

DON'T FORGET...We need your programs to publish!

(NOTE: *All programs are accepted in good faith and should be accredited with the original authors name)*

Happy programming...

Michael Nanscawen (Editor)
Garry Jenkinson (Vice Pres.)

Please send all programs
and articles to:-

SEGA ADDICT
P.O. Box 102
Doveton
Vic. 3177

SYNTAX:

...Towing the LINE...

or

...making a BeeLINE for home...

Many of you will think that I have gone somewhat potty (*who needs to think it, many already know??...ED*) or that I have a page or two missing from my instruction manual once you realise that this issue's treatise for *SYNTAX* is a discourse upon the BASIC instruction ~ **LINE** ~ and it's counterpart ~ **BLINE**! What happened to **HCOPY** and **CURSOR** (*and why is BLINE being discussed in 'L' instead of 'B'??...ED*)? Answers to all of these questions (and more) will be revealed (*get out your magnifying glasses...ED*) in the words which follow....

....Covering one's tracks....

Firstly, nothing has happened to either of those BASIC instructions mentioned above. That is, unless you happen to be using the alternative to Disk BASIC ver. 1.0p, **GBASIC** and IIIA or IIIB Cartridge BASIC (none of which support **HCOPY** in a format for dumping SCREEN 2) and as for the other neglected instruction, **CURSOR**, sufficient words have been expended within these pages upon that subject, leaving little or no more to add!! So there!!! Whether you like it or not, **LINE** is the thesis for this issue (*the judge's decision is final and no further correspondance will be entered into!! Mind you, we don't mind* ***some*** *mail on occasions...ED*)!!

....Twelve to the inch....

The BASIC instruction ~ **LINE** ~ is perhaps the simplest of the numerous statements which can be understood and utilised by the BASIC programmer (although it is not without it's pitfalls). Let us peruse the contents of program #1 and then examine it's relevence to this chapter of *SYNTAX*....

```
10 REM * PROGRAM #1 *
20 SCREEN 2,2:CLS
30 COLOR 1,15,(0,0)-(255,191),15
40 COLOR ,5,(56,50)-(135,110)
50 COLOR ,4,(80,80)-(95,110)
60 COLOR ,7,(64,70)-(79,96)
70 COLOR ,7,(112,70)-(127,96)
80 LINE (56,50)-(135,50)
90 LINE (135,50)-(135,110)
100 LINE (135,110)-(56,110)
110 LINE (56,110)-(56,50)
120 LINE (82,82)-(93,108),10,B
130 LINE (85,85)-(90,90),,B
140 LINE (85,90)-(95,105),,B
150 LINE (64,70)-(79,96),1,B
160 LINE (112,70)-(127,96),,B
170 LINE (72,70)-(72,96)
180 LINE (120,70)-(120,96)
190 LINE (64,78)-(79,78)
200 LINE (112,78)-(127,78)
210 LINE (128,100)-(135,30),6,BF
220 LINE (48,64)-(96,20),12
230 LINE -(128,50)
240 GOTO 240
```

....wherein you will find that I have made use of every possible *SYNTAX* requirement compatible with the **LINE** statement. However, before I take you on an unexpurgated romp through this program, I believe you are entitled to be informed as to what these requirements are and how they can be used (so that, at least, you cannot accuse me of misleading you)

....Co-ordinating your Plan....

It is with the utmost conviction that I realise that many of you have observed from the contents of PROGRAM #1, that there is an enormous similarity between the SYNTAX for **COLOR** (which was the subject of discussion in the previous volume of this tome) and what is required for the needs of the small but significant topic currently under scrutiny!! In fact, PROGRAM #1 is as much a 'tour de force' for **COLOR** as it is for **LINE.** All of this notwithstanding, let's see what is needed to keep the 'old' Syntax error (or worse still) Statement parameter error messages at bay.

Firstly, though YOU should not need reminding, the program must be 'writing' to SCREEN 2. It may seem absurd to emphasise this small detail but many is the time *I* have attempted to **COLOR** an area in the background of the wrong screen, only to incur the ire of BASIC. As some of you may have gathered from the subheading of this section, the most important craving of this instruction is no more than the co-ordinates of the points which constitute the starting and ending points of the **LINE** to be drawn. Naturally, by this stage of your development, I would expect you to realise that these co-ordinates are specified as 'x co-ordinate, y co-ordinate' within the ever present (as nearly so as 'comma') brackets in the following format....

```
LINE (x,y)-(xx,yy)
```

....where 'x' and 'y' are the starting co-ordinates and 'xx' and 'yy' are the ending co-ordinates of the line being drawn. Before I expound upon the optional aspects of this instruction, it is imperitive that these co-ordinates lie within the confines of the limits of your SEGA's SCREEN 2. Should you have forgotten (*heaven forbid...ED*), the 'x co-ordinate' must reside betwixt 0 and 255 and the 'y co-ordinate' must not extend below 0 nor be greater than 191, irrespective of the point of the line which is being defined. Should you be remiss with your values in this area, you will find yourself at the mercy of the (aforementioned) **Statement parameter error** message, where upon you are left to your own devices to alleviate the problem (*Oh, the ignominity of it all...ED*).

Now that we have established where our line is and where it's going, we can take a more lesuirely glance at the various options which now become available to the programmer. Taking into account the eternally present delimeter (your friend and mine, com,ma), the next most important decision the programmer must make is what value to put into the next parameter for this statement. This is, of course, (*as if we didn't already know...ED*) the desired 'writing' colour for the LINE being drawn! As I indicated in last issue's ramblings, this value must lie within the range 0 through 15 (where each value correxponds to the code of the intended colour), lest you be thwarted by that confounded 'Statement parameter error' yet again!!!

....Choc-a-Bloc....

The next of the optional variations which can be applied to this statement is perhaps the most versatile. In actual fact, many of you will notice that it bears a striking resemblence to the last of the parameter selections which accompany the CIRCLE statement. Dare I say, that it would not surprise those of you so observant, that this option performs much the same function!! However, where that parameter might be considered a 'Boundary' or 'Boundary Fill' option, terms appropriate for their function with CIRCLE, the nomenclature 'Box' and 'Box Fill', when they are applied to LINE, allow the programmer a subtle distinction between the purpose of what is otherwise a potentially confusing similarity. Whether *you* make this differentiation or not, the addition of the character 'B' after the COLOR specification (not forgetting the comma ~ *would any of us do that...ED?*) will cause your BASIC to reproduce a BOX upon the SCREEN, where the top left hand corner corresponds with the starting x and y co-ordinates and the bottom right hand corner is the same as the co-ordinates supplied for the ending x and y points. Attachment of the 'Fill' specification (in the required form ~ BF ~ allows the programmer the facility to 'block-fill' the area created by the Box upon which I have just deliberated. All of this aforementioned activity will take place at none other than the specified COLOR.

For example, if your LINE is drawn in cyan (COLOR CODE 7), then the BOX will be outlined in that hue and, without exception, 'BF' will fill that box with the same colour. This leads me to dispel the idea that it is absolutely pertinent for the BASIC programmer to provide the COLOR specification (to the LINE instruction) upon each and every occasion upon which it is used. If you desire to draw a following LINE of your program in the same colour as that allocated to the previous LINE, it is not necessary to repeatedly tell your SEGA to do so! One of the most pleasing capabililties of our SEGA's interpreter is it's ability to remember any specified 'writing' colour (those amongst you who follow *WORKSHOP* as well as *SYNTAX*will appreciate how this capability is effected) and once the SEGA's interpreter has committed this to memory, the BASIC programmer need not specify a further COLOR value, unless it be that of another desired pigment. This facility applies irrespective of the BASIC statement under scrutiny (as long as it is one of those that requires this parameter specfication). The ommission of this secification (when it is not needed) is accomplished by the simple expedient (in ALL cases), of the programmer failing to provide an appropriate value where it is required (as I have done in line 230 of PROGRAM #1).

Of course, it goes without saying (*not where you're concerned...ED*), that the 'presentation' of the co-ordinates, as indicated so far, presupposes that YOUR line 'runs' from left to right and from top to bottom! What I have thus far failed to make public (and I think the writers of our illustrious instruction manuals also fail to stress) is that the SEGA's BASIC interpreter is clever enough to realise that the co-ordinates which you have supplied (should you actually supply them back-to-front) *are* 'topsy turvy' and will thus carry out your wishes regardless, without so much as turning a hair. If you haven't tried it yet, enter this direct input statement....

```
SCREEN 2,2:CLS:LINE (255,191)-(0,0)
```

....and after the 'flash' to SCREEN 2 that you see, if you press [CTRL] + [O] or [SHIFT] + [BREAK], you will observe that there has appeared upon the graphic screen, a line that 'runs' from the top right hand corner to the bottom left hand corner. Isn't it a clever little SEGA (*well, shuuut my mouuuth and caaall me Cyril...ED*)?????!!!!

