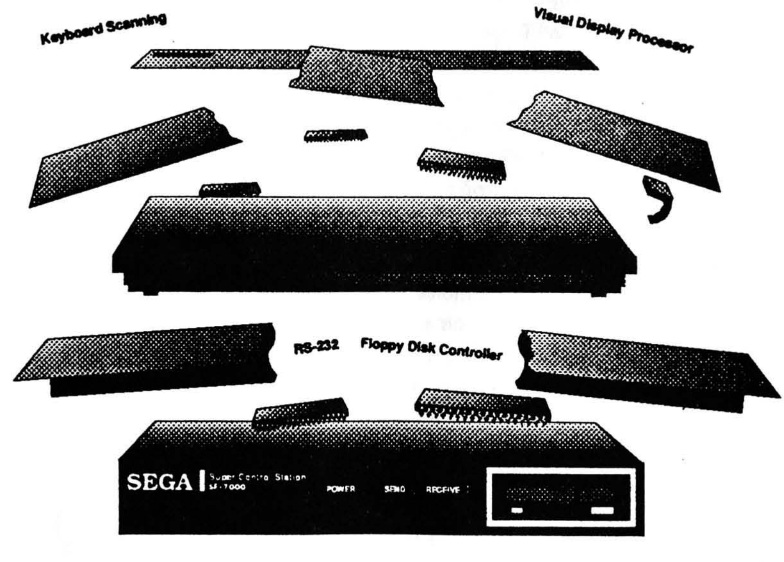
SEGA®

Computer

July / August Issue 1988

The official magazine of the SEGA User Club of New Zealand

Inside the SEGA



Published by MJH Software in association with Poseidon Software

SEGA MAGAZINE

SUBSCRIPTION YEAR

Oct 1987 - Nov 1988

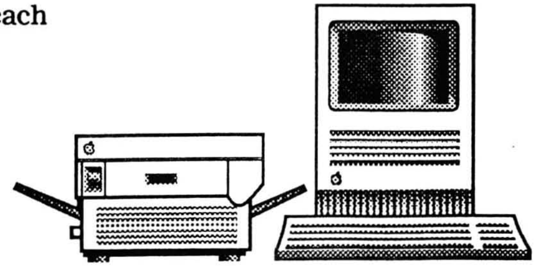
New Zealand Subscription: NZ$25 incl GST
Australia Subscription: A$26 Airmail

- All contributions are welcome, but please include your name, address and telephone number.
- A question and answer page in the form of Letters To The Editor is provided and we will do our best to answer any questions about software or programming.
- It is preferable that programs be submitted on tape or disk in a listable form. (No copyright protection please). A listing is useful but don't worry if you aren't lucky enough to own a printer. Where required please include instructions on how to type in the program.
- Please check your programs thoroughly for errors and spelling mistakes before sending it to us. Please send updates if any errors are discovered, so we can publish corrections.
- All software programs received by the magazine becomes the property of MJH Software unless by prior arrangement. They are accepted on the basis that they are the original work of the author or required modification to run on a SEGA.
- All contributions are subject to approval by the editor and may be edited to suit the magazine style. Submitted programs will be returned on request
- Each issue two prizes of NZ$40 and NZ$20 may be awarded to the program of the month at the descretion of the Editor.
- The pages in this magazine are created using only personal computers and DeskTop Publishing technology. All text is entered and edited on an Apple Macintosh using Microsoft Word and then assembled using Aldus PageMaker 3.0 page layout software. The finished pages are printed on an Apple LaserWriter Plus and then sent on to offset printers.
- Offset printed by Garden Graphics Printing, Tokoroa.

Published bi-monthly by **MJH Software**
36 Colum Place Bucklands Beach
Auckland New Zealand
Telephone (09) 534-3379

in association with

Poseidon Software
FREEPOST 243 P.O. Box 277
Tokoroa New Zealand
Telephone (0814) 67-105



Welcome to the new look

SEGA Computer

The official magazine of the SEGA User Club of New Zealand

Issue 5 Contents

MJH Software

Letters to the Editor

- **This question and answer page is provided to help you. So send me some questions.**

- The last letter from the last issue was in fact from Geoff McM of Hamilton, not Geoff from Tokoroa as stated. This will put many peoples minds at rest!

Dear Editor,

❶ What is the address for BOOT in Basic Version 1.1?

❷ Is there a quicker version of basic around?