....From hither and Yon....

Lest you should think that I have fully extolled the complete virtues of the story of LINE, you will be reminded by a quick perusal of PROGRAM #1 to dash any misconceptions upon those lines (pun intended!!) There is more to this otherwise humble instruction than meets the eye (though everything it does you see eventually)!!

As if the facility to make a BOX from any set of given points (or to fill that box, should it be so desired) does not sufficiently mitigate the complexities of the use of the BASIC instruction currently under discourse, for even the most unknowledgable BASIC exponent, the writers of our own dialect of this formidable computer language managed to find a few more tricks up their sleeves! What do you my disciples (*that's not the sort of 'faith' I had in mind when I asked you to take over from Ron...ED*) make of the contents of PROGRAM #1's line 230?

The discourse thus far would have you believe that the SEGA's BASIC interpreter will always warrant the starting point and ending point of any LINE which it is instructed to draw. Fortunately, for numerous reasons, not the least of which is the inherent lazy (read "how can I make it easier for me do this") attitude of HOMO SAPIENS, the necessary room was found within the memory space occupied by BASIC to allow the programmer the small, but very significant facilty to specify the point to which he/she would desire to dispatch the next LINE, without the need to nominate the originating point of that LINE. This is the explanation of the seemingly incomplete instruction in line 230!!

Unfortunately, there are a number of situations (outside of the LINE statement) that will affect the outcome (or is that 'income') of this abbreviated version of LINE. These are the yet to be discussed (in this article) **BLINE** and the ("I'll leave them for another issue!") instructions, **PRESET** and **PSET.** How all of this gibberish makes the task of the BASIC programmer simpler is extremely easy to explain. When the originating point of the specified LINE is omitted (as is the case in line 230), the SEGA's interpreter will automatically assume that the new line to be drawn will START from the finishing point of the previous LINE's parameters. This neat little trick can also be accomplished even if the previously specified LINE isn't (*what nonesense is this...ED?*)!!

....Writing with invisible Ink....

I wonder how many of my devoted votaries understood the programming implications behind the concept of BCIRCLE, about which I articulated in ***SEGA ADDICT*** last issue (Vol. 4, No. 3)?? Those of you who have grasped this (not very complicated) abstraction, will be pleased to discover that the same capability can be applied to SEGA's **LINE** instruction. In simpler layman's terms, the specification of **BLINE** (followed by the same necessary parameters [with certain restrictions], used in the identical manner) will cause the LINE that you nominate NOT to be drawn. That is, it appears in the colour of the background 'behind' where it should be drawn. As is the case with BCIRCLE, BLINE can be used to expose an area within an already 'block-filled' section of the screen. "Why would I want to do this?", I hear you all asking. Well may you ask if you had not encountered something similiar to the following code within a program....

```
LINE (50,50)-(120,100),1,BF:
COLOR15:CURSOR 70,60:PRINT
"hello"
```

....and if you fully grasped the 'simple complexities' of the workings of the SEGA's Video Display Processor (as I expounded them in the last issue), you will realise that the results of such code will not be those that are desired. The reason is because the PRINT statement is attempting to turn 'ON' the bits of the VDP pattern data to write the message to the screen upon an area of (VDP) memory which has already had all of it's 'foreground' bits SET by the LINE statement. LINE literally draws a line onto the screen (all the pixels that from the LINE are 'ON'). When 'BF' is specified, all of the pixels within the enclosed Box are also switched active. Thus, although a different COLOR has been set for the message to be displayed, there has appeared on the screen an unintelligable mess where the word 'hello' should appear.

There is a way to rid ourselves of this mess (*have you tried sweeping it under the carpet...ED?*)! It is problems of just this sort where our newly found friend **BLINE** shows it's true colours. Well, when I say 'true colours', this is not strictly possible!! You see, because we are in fact NOT drawing a line, there is no need to specify a colour parameter for that purpose. Actually, any colour specified to the **BLINE** instruction will be ignored by the BASIC interpreter. That is, until you attempt to draw a subsequent **LINE** without specifying it's colour. If the colour code specified to the 27,69 BLINE instruction differs from that which was initialised by the previous **LINE**, the BASIC interpreter will ignore YOUR intention and reproduce the LINE in the colour specified by BLINE. Getting back to the point in question, what actually happens (with **BLINE**) is that the VDP is instructed to turn 'OFF' each of the pixels that constitute the make-up of the specified BLINE and consequently, the colour of the BACKGROUND is what is seen as the colour of the line presented to the observer!! So how is it that we can correct the problem created by the above code??

To start with, it becomes necessary to create an area behind the message so that the pixels of the 'foreground' are turned OFF, thus allowing **PRINT** to do it's thing as intended. Let's make a program of our little snippet of code and see what magic can be wrought....

```
10 REM * PROGRAM #2 *
20 SCREEN 2,2:CLS:COLOR ,1,15,
(0,0)-(255,191),15
30 LINE (50,50)-(120,100),,BF
```

```
40 COLOR 15:CURSOR 70,60:PRINT
"hello"
50 END
```

So, the first thing to do is to add the instruction to clear the area behind the message. To do this, we can introduce the additional line....

```
35 BLINE (70,60)-(100,68)
```

....and after the program is reRUN, all of our problems should be solved!! Well, those who have done so will find that, in fact, this addition seems to have created more problems than it has solved. What happened to the message??? It certainly seems to have dissappeared, doesn't it??? Actually, it is still there, as a very careful study of the program will show that it must be. It is not visible because the program is trying to PRINT "hello" in white ink on white paper!!! There are now a couple of paths along which we can travel to reach our desired goal. The first of these thoroughfares will prompt us to change the writing colour of the message. This is all good and well, but what dilemma does this create if our original purpose was to display the word 'hello' in white within a black block (*that's what I thought you were doing...ED*)? Clearly, this solution is at odds with our purpose and it is ill advised to take this route. If you actually do change the writing colour of the message (by altering the value to the COLOR statement in line 40), you will observe that the message is still there, as I have intimated!!! However, our desire is to have our message **PRINT**ed white (upon a black background, what's more). How is this conundrum assuaged? To solve our problem, we must once more revert to the contents of *SYNTAX* Vol. 4, No. 3, where my comments about the execution of the COLOR statement will come to our rescue (*like a 'white' Knight on a 'white' Charger, eh???...ED*).

Remembering that **COLOR**'s purpose in life is to provide the specified area with the specified 'BACKGROUND' colour, by providing that colour to the OFF bits of the VDP colour pattern data, our suffering can be abated somewhat with this simple addition to line 35....

```
35 BLINE (70,60)-(100,68):COLOR
,1,(64,60)-(104,68),15
```

....Reaching the end of the LINE....

Sorry about the above reference to my usual mode d'employ, but I couldn't resist the temptation (I've always been a sucker for a good 'double en'tendre)!! There isn't a great deal more that I can add to the rubbish (*I'm glad YOU said that...ED*) that you have had to wade through thus far but one small detail causes me some concern and I can't think of any better place than now, to bring it up (you've been at the Port again, haven't you???...ED)!

Some of you (the more observant perhaps) MUST have noticed that the co-ordinate values that I provided for the **COLOR** instructions in program #1 are located at the required multiples of 8 which I had stressed should be done with this statement in the aforementioned episode of this tome. This has caused the values that must be thus supplied to the **LINE** statements which follow it to appear to be somewhat haphazard. This apparent irrationality is necessary to reduce to a minimum the occurance of the most common programming error attempted by the BASIC programmer. This is the same mistake that I have already discussed (in the above mentioned episode of this series), whereby the programmer tries to tell the SEGA's interpreter to demand more than TWO colours within any ONE byte of the SEGA's VDP's colour pattern data. It is simply IMPOSSIBLE to have any more than two colours here, the LOGIC of BINARY should be sufficient to prove otherwise.

After all of this rambling, I don't think that there is very much more that I can add to this recitation, besides which, ABC FM (the radio station which provides the 'muzak' [sorry!!] that keeps my thought processes perpetually coherent [*so that's your problem...ED*!!]) is about to offer Alfred Brendel's rendition of Beethoven's "Waldstein" Piano Sonata and one must get one's priorities right. See you in the next issue...

CLUB NOTES FOR VICTORIAN SEGA USER'S GROUP INC.