David Martin, South Australia

Editors reply

❶ I've only got Disk Basic version 1.0p and unitl recently I thought this was the only version around. You could try #051B which is the BOOT routine in version 1.0p (the only version which I have documented).

It is possible that version 1.1 is quite different from version 1.0p and this brings up another problem - the majority of my machine code programs may not work with your version. Anyway I will try and find a copy of version 1.1 and check out whether it is different - hopefully Japan have just corrected a few bugs, and in 1.1 there will be no drastic changes.

❷ Not that I know of - maybe 1.1 is faster than 1.0p. I wouldn't worry about it, as Basic is too slow in many cases for it to be useful - it all depends on the application. Often you are better off using machine code and if you need Basic you can always make CALL's to the necessary routines. The next Machine Code Programming Course starts to detail how to call certain routines in the Basic interpreter.

Dear Editor,

❶ How do you detect sprite collisions in machine code? In Basic you could use
IF X+16>Y AND X<Y+16 THEN PRINT "YOU CRASHED INTO THE SPACESHIP"

❷ How do you get one sprite (eg a man) and make him walk and how do you flip sprites (make a car face to the right then face left when you turn left)?

❸ I noticed that you can increase or decrease any variable (by one), but can you add and subtract to A and HL? How can you print the result?

❹ How can you draw really fast graphics in machine code? I know it can be done because the maps are drawn so quickly in geography quiz. What would be the machine code equivalents to line, circle, paint etc?

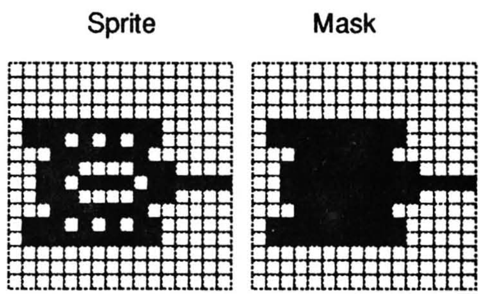
Steven Boland, Auckland

Editors reply

❶ Basically you have to use the same method as in Basic. There are more complicated methods such as those used in "Tank Battle", which makes use of the SEGA's built-in sprites (called "Hardware" sprites) to display the tanks, but collision dectection is done at a software level by actually checking the screen bitmap.

The way it works is to have two copies of the sprites. One is used by the SEGA hardware and is stored in VRAM. The other is a "shadow" or mask of the sprite and is stored in RAM for use by the software.

To check for a collision, the area beneath the mask in place of the sprite on the screen is checked for 1's (ie bits set). If any are found then a collision has occured.



This involves quite a lot of work in machine code, but the result is perfect collision detection. If you use the method of checking as in Basic then a bullet which moves past a tank (but not actually hitting it) would cause a collision.

❷ You don't really flip the sprite, but you define another spirte pattern for the car facing right. Tank Battle has patterns for 12 directions. (0,30,60,90 degrees etc). To make the tank turn you simply change the pattern which is being displayed. In Basic using MAG 2 sprites where sprite pattern 0 is left and pattern 4 is right then

SPRITE 0,(128,95),0,1 Sprite facing left
SPRITE 0,(128,95),4,1 Sprite facing right

❸ See "Simple Arithmetic" in the Machine Code Programming Course from Issue 2 (page 11) as this explains how to add and subtract from A and HL. The example program also shows you how to print the result using a routine at #7B9E (#2B3A for Cartridge Basic). This routine is equivalent to PRINT HL in Basic.

❹ A bit much to explain right now - see future Machine Code Programming Courses.

Sorry, couldn't fit in all the letters that arrived.

EDITORIAL

Sorry for this issue being so late, but this has been the hardest issue to complete so far. It took a long time to write the article on SEGA hardware and I found it very hard to compress all my information in to so few pages.

As there are only two programs in this issue and I am running out of money, no magazine tape will be included with this issue. The two programs will be added to the next issue's tape. Sorry, but this will mean a slight delay for some of you before you will be able to use these programs. - they are actually pretty short, so you can probably type them in anyway.

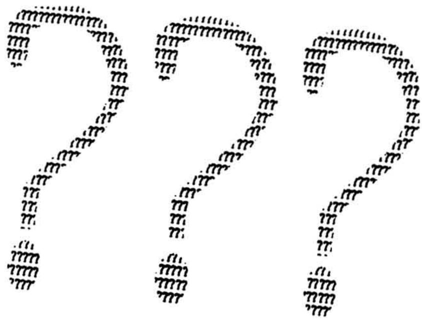
The next issue of the magazine will most likely be the last. The magazine has become too time consuming for me to continue again next year. The technology which makes this magazine possible, is also very expensive and so far this year the cost to me personally has been over $1,500.

EDITOR: Michael Hadrup

MAN LOGIC

Original Idea by Neil Bradley

Mystery Program



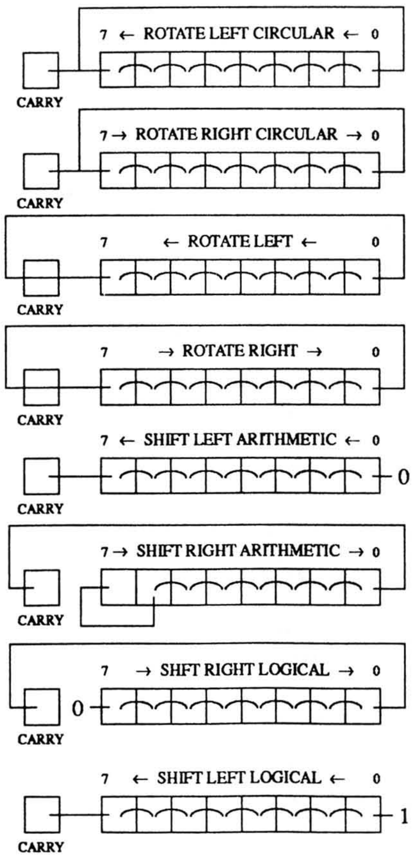
```
1 REM Picture expander
2 REM
3 REM By Michael Hadrup
4 REM
5 REM Original size    = 6144
6 REM Compressed size = 1067
7 REM
8 REM Saving = 83%
9 REM
10 X=72:X1=184:SY=1
20 WIDTH=X1-X
30 Y=X+SY*256
40 SCREEN2,2:COLOR1,11,,11:CLS
50 RESTORE400
60 FORN=0TO3
70 READB(N)
80 NEXT
90 RESTORE410
100 FORN=0TO5
110 READC(N)
120 NEXT
130 RESTORE1000:M=0
140 GOSUB270:IFB<>0THEN150P
145 GOTO145
150 IFB>128THEN200
160 FORN=1TOB:GOSUB270
170 VPOKEY,B
180 Y=Y+1:IFYMOD256=X1THENY=Y+256-WIDTH
190 NEXT:GOTO140
200 C=B-128:GOSUB270
210 IFB=0THEN250
220 FORN=1TOC:VPOKEY,B
230 Y=Y+1:IFYMOD256=X1THENY=Y+256-WIDTH
240 NEXT:GOTO140
250 IF(C+YMOD256)>=X1THENY=Y+256-WIDTH
260 Y=Y+C:GOTO140
270 IFM<>0THEN360
280 READA$
290 IFLEN(A$)<>6THENPRINT"Error in line";1000+INT(N/64)*10
300 A=0
310 FORF=0TO5
320 B=ASC(MID$(A$,F+1))-65
330 IFB>25THENB=B-6
340 A=A+C(F)*B
```

```
350 NEXTF
360 IFA=0THENB=0:GOTO380
370 B=INT(A/B(M)):IFB>0THENA=A-B*B(M)
380 M=M+1:IFM=4THENM=0
390 RETURN
400 DATA 16777216,65536,256,1
410 DATA 380204032,7311616,140608,2704,52,1
1000 DATA HnsIWP,ASaMwg,AHXvDz,FxoEIY,AROIFD,JuKifW,AGvrba,AABRjs,IqwDCo,
AEtfrZ,AABXhs,AluIYi,BVHbqI,AHQXMg,FquGbk,AJKKzL
1010 DATA AADBdM,CGLCjG,IYdaLE,FrBeEh,AoJyIC,AKRJSI,AMgiaw,FwhGTE,FoepoQ,
CpmIRQ,AEfgUQ,IdIHKQ,GLeFeY,FtnhcA,ACRWGq,AEfRZg
1020 DATA AEgaLl,LNGfRc,AMCcJt,DZGzUA,AMPuPU,CGLOuI,AIBpIg,HsWqxP,Akqkeo,
AQDMHX,KDWDyI,IYruEQ,FtJWrk,GEiOMx,AHmwiY,FtJWzK
1030 DATA ACQBQw,AUirQY,ATeJsE,HNSlIQ,AFEmZQ,ACRJlu,FisoMQ,FvZYUw,AMgZTb,
AQEVAw,AWxiOs,DJsods,GSkxOf,AAAqYo,ABJbUI,AACFFs
1040 DATA CrWyQg,ACRVxC,AElJtU,AACXNs,BVjNub,AACRTX,AQOrED,ACWxXD,LPVjTy,
LPVgcA,LGoVfg,LIWXbU,AJhDip,ADakrI,FitwGN,AjfoSb
1050 DATA FjWpTk,AABUiX,FvYndQ,IiwbrV,AUAQOp,FiuZEj,BWhgex,IYgePk,ASWPAS,
ADdyWw,AGxlJs,BWAIJs,GbfLip,FqvpyI,FrJOJc,GAKgeo
1060 DATA FsuoAA,AGQMhS,AAAwxE,GycVoj,AtCqMW,AGtzPU,FtYgiY,FomZug,FxvDbb,
FqfXLE,FvYnZD,CZkQsR,LOROrm,IZDhNM,AEwVqu,AOyTNu
1070 DATA AGzpFs,AHBHyo,FquGYg,BMOeDE,AKRFnU,ATdZFy,AQGaTK,KjLYgw,IZmQgC,
IasSmo,AGmbXp,AAHAGJ,ABldzc,AJKKrk,FidfzA,AIFRyJ
1080 DATA FiwLwS,AQGmKb,FofmTU,HEsoPI,AABVYU,AIDxNQ,AJNBsg,FquGUW,AQGmKb,
AJJmkQ,GRQfvY,LPVLRU,AQDMIA,JvWCVc,FilDAU,AGtxyJ
1090 DATA FtJXGY,AAAMIl,AADryI,Akorma,AEfSPU,FqvPEE,HVZrvF,FquSuH,AFlepQ,
GSXrjE,BVlTUA,AFmbhf,CqttgQ,AlwZkl,ACQlZe,AKUidE
1100 DATA AJLJyk,AuLLUg,FvYnYx,ADYTfJ,BXiNKk,LHiyzG,ChudjU,AGuWGI,FpIguo,
FixIVA,AkorwR,FxoEGI,FgeeIF,ASaCDk,Fixgjp,BWfUAC
1110 DATA ACQBao,AGscmL,KxEmEw,CsCtea,CjWODE,ALcSUw,EOPPEA,AClwTE,ASTZUs,
AlAEEw,FjFckz,ACSGKY,BWHnaY,FoioOo,GAhYUw,ALaFtO
1120 DATA GLUoLE,GADUuI,EMdfXo,AlwZiL,BdOlPp,Cwtgyb,AEfFnI,AACRxM,AlATOD,
ABLaPY,ABJbUE,IYdrxf,BWhfGa,AkspxE,GADhEi,AHJsxt
1130 DATA HlcriI,GADUeo,FyvmOJ,FxpNHk,ACswJs,AFmDVS,ASTwQM,GADhIx,AEmetf,
ABJDFS,AEflNc,GSdhZk,GADhUg,FvZjxd,ADZoQC,ACWEeo
1140 DATA AGfEbM,AKUicz,AnIThf,ASYHxi,AAAlmo,IYelAg,BVHbnk,ARoogY,FidgmM,
FofmFe,GADUnW,AEgOZM,AJQnjU,IZqapU,CsDqDF,KfvZyJ
1150 DATA AKSRdQ,ACPvAQ,FvZkCW,FnatFv,AGvrHp,AABjts,CsCVSK,AJKXMh,AZMgeI,
BaMzLe,AOxjTw,HDHCuI,CsBbvG,BMOeDE,AGqMev,AAAwWM
1160 DATA BVYFZt,CqOrIy,AFmnso,AABOjD,FxoEDj,FxoFPj,AAHYcU,AJKjSk,AAdxNM,
FjGxTW,AAAADl,HmCrjF,IbVZBj,CYAkJc,JdvVXD,ItUBzS
```

Machine Code Programming

By Michael Hadrup

- I have reproduced the diagram below of the shift and rotate instructions from the last issue, which was slightly unreadable. (well, you couldn't read it at all).



In's and Out's

This issue is devoted to the hardware side of the SEGA and therefore I will discuss the machine code commands with which we can communicate with the other parts of the SEGA.