FOLLOWING SUCCESSFUL COURSES IN BASIC AND MACHINE CODE CONDUCTED OVER THE PAST FEW MONTHS AT THE FRANKSTON CHRISTIAN SCHOOL MORE DATES HAVE BEEN SET THE NEXT DATES ARE;NOV 11TH & DEC 16TH

SHOULD THERE BE ANY INTERESTED PEOPLE WHO WOULD LIKE TO ATTEND THESE COURSES PLEASE CONTACT ;

STEWART CROWTHER PH:783 7328 A/HOURS
OR
GARY JENKINSON PH:309 7130

****

CHANGE OF MEETING DATES AT Y.W.C.A.
NOVEMBER 6TH DUE TO THE MELBOURNE CUP
TO NOVEMBER 13TH 7.30 P.M.

###

BACK COPIES OF MAGAZINES ARE STILL AVAILABLE & SHOULD YOU REQUIRE THEM PLEASE CONTACT DEBBIE CROWTHER ON 783 7328 AFTER HOURS

PEOPLE TO LAZY TO TYPE THE PROGRAMS FROM ALL THE MAGAZINES PLEASE CONTACT EITHER STEWART OR GARY TO ORGANIZE TAPE VERSION OR DISKS

THE COST OF THIS SERVICE IS $15 (GROUP DISK)
$5 (YOUR DISK)

THIS COST IS FOR EACH MAGAZINE YEAR

A CASSETTE SERVICE CAN BE PROVIDED
PLEASE CHECK PRIOR TO ARRANGING THIS

> >

TOP THE TOPS:

Look, up in the sky! Is it an ORGROID? Is it EXERION?? Maybe it's just SUPERMOUSE!!! Whatever it is, we want to see your best efforts at getting the best out of these (and other) GAMES. Any better effort will be acceptable, but those verified by a PHOTOGRAPH ***, will surpass all others.

GAME**	SCORE**	NAME**	
BASEBALL	67	Danielle Anderson	
BOMB JACK	150,150	Meaghan Jenkinson	
BORDERLINE	429,610	Amanda Lawford	***
CHAMPION BOXING	K.O./Level 5	Jason Puschenjak	
CHAMPION GOLF	-11	M. Meehan	***
CHAMPION TENNIS	6-0	Jason Anderson	
CHOPLIFTER	180,500	Marcus Thompson	
CONGO BONGO	235,360	Sean Jenkinson	***
DEMON GOBBLER	53,550	Neuzat Ismaili	
DRAGON WANG	796,000	Stephen Emmett	***
EXERION	3,263,300	Malcom Chin	
GIRL'S GARDEN	51,130	Carly Pickett	
G.P. WORLD	1,008,560	Stephen Emmett	***
HANG-ON II	2,269,500	Stewart Crowther	
HUSTLE CHUMY	117,500	Jarron Hall	***
HYPER SPORTS	961,520	Jason Anderson	
LODE RUNNER	1,059,300	Richard Bolt	
MONACO G.P.	999,999	Adam Bolt	
NINJA PRINCESS	249,750	Craig Flowers	***
N-SUB	119,264	Dean Lyons	
ORGUSS	535,000	Michael Van Den Heuvel	
PACAR	201,342	Shane Gardner	***
PITFALL II	300,000	Stephen Emmett	***
		Sean Jenkinson	***
POP FLAMER	937,360	Malcolm Chin	
SAFARI HUNTING	69,230	Jason Anderson	
SAFARI RACE	74,720	Jane Anderson	
SEGA FLIPPER	1,450,500	Neuzat Ismaili	
SEGA GALAGA	172,390	Craig Flowers	***
SINBAD MYSTERY	269,350	Richard Bolt	
STAR JACKER	239,443	Brett McCallum	***
VERMIN INVADER	26,150	Danielle Anderson	
VORTEX BLASTER	395,290	Nelson Chin	
YAMATO	220,000	David Skene	***

< < < < < < < < < < < < < < < < < < < < < < < < < <

UTILITY:

....Spreading it around....

Snivel!! Snivel!! Cough!! Wheeze!! Oh, dear!! It seems that your favourite editorial team has come down with a case of that dreaded *Mutatis mutandis* we mentioned in *UTILITY* in the last issue. Being as we're such generous souls, we have, in our infinite wisdom, decided not to be selfish about it and will share some of it with the rest of our loyal followers. That is, all of those who do not own an assembler (we already know that those who do are suffering along with us ~ *time to get out your hankies...ED*). Enough of this (Aaaah...Chhoooo!!) nonsense. What we really mean is that we have been able to find the time to develop the necessary BASIC code required to allow those who utilise Disk BASIC or IIIB Cartridge BASIC to execute the SCREEN 2 SWAP Assembly Listing which appeared in the last issue!

The instructions as published for the use of that program also apply to users of IIIB BASIC, disregarding all references to SAVEM and LOADM, which cartridge BASIC does not support. Once this program has been **LOAD**ed into memory and **RUN**, the BASIC code can be **NEW**ed. After this has been accomplished, the program which is to provide the first of your screens can then be **LOAD**ed,**RUN** and then **NEW**ed (*why do I have this feeling of Deja Vu...ED?*). WITHOUT getting your grubby little mitts anywhere within the vicinity of the **[RESET]** key, the machine code program should now be **CALL**ed using ~ **CALL &HCFD4**. This will cause the screen that you now see to be transfered from VRAM into RAM. If you have accomplished this correctly, it is now a simple matter to repeat the proceedure with the program that you have to draw your second screen. Once this has been executed, it is simply a matter of **CALL**ing the imbedded machine code yet again (or as many times as you like) so as to vary your **SCREEN 2** display!!!

By the way (snivel...cough..splutter), there is no known MEDICAL cure for *Mutatis mutandis* (wheeze), we can only recommend the therapy obtained from spending many enjoyable hours *tied* to your SEGA computer (sniff...sniff)!!!

provide the first of your screens can then be **LOADed,RUN** and then **NEWed** (*why do I have this feeling of Deja Vu...ED?*). WITHOUT getting your grubby little mitts anywhere within the vicinity of the **[RESET]** key, the machine code program should now be **CALLed** using **~ CALL &HCFD4**. This will cause the screen that you now see to be transfered from VRAM into RAM. If you have accomplished this correctly, it is now a simple matter to repeat the proceedure with the program that you have to draw your second screen. Once this has been executed, it is simply a matter of **CALLing** the imbedded machine code yet again (or as many times as you like) so as to vary your **SCREEN 2** display!!!

By the way (snivel...cough..splutter), there is no known MEDICAL cure for *Mutatis mutandis* (wheeze), we can only recommend the therapy obtained from spending many enjoyable hours *tied* to your SEGA computer (sniff...sniff)!!!

```
10 REM **   Screen 2 Swap     **
20 REM **                     **
30 REM ** Disk basic ONLY     **
40 REM **                     **
50 LIMIT &HCFD4
60 C=0:A=&HCFD4
70 FOR B=1 TO 44:READ D$:D=VAL("&H"+D$
 ):C=C+D
80 POKE A,D:A=A+1:NEXT
90 IF C <> 5511 THEN BEEP 2:PRINT "BAD
 DATA!!!":STOP
100 END
110 REM ** Machine Code Data **
120 REM ------------------
130 DATA F3,21,00,00,11,00,D0,01
140 DATA 00,18,CB,AC,CD,F0,CF,CB
150 DATA EC,CD,F0,CF,23,0B,78,B1
160 DATA 20,F0,FB,C9,CD,28,65,DB
170 DATA BE,08,1A,CD,3A,65,D3,BE
180 DATA 08,12,13,C9
190 REM **                      **
200 REM ** CHECK ALL DATA VERY **
210 REM **       CAREFULLY      **
```

```
10 REM **   Screen 2 Swap   **
20 REM **                   **
30 REM **  IIIB basic ONLY  **
40 REM **                   **
50 POKE &H995C,&HD4:POKE &H995D,&HCF
60 C=0:A=&HCFD4
70 FOR B=1 TO 44:READ D$:D=VAL("&H"+D$
):C=C+D
80 POKE A,D:A=A+1:NEXT
90 IF C <> 5569 THEN BEEP 2:PRINT "BAD
 DATA!!!":STOP
100 END
110 REM ** Machine Code Data **
120 REM -------------------
130 DATA F3,21,00,00,11,00,D0,01
140 DATA 00,18,CB,AC,CD,F0,CF,CB
150 DATA EC,CD,F0,CF,23,0B,78,B1
160 DATA 20,F0,FB,C9,CD,CB,2B,DB
170 DATA BE,08,1A,CD,44,2C,D3,BE
180 DATA 08,12,13,C9
190 REM **                   **
200 REM ** CHECK ALL DATA VERY **
210 REM **      CAREFULLY      **
```

```
10 REM ** Screen 2 Swap Demo **
20 REM
30 SCREEN 2,2:CLS
40 FOR T=0 TO 20
50 CIRCLE (INT(RND(1)*255),INT(RND(1)*
191)),INT(RND(1)*100),INT(RND(1)*14)+1
60 NEXT :BLINE(58,80)-(198,112),1,BF
70 CURSOR 62,86:PRINTCHR$(17);"This is
 the"
80 CURSOR 68,98:PRINT"1st screen"
90 CALL &HCFD4
100 CLS:FOR T=0 TO 20
110 LINE (INT(RND(1)*255),INT(RND(1)*1
91))-(INT(RND(1)*255),INT(RND(1)*191))
,INT(RND(1)*14)+1,B
120 NEXT :BLINE(58,80)-(198,112),1,BF
130 CURSOR 62,86:PRINTCHR$(17);"This i
s the"
140 CURSOR 68,98:PRINT"2nd screen"
150 FOR D=0TO 1000:NEXT
160 CALL &HCFD4:BEEP:GOTO 150
```

KEYBOARD MAESTRO:

...CONTinuing the RUN...

This issue, we continue the series that has kept you all in suspense waiting for the next instalment, by providing you with the facility to start and stop a program or to re-start one at your merest whim, by means of a simple two-handed keypress (though for the more dexterous amongst you, one hand can suffice for at least one of these).

The subject of this issue's *KEYBOARD MAESTRO* are the alterations that are required to convert your standard (or otherwise modified) BASIC Disk to allow you to invoke an immediate **CONT**inue of the the current program flow or to produce an instantaneous re**RUN** of the resident listing, *ad libitum*. What we have acheived for our avid addicts is yet another time (read 'labour') saving alteration for your SEGA BASIC Disk!!

No doubt, many of you will realise that what we have to offer is already available within BASIC ver. 1.0p!! What we have that is different is that the first of the alterations can now be made available to the BASIC programmer as a simple **[CTRL] + [C]** keypress. This will save the programmer from the tedious task of having to enter **C - O - N - T**, at any time it is desired to **CONT**inue the program flow after the **[BREAK]** key has been activated. There are, no doubt, a few of our avid readers who have realised that **[CTRL] + [C]** is already installed within SEGA BASIC as the internal Control Code to invoke the equivalent of the **[BREAK]** keypress!! This facility (*our* **[CTRL] + [C]**) is, however, not normally available to the programmer as an executable Control Code within a program.

The other alteration that we have to offer is nothing more than the facility to **RUN** a program with a mere dual keypress ~ [CTRL] + [¥] ~ to invoke the execution of a program. "*But we already have this ability built-in!!*", you are heard to exclaim! So it is!!! But the difference between our modification and standard BASIC is that **[CTRL] + [¥]** will only execute a program that has already been installed into memory (it MUST be **LOADed** beforehand). Or, the equivalent **CHR$(25)** can be **PRINTed** within a program to cause a re-start of that program!

As is the case with all previous *KEYBOARD MAESTRO*'s, all due course must be taken with the entering of the following program, the DATA must be thoroughly checked (*the King's Gambit is best for this...ED*) before the program is executed. You should have your BASIC disk inserted in the Disk Drive (with the 'Write Protection Tab' enabled), and after 27,69 RUNning our program, you will find that these alterations will produce the effects, that we have found indispensable, will occur. Here is the program...

```
10 REM CTRL+(¥)=CHR$(25)=RUN
20 REM
30 REM CTRL+(C)=CHR$(1) =CONT
40 REM
50 REM   Disk BASIC ONLY!!!
60 REM
70 CLS:PRINT "Standby...":PRINT :C=0
80 READ D$:L=LEN(D$):IF D$="*" THEN 12
0
90 IF L=4 THEN AD=VAL("&H"+D$):C=C+AD:
GOTO 80
100 IF L=2 THEN DT=VAL("&H"+D$):POKE A
D,DT:AD=AD+1:C=C+DT:GOTO 80
110 PRINT "BAD DATA!!!!":BEEP 2:STOP
120 IF C<>87307 THEN 110
130 DSKI$ 0,1;A$,0,15
140 IF A$<>"SYS: disk BASIC" THEN PRIN
T "This is NOT a BASIC system disk!!":
BEEP 2:STOP
150 CALL &HF000
160 REM
170 DATA 5157,E0,05,5187,D3,05
180 DATA 5C85,01,5CAC,19,F000
190 DATA CD,19,37,11,00,51,01,06
200 DATA 02,CD,85,36,11,00,5C,01
210 DATA 06,0D,CD,85,36,CD,3E,38
220 DATA C9,*
230 REM
240 REM Please check all data
250 REM   very carefully!!!
```

```
10 REM ****************
20 REM * RUSSIAN  DANCE *
30 REM *       BY       *
40 REM * Gwen.Wood 1989 *
50 REM ****************
60 SCREEN 2,2:COLOR 1,15,,6:CLS
70 PRINT CHR$(17):CURSOR 55,50:PRINT "
RUSSIAN DANCE":PRINT CHR$(16):CURSOR 1
00,160:PRINT "By Gwen WooD.  1989"
80 FOR A=1 TO 600:NEXT A
90 CLS:SCREEN 1,2:COLOR 1,10:CLS:PRINT
 "TUNING UP"
100 SCREEN 2,1
110 V=INT(RND(1)*10)+1
120 C=523:D=587:E=659:F=698:A=440:P=50
:T=100:S=25
130 SOUND 1,C,V:FOR N=1 TO P:NEXT
140 SOUND 1,A,V:FOR N=1 TO P:NEXT
150 SOUND 1,D,V:FOR N=1 TO T:NEXT
160 SOUND 1,A,V:FOR N=1 TO T:NEXT
170 SOUND 0
180 COLOR ,15,(0,0)-(255,120),6
190 COLOR 14
200 FOR Y=0 TO 160 STEP 6
210 CURSOR 0,Y:PRINT "[][][]
                                [][][":REM 32 SP
ACES
220 NEXT Y
230 COLOR ,14,(0,121)-(255,165),6
240 COLOR ,14,(0,167)-(255,191),6
250 SOUND 1,C,V:FOR N=1 TO P:NEXT
260 SOUND 1,A,V:FOR N=1 TO P:NEXT
270 SOUND 1,D,V:FOR N=1 TO T:NEXT
280 SOUND 1,A,V:FOR N=1 TO T:NEXT
290 SOUND 0
300 COLOR 14
310 FOR Y=0 TO 115 STEP 6
320 CURSOR 35,Y:PRINT "{}{}{}{}{}{}{}{
}{}{}{}{}{}{}{}{"
330 NEXT Y
340 SOUND 1,C,V:FOR N=1 TO T:NEXT
350 SOUND 1,F,V:FOR N=1 TO T:NEXT
360 SOUND 1,E,V:FOR N=1 TO P:NEXT
```