The first is IN A,(N) which is similar to the INP() function in Basic. The original value of A provides bits 8-15 and N bits 0-7 of the I/O address. (see "Inside the SEGA" in this issue for more information).

A second form of IN is IN R,(C) where B provides bits 8-15 and C bits 0-7 of the I/O address. Where r is a register B, C, D, E, H, L or A.

Of course there are similar versions of OUT - similar to OUT in Basic - OUT (N),A and OUT (C),R.

In the case of the SEGA only bits 0-7 of the I/O address are used, so you don't have to worry about bits 8-15 held in A or B - depending on which instruction you are using.

Continued on page 10

Directory Program *Disk Basic Only*

By Denver Scott

```
1 REM
2 REM Patterns.Cde file creator
3 REM This creates the file
4 REM "Patterns.Cde" used in the
5 REM Directory program
6 REM Just run it and it saves.
7 REM
8 REM (c) 1988 MJH Software
9 REM
10 PATTERNC#48 ,"708898A8C8887000"
15 PATTERNC#79 ,"7088888888887000"
20 FORN=0TO63
25 A=PEEK(&H5D32+N)
30 IFA=0THEN65
35 IFA=32THEN65
40 B=&H1D00+(A-160)*8
45 C=&H1800+PEEK(&H5DB2+N)*8
50 FORM=0TO7
55 VPOKEB+M,VPEEK(C+M)XOR255
60 NEXTM
65 NEXTN
70 INPUT"Place disk in drive";D$
75 FORN=1TO500:NEXT
80 CALL &H64CA
85 SAVEM"Patterns.Cde",&HA316,&HA316+2047
90 CALL&H64CA
95 PRINT"Finished":END
```

Main Program

The main program listing starts below and is hopefully listed in 38 column format, so that it looks like what you should see on the screen.

The program contains Inverse characters. These are shown using "strike thru" and "~~look like this~~". Inverse characters are typed with the ENG DIER's key.

The character "_" is also used. This is typed in the GRAPH mode with SHIFT-Z.

```
10 SCREEN1,1:CLS:COLOR15,4
20 IF PEEK(&HB02A)>0 THEN 50
30 PRINT"Insufficient maxfile numbe
40 BEEP2:END
50 IF PEEK(&HFFF0)=1THEN100
60 CALL&H64CA
70 LOADM"Patterns.Cde",&HA316
80 CALL&H64CA
90 POKE&HFFF0,1
100 ERASE
105 GOSUB6000
110 A$=CHR$(32):A1$=A$+A$:B$=""
120 M$(1)="__________
         NAME      : "
130 M$(2)="__________
         STREET    : "
140 M$(3)="__________
         SUBURB    : "
150 M$(4)="__________
         CITY      : "
160 M$(5)="__________
         PHONE     : "
170 T$(1)="_____________________
         READ FROM DIRECTORY"
180 T$(2)="____________________
         WRITE TO DIRECTORY"
190 T$(3)="______________________
         DELETE FROM DIRECTORY"
200 T$(4)="________________________
         LIST NAMES IN DIRECTORY"
210 T$(5)="_____________________
         CLOSE THE DIRECTORY"
220 CURSOR1,6:PRINT"___________________
_________________   SEGA DISK BASED A
DDRESS DIRECTORY"
230 CURSOR6,9:PRINT"1988 Version By D.
Scott"
240 CURSOR1,11:PRINT"Please wait for f
ile initialisation"
250 DIM N$(300)
260 OPEN "A Direct.DTA"AS #1
270 IF LOF(1)=0THEN2900
280 CLOSE
290 OPEN"B Direct.DTA"FOR INPUT AS #1
```

```
300 FORN=1TO300:INPUT#1,N$(N):NEXTN
310 CLOSE
320 SFUD=0
330 BEEP:CLS
340 CURSOR0,0:PRINT"____________________
                   ADDRESS DIRECTORY
_
_"
350 CURSOR0,3:PRINT"______
                   MENU "
360 PRINT:PRINT"___"
370 PRINT"{1}    Read from Directory":P
RINT"___"
380 PRINT"{2}    Write to Directory":PR
INT"___"
390 PRINT"{3}    Delete from Directory"
:PRINT"___"
400 PRINT"{4}    List names in Director
y":PRINT"___"
410 PRINT"{5}    Close the Directory"
420 PRINT:PRINT"                  _____
              Select numbe{1-5}"
430 J$=INKEY$:IFJ$<"1"ORJ$>"5"THEN430
440 ONVAL(J$)GOTO 1000,2000,3000,4000,
5000
1000 REM READ FROM DIRECTORY
1010 BEEP:CLS:PRINTT$(1)
1020 GOSUB 1300
1030 IF F=0 THEN PRINT:PRINT"Name not
on file":GOTO1050
1040 GOSUB1500:GOSUB1200
1050 GOSUB1400
1060 J$=INKEY$
1070 IFJ$=CHR$(13)THEN1000
1080 IFJ$="M"THEN 330
1085 IFJ$="P"THEN 7000
1090 GOTO1060
1200 REM CHECK FOR PHONE NUMBER
1210 FORL=1TOLEN(P$)
1220 IFMID$(P$,L,1)>CHR$(0)THENR=1:L=L
EN(P$)+1
1230 NEXTL:RETURN
1300 REM SEARCH FOR NAME
1310 CURSOR0,5:INPUT"___________
                   NAME      :  ";K$
1315 F=0:PRINT:PRINT"Please wait   Sear
ching Directory"
1320 FORN=1TO300
1330 IF K$=LEFT$(N$(N),LEN(K$))THENF=N
:N=301
1340 NEXTN
1350 RETURN
1400 REM MENU OPTION
```

```
1410 CURSOR0,21:PRINT"____            _
__                      {CR}  New name{
M}  Menu ";:IF R=1THENPRINTCHR$(30);"_
__";CHR$(31);CHR$(29);CHR$(29);CHR$(29
);"{P}  Phone":R=0
1420 RETURN
1500 REM GET AND DISPLAY FILE
1510 OPEN"A Direct.DTA"AS#1
1520 GET#1,F;S$,0,30;D$,30,30;C$,60,30
;P$,90,30
1530 CLOSE
1540 PRINTM$(1);N$(F)
1550 PRINTM$(2);S$
1560 PRINTM$(3);D$