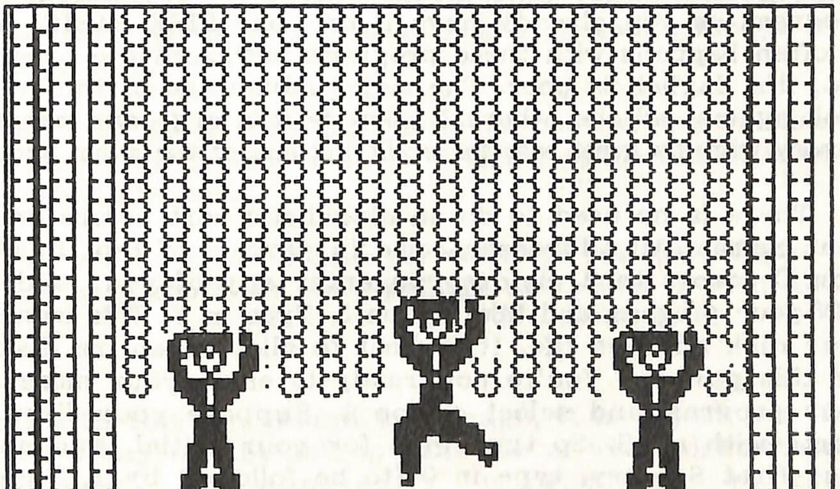
```
370 SOUND 1,F,V:FOR N=1 TO S:NEXT
380 SOUND 1,E,V:FOR N=1 TO S:NEXT
390 SOUND 1,D,V:FOR N=1 TO T:NEXT
400 SOUND 0
410 COLOR ,9,(0,0)-(30,155),6
420 COLOR ,9,(225,0)-(255,155),6
430 SOUND 1,C,V:FOR N=1 TO P:NEXT
440 SOUND 1,A,V:FOR N=1 TO P:NEXT
450 SOUND 1,D,V:FOR N=1 TO T:NEXT
460 SOUND 1,A,V:FOR N=1 TO T:NEXT
470 REM  ARMS UP LEFT
480 PATTERNS#0,"7FDFDDEAC8E5627F"
490 PATTERNS#1,"7D3D1D0F050F0F0F"
500 PATTERNS#2,"C0C0C080800000C0"
510 PATTERNS#3,"F0F8FCBC3C9E8683"
520 REM  LEG TO LEFT
530 PATTERNS#4,"0003071F3F3F3CB8"
540 PATTERNS#5,"B040000000000000"
550 PATTERNS#6,"7CFCFEFFFF9F0F0F"
560 PATTERNS#7,"0F0E0E0E04060000"
570 REM  LEG STRAIGHT
580 PATTERNS#8,"0F1F1D1D1D183D3D"
590 PATTERNS#9,"3818300000000000"
600 PATTERNS#10,"80C0C0C0C0C0E0E0"
610 PATTERNS#11,"E0C0600000000000"
620 REM  LEGS BENT
630 PATTERNS#12,"070F1F3F3E381C1E"
640 PATTERNS#13,"0E060C0000000000"
650 PATTERNS#14,"C0E0F0F8F83870F0"
660 PATTERNS#15,"E0C0600000000000"
670 REM  ARMS ON HIPS
680 PATTERNS#16,"1F1F1D0A0805021F"
690 PATTERNS#17,"3D6D6D7F350F0F0F"
700 PATTERNS#18,"C0C0C080800000C0"
710 PATTERNS#19,"E0E0B0F060808080"
720 REM  LEG TO RIGHT
730 PATTERNS#20,"3E3F7FFFFFF9F0F0"
740 PATTERNS#21,"7070702060000000"
750 PATTERNS#22,"00C0E0F8FCFC3F1D"
760 PATTERNS#23,"0702000000000000"
770 REM  ARMS UP
780 PATTERNS#24,"7FDFDDEAC8E5627F"
```

```
790 PATTERNS#25,"7D3D1D0F050F0F0F"
800 PATTERNS#26,"D8D8CC9C9C1C38F0"
810 PATTERNS#27,"F0E0C08000808080"
820 REM  LEGS APART IN AIR
830 PATTERNS#28,"07060E1E3E7CFCF8"
840 PATTERNS#29,"F86060C000000000"
850 PATTERNS#30,"C0C0E0F0F87C7E3E"
860 PATTERNS#31,"3E0C0C0600000000"
870 REM  ARM UP RIGHT
880 PATTERNS#32,"0303030101000001"
890 PATTERNS#33,"070F1F1D787161C1"
900 PATTERNS#34,"FEFBBB753A746FE"
910 PATTERNS#35,"BEBCB8F0A0F0F0F0"
920 MAG 3
930 SCREEN 2,2
940 G=196:A=220:B=247:C=262:D=294:E=33
0:F=370:G1=392:A1=440:B1=494:C1=523:C2
=554:D1=587:E1=659:F1=740:G2=784:A2=88
0:B2=988
950 S=S:N=N:P=8:H=24
960 FOR X=1 TO 3
970 S=P:N=G2:GOSUB 1750:GOSUB 1770
980 S=H:N=G2:GOSUB 1750:GOSUB 2090
990 S=H:N=F1:GOSUB 1750
1000 S=P:N=G2:GOSUB 1750:GOSUB 1770
1010 S=P:N=G2:GOSUB 1750:GOSUB 2010
1020 S=P:N=E1:GOSUB 1750:GOSUB 2010
1030 S=P:N=D1:GOSUB 1750:GOSUB 1850
1040 S=P:N=C1:GOSUB 1750:GOSUB 2010
1050 S=P:N=E1:GOSUB 1750:GOSUB 2010
1060 S=P:N=D1:GOSUB 1750:GOSUB 1930
1070 S=H:N=D1:GOSUB 1750:GOSUB 1770
1080 S=H:N=C2:GOSUB 1750
1090 S=P:N=D1:GOSUB 1750:GOSUB 2090
1100 S=P:N=D1:GOSUB 1750:GOSUB 1770
1110 S=P:N=B1:GOSUB 1750:GOSUB 2010
1120 S=P:N=A1:GOSUB 1750:GOSUB 1850
1130 S=P:N=G1:GOSUB 1750:GOSUB 2010
1140 S=H:N=B1:GOSUB 1750:GOSUB 1850
1150 S=H:N=D1:GOSUB 1750
1160 S=P:N=A1:GOSUB 1750:GOSUB 2010
1170 S=H:N=A1:GOSUB 1750:GOSUB 1850
```

```
1180 S=H:N=D1:GOSUB 1750
1190 S=P:N=G1:GOSUB 1750:GOSUB 2010
1200 S=H:N=G1:GOSUB 1750:GOSUB 1850
1210 S=H:N=D1:GOSUB 1750
1220 S=P:N=F :GOSUB 1750:GOSUB 2090
1230 S=H:N=F :GOSUB 1750:GOSUB 2090
1240 S=H:N=D1:GOSUB 1750
1250 S=P:N=A1:GOSUB 1750:GOSUB 2010
1260 S=H:N=A1:GOSUB 1750:GOSUB 2010
1270 S=H:N=D1:GOSUB 1750
1280 S=P:N=B1:GOSUB 1750:GOSUB 2010
1290 S=P:N=D1:GOSUB 1750:GOSUB 1770
1300 S=P:N=G1:GOSUB 1750:GOSUB 2090
1310 S=P:N=D1:GOSUB 1750:GOSUB 1770
1320 S=P:N=A1:GOSUB 1750:GOSUB 2010
1330 S=P:N=D1:GOSUB 1750:GOSUB 1930
1340 S=P:N=E1:GOSUB 1750:GOSUB 1930
1350 S=P:N=F1:GOSUB 1750:GOSUB 2010
1360 S=P:N=G2:GOSUB 1750:GOSUB 1850
1370 S=P:N=F1:GOSUB 1750:GOSUB 1850
1380 S=P:N=G2:GOSUB 1750:GOSUB 2010
1390 S=P:N=G2:GOSUB 1750:GOSUB 1770
1400 S=P:N=E1:GOSUB 1750:GOSUB 2090
1410 S=P:N=D1:GOSUB 1750:GOSUB 1770
1420 S=P:N=C1:GOSUB 1750:GOSUB 2090
1430 S=P:N=E1:GOSUB 1750:GOSUB 1930
1440 S=P:N=D1:GOSUB 1750:GOSUB 1770
1450 S=P:N=C2:GOSUB 1750:GOSUB 1930
1460 S=P:N=D1:GOSUB 1750:GOSUB 1770
1470 S=P:N=D1:GOSUB 1750:GOSUB 2010
1480 S=P:N=B1:GOSUB 1750:GOSUB 1850
1490 S=P:N=A1:GOSUB 1750:GOSUB 2010
1500 S=P:N=G1:GOSUB 1750:GOSUB 1850
1510 S=P:N=D1:GOSUB 1750:GOSUB 2010
1520 S=P:N=A1:GOSUB 1750:GOSUB 1850
1530 S=P:N=D1:GOSUB 1750:GOSUB 2010
1540 S=P:N=G1:GOSUB 1750:GOSUB 1930
1550 S=P:N=D1:GOSUB 1750:GOSUB 2010
1560 S=P:N=F :GOSUB 1750:GOSUB 1930
1570 S=P:N=D1:GOSUB 1750:GOSUB 2010
1580 S=P:N=A1:GOSUB 1750:GOSUB 1930
1590 S=P:N=D1:GOSUB 1750:GOSUB 1770
```

```
1600 S=P:N=B1:GOSUB 1750:GOSUB 2010
1610 S=P:N=D1:GOSUB 1750:GOSUB 1770
1620 S=P:N=G1:GOSUB 1750:GOSUB 2010
1630 S=P:N=D1:GOSUB 1750:GOSUB 1770
1640 S=P:N=A1:GOSUB 1750:GOSUB 2010
1650 S=P:N=D1:GOSUB 1750:GOSUB 1770
1660 S=P:N=E1:GOSUB 1750:GOSUB 2010
1670 S=P:N=F1:GOSUB 1750:GOSUB 1770
1680 NEXT X:FOR Z=1 TO 500:NEXT Z
1690 LINE (0,0)-(255,120),6,BF
1700 SPRITE 2,(10,120),24,6:SPRITE 4,(
230,120),24,0:SPRITE 6,(230,120),24,0:
SPRITE 3,(230,120),24,0
1710 LINE (0,121)-(255,155),6,BF
1720 BLINE (100,100)-(150,120),0,BF:CO
LOR 1
1730 CURSOR 105,106:PRINT "THE END"
1740 FOR Z=1 TO 1000:NEXT Z:GOTO 10
1750 FOR T=10 TO 2 STEP -S:SOUND 1,N,T
:NEXT :RETURN
1760 REM  STRAIGHT LEGS
1770 SPRITE 1,(50,100),24,6
1780 SPRITE 2,(50,132),8,1
1790 SPRITE 5,(188,100),24,6
1800 SPRITE 6,(188,132),8,1
1810 SPRITE 3,(120,90),24,6
1820 SPRITE 4,(120,122),8,1:SOUND 1,26
2,10:SOUND 0
1830 RETURN
1840 REM  LEG TO RIGHT
1850 SPRITE 1,(54,100),32,6
1860 SPRITE 2,(50,132),4,1
1870 SPRITE 5,(192,100),32,6
1880 SPRITE 6,(188,132),4,1
1890 SPRITE 3,(124,90),32,6
1900 SPRITE 4,(120,122),4,1:SOUND 1,44
0,10:SOUND 0
1910 RETURN
1920 REM  LEG TO LEFT
1930 SPRITE 1,(46,100),0,6
1940 SPRITE 2,(50,132),20,1
1950 SPRITE 5,(184,100),0,6
```

```
1960 SPRITE 6,(188,132),20,1
1970 SPRITE 3,(116,90),0,6
1980 SPRITE 4,(120,122),20,1:SOUND 1,1
96,10:SOUND 0
1990 RETURN
2000 REM  BENT KNEES
2010 SPRITE 1,(50,105),16,6
2020 SPRITE 2,(50,132),12,1
2030 SPRITE 5,(188,105),16,6
2040 SPRITE 6,(188,132),12,1
2050 SPRITE 3,(120,95),16,6
2060 SPRITE 4,(120,122),12,1:BEEP 1:SO
UND 0
2070 RETURN
2080 REM  LEGS APART IN AIR
2090 SPRITE 1,(50,75),24,6
2100 SPRITE 2,(50,107),28,1
2110 SPRITE 5,(188,75),24,6
2120 SPRITE 6,(188,107),28,1
2130 SPRITE 3,(120,65),24,6
2140 SPRITE 4,(120,97),28,1
2150 RETURN
```



ADDRESS BOOK:

....Notes on the Address Book Program....

by

....Julius Guest....

This month's contribution from our irrepressible subscriber is a set of instructions that are intended to accompany the program..

"ADDRESS .BAS"

..which Disk drive users will find was provided FREE with the purchase of your original Disk BASIC disk. As Julius explains....

~~~~~

For once there is no need to copy a very useful utility program, as you already have it on your DISK BASIC! but as so often happens with these programs, although the program is o.k., it's English is poor. The major advantage of this address book against all the others I know is it's large and expansible memory with it's great ease for modifying any of the given addresses.

There is no need to do an alphabetic sort before entering your names. All addresses come in parcels of ten, numbered from 0 across to 9. Before you enter any of your addresses RUN your program and take option 3. Type in Y. This establishes your disk address file. It is best to allot an entire disk side for this purpose. You're now ready to enter your names. RUN your program and select option 1. Suppose your first name starts with an S. So type in S for your initial. And as it is your first S entry, type in 0 (to be followed by 1, 2 etc. for later names which start with S). Of course, you might easily have
~~~~~

more than 10 names starting with an S. If so, create a new file and call it S1. This makes room for ten more S names. Now, after you have supplied all your 4 bits of info. for each name, you have three available options.

1. If you wish to enter another name type in N with a <CR>.

2. If you wish to modify the above info. type in B with a <CR>.

3. If you wish to enter this name in your disk file type in M with <CR> but make quite sure your correct disk file is in the disk slot!

Finally, when searching for a name which is already on your disk file you proceed as follows:

1. Select option 2 on your menu.

2. Type in the initial letter of your name with a <CR>.

3. Now, examine each name within your parcel of ten names. If this happens to be the name you seek that's lucky. If not see next step.

4. You can move from one name to another one within the same parcel by typing in turn: N,<CR>,N,<CR> and type in the next block number.

5. Once you have found your correct name you may, if you wish, also get a printout on your printer on typing Y with a <CR>.

~~~~~

....and with this new-found knowledge, it is hoped that those concerned may now be able to make use of what might otherwise have been an apparently unpromising program!!!
~~~~~

```
10 REM"EASWIN.BAS"
20 REM
30 REM *** A blue brooch ***
40 REM
50 REM         J.Guest.
60 REM
70 REM This design takes about two
80 REM hours and 23 minutes to run,
90 REM but it is worth waiting for.
100 SCREEN 2,2:COLOR 1,15,,15:CLS
110 POSITION(128,95),0,1
120 CIRCLE(0,0),95,10,1
130 CIRCLE(0,0),91,10,1
140 PAINT(93,0),10
150 FOR T=0 TO 6.28 STEP 0.01
160 X=69.640*COS(T)+4.353*COS(16*T)
170 Y=69.640*SIN(T)-4.353*SIN(16*T)
180 PSET(X,Y),4
190 NEXT T
200 CIRCLE(0,0),15,4,1
210 FOR T=0 TO 5.92 STEP 0.3696
220 FOR K=1 TO -1 STEP -2
230 FOR S=0.8 TO 6.28 STEP 0.02
240 X=22.2*(2*COS(S)+COS(2*S))
250 Y=22.2*K*(2*SIN(S)-SIN(2*S))
260 X=X*SIN(S)*SIN(5.5*S)
270 A=X*COS(T)-Y*SIN(T)
280 B=X*SIN(T)+Y*COS(T)
290 IF A*A+B*B<225 THEN 310
300 PSET(A,B),4
310 NEXT S:NEXT K:NEXT T
320 BEEP2:BEEP2:REM That's it!
330 GOTO 330
```

```
10 REM TONE GUESS
20 REM ADAPTED & ENHANCED FROM A TI 99
/4A PROGRAM BY R.BRIFFA & N.JUHASZ
30 SCREEN1,1:CLS
40 A =INT(RND(1)*2882)+117
50 HG=3000:LG=117
60 CURSOR2,19:PRINT"HIGH":CURSOR28,19:
PRINT"LOW"
70 CURSOR2,1:PRINT"HERE'S THE TONE!"
80 SOUND1,  A,15
90 FOR S=1 TO 100
100 NEXT S
110 SOUND 0
120 CURSOR2,4:PRINT "GUESS PLEASE
                                    ":CU
RSOR 15,4:INPUT GUESS
130 IF GUESS<117ORGUESS>3000THEN120
140 IF GUESS =A  THEN 310
150 IF GUESS >A THEN 210
160 CURSOR2,8:PRINT "TOO LOW! "
170 IF GUESS > LG THEN 190
180 GOTO 250
190 CURSOR 27,21:PRINTGUESS:LG=GUESS
200 GOTO 250
210 CURSOR2,8:PRINT"TOO HIGH!"
220 IF GUESS < HG THEN 240
230 GOTO 250
240 CURSOR 2,21:PRINTGUESS:HG=GUESS
250 SOUND 1,GUESS,15
260 PRINT
270 FOR B =1 TO 50
280 NEXT B
290 SOUND 0
300 GOTO 70
310 SCREEN 2,2:CLS:PRINTCHR$(17):CURSO
R 40,60:PRINT"YOU GUESSED IT!!"
320 FOR PLAY =1 TO 100
330 SOUND 1,A,15
340 NEXT PLAY
350 SOUND 0
360 CLS:GOTO 30
```

```
10 REM"CDOGN.BAS"
20 REM    Decomposition of a given
30 REM integer into its prime factors
40 REM
50 REM          J.Guest
60 REM
70 CLS:DIM B(500)
80 CURSOR 9,10:INPUT"Enter given numbe
r: ";N
90 CLS
100 K=1:F=1
110 K=K+1:IF K>1+INT(N/2) THEN 150
120 A=N-K*INT(N/K)
130 IF A=0 THEN B(F)=K:F=F+1
140 GOTO 110
150 IF F=1 THEN CURSOR 4,13:PRINT"Give
n number is a prime!":CURSOR 0,22:END
160 FOR Z=1 TO F-1
170 FOR S=Z+1 TO F-1
180 IF B(Z)=1 THEN 230
190 P=B(Z):Q=B(S)
200 GOSUB 370
210 IF GCD<>1 THEN B(S)=1
220 NEXT S
230 NEXT Z
240 C=0:K=0
250 C=C+1:IF C>F-1 THEN 280
260 IF B(C)=1 THEN 250
270 K=K+1:B(K)=B(C):GOTO 250
280 CURSOR 7,0:PRINT"Decomposition of
";N
290 FOR I=1 TO K
300 P=0
310 S=B(I):M=B(I)
320 P=P+1
330 IF N-B(I)*INT(N/B(I))=0 THEN B(I)=
S*B(I):GOTO 320
340 CURSOR 6,2*I+2: PRINT M;" to the p
ower of";P-1
350 NEXT I
360 CURSOR 0,22:END
370 IF Q=0 THEN GCD=P:GOTO 410
380 T=ABS(P-Q)
390 P=Q:Q=T
400 GOTO 370
410 RETURN
```

ERRATUM:

....It doesn't happen very often....

or

...Who invented printers anyway??...