1570 PRINTM$(4);C$
1580 PRINTM$(5);P$
1590 RETURN
2000 REM WRITE TO DIRECTORY
2010 BEEP:CLS:PRINTT$(2)
2020 REM SEARCH SEQUENTIAL FILE
         FOR FREE SPACE
2030 PRINT:PRINT:PRINT"Please wait   Se
arching for free space"
2040 FORN=1TO300:IF N$(N)="
         "OR N$(N)=" "OR N$(N)=""THEN F=
N:N=305
2050 NEXT N
2060 IF N=301 THEN CURSOR0,8:PRINT"Fil
e full  No free space":FORD=1TO750:NEX
TD:GOTO 330
2070 REM DATA ENTRY
2080 CURSOR0,8:INPUT"___________
                   NAME      :   ";N$
(F)
2090 INPUT"__________
          STREET    :   ";S$
2100 INPUT"__________
          SUBURB    :   ";D$
2110 INPUT"__________
          CITY      :   ";C$
2120 INPUT"__________
          PHONE     :   ";P$
2140 OPEN"A Direct.DTA"AS#1
2150 PUT#1,F;S$,0,30;D$,30,30;C$,60,30
;P$,90,30
2160 CLOSE:SFUD=1
2170 GOSUB1400
2180 J$=INKEY$
2190 IF J$=CHR$(13)THEN2000
2200 IF J$="M"THEN330
2210 GOTO2180
2900 REM OPEN MAIN RANDOM FILE FOR
```

```
        THE FIRST TIME
2910 PUT#1,300;S$,0,30;D$,30,30;C$,60,30;P$,90,30:CLOSE
2920 REM CLEAR MEMORY FOR SEQUENTIAL
2930 FORN=1TO 300:N$(N)="":NEXT N
2940 GOSUB5110
2950 GOTO320
3000 REM DELETE FROM DIRECTORY
3010 BEEP:CLS:PRINTT$(3)
3020 GOSUB1300
3030 IF F=0THEN PRINT:PRINT"Name not on file":GOTO3150
3040 GOSUB1500
3050 CURSOR0,21:PRINT"___ ___ {Y} Delete {N} Remain"
3060 J$=INKEY$
3070 IF J$="Y"THEN 3100
3080 IF J$="N"THEN BEEP:GOTO3150
3090 GOTO3060
3100 BEEP
3110 OPEN "A Direct.DTA"AS#1
3120 PUT#1,F;A$,0,30;A$,30,30;A$,60,30;A$,90,30
3130 CLOSE
3140 N$(F)=B$:SFUD=1
3150 GOSUB1400
3160 J$=INKEY$
3170 IF J$=CHR$(13) THEN3000
3180 IF J$="M"THEN330
3190 GOTO3160
4000 REM LIST NAMES ON FILE
4010 A=1:B=30:C=1
4020 BEEP:CLS:PRINTT$(4):PRINT:PRINT"Page ";C;" of 10":PRINT
4030 FOR X=ATOB
4040 PRINTN$(X),:NEXT
4050 A=A+30:B=B+30:C=C+1
4060 CURSOR0,21:PRINT"_____ _ __ {CR} Continue {M} Menu"
4070 J$=INKEY$
4080 IF J$=CHR$(13)AND B=330 THEN4010
4090 IF J$=CHR$(13) THEN4020
4100 IF J$="M"THEN330
4110 GOTO4070
5000 REM CLOSE FILE
5010 BEEP:CLS:PRINTT$(5)
5020 IF SFUD <>1 THEN 5040
5030 CURSOR0,6:PRINT"Closing Directory ":CURSOR0,10:PRINT"Please wait while disk is updated":GOSUB5050
5040 CURSOR 0,15:PRINT"Directory closed":BEEP:END
5050 KILL "B Direct.DTA"
5060 GOSUB5110
5100 REM SEQUENTIAL FILE UPDATE
5110 OPEN "B Direct.DTA"FOROUTPUTAS#1
5120 FORN=1TO300:PRINT#1,N$(N):NEXTN
5130 CLOSE
5140 RETURN
6000 REM SET UP TELEPHONE CODES
6010 RESTORE 6040
6020 FORN=1TO10:READT(N),T1(N):NEXTN
6030 RETURN
6040 DATA 941,1209
6050 DATA 697,1209
6060 DATA 697,1336
6070 DATA 697,1447
6080 DATA 770,1209
6090 DATA 770,1336
6100 DATA 770,1477
6110 DATA 852,1209
6120 DATA 852,1336
6130 DATA 852,1477
7000 REM CALL TELEPHONE NUMBER
7010 IF LEN (P$)=0THEN7100
7020 FORI=1TOLEN(P$)
7030 A=ASC(MID$(P$,I,1))-47
7040 IF (A<1)OR(A>10)THEN7090
7050 SOUND1,T(A),15
7060 SOUND2,T1(A),15
7070 FORQ=1TO70:NEXTQ
7080 SOUND0
7090 NEXTI
7100 R=1:GOTO1050
```