It truly is a rare occurance (*you're missing out too, eh??...ED*)!! In fact, I can't think of any other occasion since the current editorial committee (Mike and I) has been behind the reins of your favourite mag., where we decided to deliberately throw a 'curved ball' in your direction, just to see how many of you are as observant as you tell us you are. If you failed to find the inaccuracy that is of concern at this time, even though you may have entered and **RUN** this program then you should be ashamed (what *did* you use instead of the proper characters?)!!

It is with much pleasure that we commend the Sachs 'family' for being the first to brave our stature (**your** pedestal, NOT ours) and actually tell us of their discovery (*there you are Hannah and Arnold ~ we do keep our promises...ED*). It seems that the author of the program **'GOLD HUNT'**, which appeared in Vol. 3, No. 5, made use of some of the SEGA's internal GRAPHIC's characters as part of the screen display for the program. Where the problem arises is in the way that the printer treated these characters where they appeared within the SEGA's**PRINT**statements. Because the printer's character set did not know what the SEGA's codes were, it went ahead and provided it's own 'squiggles' in lieu thereof. This caused a major catastrophe in line 350 of the aforementioned program!! That line is reproduced below complete with the correct character code number values for the SEGA's BASIC interpreter!! In order to correct the problem, simply **LOAD** the program, re-enter line 350 and then **SAVE** the program anew and it should **RUN** as it was intended. Oh, by the way, this may not be the last time we bowl a 'googly' at you, so keep your wits about you!! Who wrote this program anyway??? (*I did!! What of it???...ED*)

```
350 SCREEN 2,2:FOR JJ=1 TO 7:CURSOR 44
+(E*6),168-(D*8):PRINT CHR$(253):IF M<
1 THEN CURSOR 44+(Z*6),168-(Y*8):PRINT
 CHR$(247)
```

WORKSHOP :

....The flavour really hits you....

or

....ask any gobbledock....

Up to this point in this series of articles, ardent followers should have a solid base of knowledge about what can and can't be done with your SEGA. This issue is going to 'lift the lid' of your computer and provide you all with a general overview of the internal arrangement of all of those multi-legged beasties and other bits (and bytes) to be found lurking therein.

We will be discussing the various chips (programmers slang for 'Silicon Chip'), which is itself a graphic description of what is technically an **IC** or Integrated Circuit. The Integrated Circuit is an extremely complex electrical circuit that has been miniaturised so as to to fit onto a small sliver of silicon, hence that description. So intricate is the circuitry inside such a device, that were the equivalent circuit constructed using standard individual electronic components, your SEGA would be larger than your house!!!! Fortunately, an intricate knowledge of such devices is not essential to learn how to program your SEGA, but knowing how they are interconnected and how they operate together (or otherwise) will make such an informed programmer more capable than an otherwise uninformed one (*and we all want to be the best, don't we??...ED*). All of this acumen can be imparted without resorting to excessive technical jargon and without YOU having to go anywhere near a Soldering Iron (*thank heaven for that...ED*)!!!

Although there are a variety of ICs within the confines of your SEGA's plastic case, many of these are provided to perform specific functions and are thus not accessible to a programmer. These chips are considered '*invisible*' (for programming purposes), thus, a description of their operation will only confuse the issue, therefore most of these have been omitted from this discourse (this series is intended to encourage you to be more proficient at your programming, not to turn you all into electronics wizards!!).

....Uggh, my brain hurts!!!....

The most important chip for the programmer in any computer system is the **C.P.U.** (the **Central** Processing Unit). In the case of the SEGA, this is the **Z80A** (pronounced ~ 'zed eighty ay') or IC D780C-1 (should you ever need to purchase one) which is an 8 bit microprocessor functionally identical to the earlier Z80, the only difference being the 'CLOCK SPEED'. Where the Z80 was designed to operate at 2 MHz (Megahertz), the later model Z80A, processes at the rate of 3.58 MHz. and will thus always win any race between these two (and many other computers to boot)! The CLOCK signal supplied to the Z80A is derived from a CERAMIC OSCILLATOR and is extremely accurate. Each CLOCK CYCLE lasts for 279.329 Nanoseconds (one Nanosecond is 10^-6 seconds or 1/100000th of a second). This signal controls the processing rate of the C.P.U., essentially determining the actual speed at which instructions are executed. The Z80A has been designed to execute over 700 different instruction codes, any one of which can be up to five bytes long. Naturally, a single byte instruction will execute many times faster than a five byte one!! Most books and manuals covering this IC give clock times for each instruction which apply to the Z80 running at 2 MHz. These times must be divided by 1.79 to suit the faster Z80A.

As far as chips go, the Z80A is a relatively large device having 40 pins (the little metal legs that sprout from it's sides) with which it is attached to the outside world. The number of pins on an integrated circuit is normally indicative of the complexity of the IC and the Z80A is no exception!! Contained within the confines of the Z80A's black exterior are various registers and busses (*can I get one to take me home...ED?*) which we intend to describe more fully in our next article. Within the electronics that comprise the SEGA computing system, the Z80A is known as **IC1** It is the brains of the microcomputer and is responsible for controlling all of the other devices within the system, according to the instructions contained within any program supplied to it. It is THE BOSS!!! It performs this feat by using **IC2** (the 'GATE ARRAY') which interfaces the Z80A with the other peripheral chips in the system. This chip has 28 pins and is essentially 'invisible' to the programmer.

....RAMming it home....

Contrary to what many of you may think, your SEGA computer has virtually NO memory in it, even though the salesman said that it had 48 K of memory when you bought it!! It is **IC3**, a 24 pin, 2KBytes RAM chip which is the sole onboard memory within the SC-3000. This 2K is only selected and used by Game Cartridges and My-cards to provide an area for the game's System RAM. It is memory addressed at &HC000~&HC7FF and is inaccessable to the programmer and thus another 'invisible' chip.

....the center of synthesis....

If we had taken the lid off our SEGA keyboard, we would find plonked dead center on the main circuit board, the rowdy member of the Z80A's encapsulated companions. This is **IC4**, a 16 legged DIL (*that's where that missing pickle from my lunch went...ED*!!). **DIL** is the abbreviation of the term Dual-In-Line package which describes a chip which has two rows of pins, one along each side. It is none other than the **SGC**, SN76489AN Sound Generation Controller, that we analysed in the last issue of this series. This chip is connected to the Z80A through that chip's port &H7F.

....PPI(e) and chips???....

Next on the menu is, of course, **IC5** which is the 8255A **PPI** (Programmable Peripheral Interface). This is another of the larger variety of chips (*extricated from a bigger spud, no doubt...ED*) being a 40 pin DIL the same as the Z80A. This IC is more often referred to as the I/O (Input/Output) interface. If the Z80A is the brains of the mob, then the 8255A must be considered the sensory receptor of your SEGA computer (it *feels* through the keyboard, it *hears* the cassette player and it *speaks* to the printer). This device is connected to the Z80A via four separate ports &HDC (input), &HDD (input), &HDE (output) and &HDF (control) and is responsible for interfacing the keyboard, joysticks, printers and cassette drive with the microprocessor. Port A (&HDC) is used for the keyboard input. Port B (&HDD) is programmed to accept further keyboard input into it's Least Significant nibble and the Most Significant nibble is used for the Cassette input and SP-400 printer/plotter status checks. The three least significant bits in Port C (&HDE) are connected to pins 13, 14 and 15 of **IC6** and are used to address the keyboard matrix and the most significant nibble is utilised by the Cassette output and the outputs for the SP-400.