Mulicolour Mode Demo Program

```
1 REM Multi colour mode
2 REM
3 REM Demonstration
4 REM
5 REM
6 REM By Michael Hadrup
7 REM
8 REM
10 X=&HF000
20 READA$:IFA$="*"THEN90
30 POKEX,VAL("&h"+A$):X=X+1:GOTO20
40 DATA F3,DB,BF,0,0,0,0,0
```

```
50 DATA 3E,0,D3,BF,3E,80,D3,BF,0,0,0,3E,E8,D3,BF,3E,81,D3,BF
60 DATA 3E,E,D3,BF,3E,82,D3,BF,0,0,0,3E,0,D3,BF,3E,84,D3,BF
70 DATA C9
80 DATA *
90 SCREEN2,2:COLOR,,,15:CLS
100 CALL&HF000:REM Change screen mode
110 IF(VPEEK(&H3820)=0)THENGOTO210
120 REM
130 REM Set up the name table
140 REM unless already set up
150 REM
160 X=&H3800
170 FORN=0TO7:FORM=0TO3:FORF=0TO31
180 VPOKEX,F+N*32:X=X+1
190 NEXTF,M,N
200 REM
210 COL=1
220 X=5:Y=1:A$="HELLO":GOSUB410
230 X=7:Y=10:A$="A demo of":GOSUB450
240 X=7:Y=20:A$="the multi":GOSUB450
250 X=16:Y=30:A$="colour":GOSUB450
260 X=22:Y=40:A$="mode":GOSUB450
270 BEEP:BEEP
280 GOTO280
290 REM
300 REM This routine prints a string
310 REM on the multi colour screen
320 REM
330 REM It uses the bottom of the
340 REM graphics screen as a copy
350 REM area for the character bit
360 REM maps
370 REM
380 REM
390 REM Double width printing
400 REM

410 BLINE(0,184)-(255,191),,BF:CURSOR0,184:PRINTCHR$(17);A$:A$=A$+A$:GOTO
460
420 REM
430 REM Single width printing
440 REM
450 BLINE(0,184)-(255,191),,BF:CURSOR0,184:PRINTCHR$(16);A$
460 L=1+INT(LEN(A$)*6/8):Z=5888

470 REM
480 REM L is the width of the string
490 REM to nearest 8 pixels
500 REM
510 FORN=0TOL:FORM=0TO7
520 V=VPEEK(Z+M+N*8):RESTORE525
525 DATA 128,64,32,16,8,4,2,1
530 REM
540 REM M is each vertical line
550 REM V is the 8 pixels of the line
560 REM
570 FORI=0TO7:READD
580 REM
590 REM Plot the point if necessary
600 REM
610 IFV-D>=0THENV=V-D:GOSUB730
620 COL=COL+2:IFCOL>14THENCOL=COL-14
630 X=X+1:NEXTI
640 COL=COL-4:IFCOL<1THENCOL=COL+14
650 X=X-8:Y=Y+1:NEXTM
660 COL=COL+4:IFCOL>14THENCOL=COL-14
670 X=X+8:Y=Y-8:NEXTN
680 RETURN
690 REM
700 REM The multi colour plot routine
710 REM Plots at (X,Y), colour COL
720 REM
730 ADDR=INT(X/2)*8+YMOD8+INT(Y/8)*256

740 IFXMOD2=0THENVPOKEADDR,(VPEEK(ADDR)AND15)+COL*16:RETURN
750 VPOKEADDR,(VPEEK(ADDR)AND240)+COL:RETURN
```