The 8255A is an extremely flexible device. It can be programmed (by means of the control register) to operate in various modes and configurations. This control register is very complex and requires many pages to cover it's full capabilities, thus a whole article will be devoted to this chip in a future issue of this tome (*allwais waitink...waitink...waitink...ED*).

....making sense from QWERTY....

IC6 is the 74LS145 Keyboard Matrix Decoder. It is a 16 pin DIL and is physically connected between the 8255A and the keyboard. It decodes any key presses and presents this decoded data to the 8255A, when that chip requests this information. In addition to this, this chip is assigned the task of obtaining and decoding the joystick input.

....drilling from the Seargent Major....

This IC (being IC7) is perhaps the MOST 'invisible' of all of the chips on the SEGA's circuit board. For openers, it's on the underside of the circuit board (which makes it harder to see), it is not a DIL (*Ah, ha, a clever IC at last...ED*) and is provided to regulate the power supply to the circuit board, thus ensuring that all the other components are presented with a regulated current. It is a PC7805, three pin regulator.

....The best way to fry your chips....

This next Integrated Circuit, **IC8**, is a hybrid affair created for the purpose of interfacing the cassette drive with the 8255A. This chip converts the input signals from the cassette into square waves (more suitable for processing) and is provided with a 20dB attenuator for producing output signals to the cassette drive (*It is worth noting that this chip is not designed to accomodate a power input of 240V, a fact I discovered when the afformentioned voltage was accidently connected directly to this chip...it fried...ED*!!!). This chip is located immediately behind the cassette input/output plugs located at the rear of the SEGA's keyboard. It is an 11 pin In-Line IC.

....Dropping in on an old friend....

Continuing in numerical order, we now find ourselves confronted by the enormity of the SEGA's VDP. **IC9**, the TMS9929A, is perhaps the most visible of all of the ICs within the SEGA, since it's output is responsible for everything that is seen upon your television (or monitor) screen. It is this output which provides the interaction between the human being behind the keyboard and the electronics beneath it. Like the Z80A and the 8255A before it, this chip is the third (and the last) of the 40 pin DIL chips that can found on the circuit board within the SEGA's console. Like the Z80A, it too is controlled by a clock (a quartz crystal oscillator). This clock however, ticks over at the rate of 10.74 MHz!!! This incredible rate is necessary in order to keep up with the speed of the 'gun' inside the television tube which 'draws' what you see upon the screen. Although we have already discussed at length the ins and outs of the RAM associated with this beast, a more detailed expose of this chip's abilities will have to wait for another chapter of our tale.

....More is better....

Naturally, you would expect to find the aforementioned VRAM following (and connected to) to SEGA's Video Display Processor. What you might not expect is that this VRAM resides in eight individual chips, **IC10** to **IC17.** Each of these DILs contains a whole 2KBytes of the VRAM and are connected to the VDP via **IC18.** This is the 74HC04 14 pin DIL which is used to interface the 8 VRAM chips with the VDP and is as 'invisible' as a chip can get!!!

We have now covered all of the Integrated Circuits which appear upon the main circuit board beneath the SEGA's keyboard. There is however a small circuit board located above the main board which is associated with the signal processing for the VDP. There are three more additional chips located upon this board all of which are completely 'invisible' to the programmer and thus any explanation of their operation would be of no real value.

All of these previously mentioned devices are by no means the complete array of chips at the disposal of the operator of a SEGA computer, there are many more contained within the confines of the Super Control Station SF-7000. Since this most desirable piece of hardware is not universally available (and most of the integrated circuits therein require an involved knowledge of Machine Code), we feel that a forage through their intricate details can be left for another episode (*that future episode of WORKSHOP is going to be a real humdinger...ED!!*).

....In one 'ere....

Many of you may have pondered upon the the earlier coverage of the MB8128 (IC3 ~ the 2K on-board RAM chip) and wondered where all of that 48 Kbytes of RAM that the salesman said you had available to you!! For those of you who may not know, **RAM** is the acronym for the term Random Access Memory (*you mean it has nothing to do with sheep...ED???*). This form of memory is designated so because of the nature of it's accessibility. That is, it can be accessed completely at random. Each of the memory cells which comprise the total can be addressed individually for the purpose of obtaining the contents there-in ~ **READ**ing ~ or to alter the contents at that location ~ **WRIT**ing. The alternative form in which memory can be found is known as **ROM.** This stands for Read Only Memory. This form is as it's name implies. It is stored memory that cannot be altered by an attempt to WRITE to it (*I had always thought computers were illiterate...ED*).

It is precisely a number of these types of Intergrated Circuits which make up the numbers to provide the aforementioned apparent memory loss. They can be found lurking within the confines of the Disk Drive I/O cassette, the IIIA BASIC Cartridge or the IIIB BASIC Cartridge and any of the GAMES Cartridges or My Cards. Thus, the SEGA's memory capability is fully user selectable (yet another feature of our marvellous machine)!!

There is however, one slight disadvantage with this arrangement. The program, which are the instructions that execute SEGA's BASIC, is contained within ROM for all of these possibilities except Disk BASIC!!

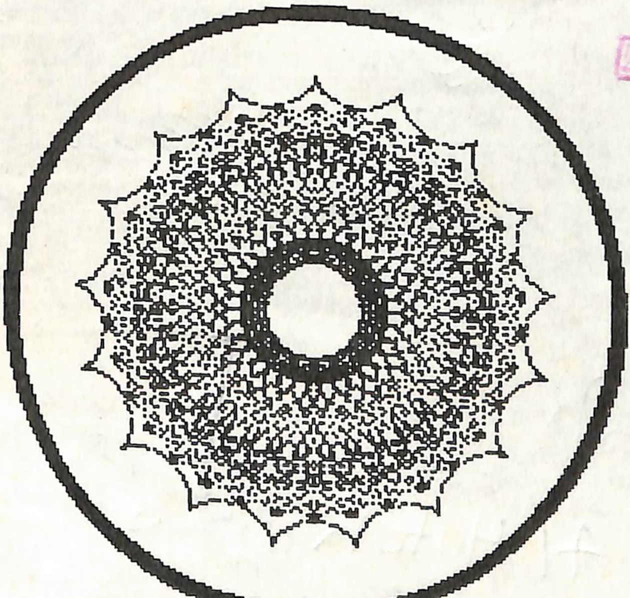
....Memories are made of this....

Both of the BASIC cartridges contain within their meagre dimensions a total of 32 Kbytes of ROM (though the two BASICs are slightly different) and either 16 Kbytes of RAM (IIIA) or 32 Kbytes of RAM (IIIB). Observant readers will note that these values are greater than the **FRE**e memory which is available according to the BASIC interpreter. This discrepancy is caused by the necessity to allocate a certain amount of this RAM for the use of the interpreter's SYSTEM RAM (which has been previously discussed in these pages).

Disk BASIC on the other hand comprises a total of 64 Kbytes of RAM. What is commonly called Disk BASIC ROM is, in fact, only a program that has been loaded from a Disk and stored within that 64 Kbytes of RAM. The only true ROM to be found within the SF-7000 is the ROM chip which provides the IPL or Initial Program Loader. This area of memory occupied by the Disk BASIC interpretor is called Disk BASIC ROM for no other reason than convention. It is this difference between the systems that enables a programmer to provide utilities (for example, those which have appeared within the pages of this magazine) that can permanently modify the operation of Disk BASIC. If Disk BASIC were contained within a ROM chip (as is Cartridge BASIC), none of these programs could exist. Conversely, because the Cartridge BASIC interpretor is contained within ROM, such modifying programs for cartridge BASIC cannot exist!

This ability to load the BASIC system program from the stored contents of a compact floppy disk makes for what is perhaps the second big (*Bigger???...ED*) advantage of the SF-7000, besides the sheer speed of program loading capacity. The presence of the IPL allows any number of alternate SYSTEM programs to be inserted into the SF-7000's RAM completely at random, sometimes two (or more) at a time!!!

This is the end of this issue (*Ohhh!! Just when I was getting involved...ED*)! We shall leave you to devour this general overview of the SEGA's hardware system and to wait with bated breath for the next issue wherein you WILL join us for an in depth promulgation of the ZILOG Z80A microprocessor's machinations. We'll 'pick your SEGA's brains'. Ta, ta!!



29 Mort Edge 0111

FILES:

40 Per Pound

oc

EINA 38

****

Sega Help Line:

If you have any urgent problems with your computer, or wish to find other Sega users in your area then call one of these members :

Garry	309-7130
Stewart	783-7328

They will be happy to help you.

Marketing Co-ordinator
P.O. Box 589
Mordialloc Vic. 3195

****

41 Hutchinson Edges

From M! Marlow & Webb