The routine sets up the Multicolour mode.

M1=0, M2=1, M3=0

Name table at #3800

Pattern Generator Table at #0000

Note that the Basic program sets up the Multicolour mode using Name Table overlapping as described on page 20 of this issue.

```
F3          DI                       DISABLE INTERRUPTS
DBBF        IN A,(#BF)               CLEAR STATUS REGISTER
0000000000  NOP:NOP:NOP:NOP:NOP      WAIT A WHILE
3E00        LD A,0                   DATA FOR REGISTER 0
D3BF        OUT (#BF),A
3E80        LD A,#80
D3BF        OUT (#BF),A              OUTPUT TO VDP REGISTER 0
EE8         LD A,#E8                 DATA FOR REGISTER 1
D3BF        OUT (#BF),A
3E81        LD A,#81
D3BF        OUT (#BF),A              OUTPUT TO VDP REGISTER 1
3E0E        LD A,#0E                 DATA FOR REGISTER 2
D3BF        OUT (#BF),A              OUTPUT TO VDP REGISTER 2
3E82        LD A,#82
D3BF        OUT (#BF),A              OUTPUT TO VDP REGISTER 2
3E00        LD A,0                   DATA FOR REGISTER 4
D3BF        OUT (#BF),A
3E84        LD A,#84
D3BF        OUT (#BF),A              OUTPUT TO VDP REGISTER 4
C9          RET
```

Graphics Mode 1 Demo Program

```
1 REM Graphics mode 1
2 REM
3 REM Demonstration
4 REM
5 REM
6 REM By Michael Hadrup
7 REM
8 REM
9 REM
10 X=&HF000
20 READA$:IFA$="*"THEN90
30 POKEX,VAL("&h"+A$):X=X+1:GOTO20
40 DATA F3,DB,BF,0,0,0,0,0
50 DATA 3E,0,D3,BF,3E,80,D3,BF,3E,E0,D3,BF,3E,81,D3,BF
60 DATA 3E,F,D3,BF,3E,82,D3,BF,3E,3,D3,BF,3E,84,D3,BF
70 DATA C9
80 DATA *
90 SCREEN1,1:COLOR1,15:CLS
100 CALL&HF000:REM Change screen mode
110 REM
120 REM Set up the colour table
130 REM
140 A=2:FORN=&H3FC0TO&H3FFF
150 VPOKEN,A+16
160 A=A+1:IFA=15THENA=2
170 NEXT
180 REM
190 REM Set "@"-"W" to white on black
200 REM
210 FORN=&H3FC8TO&H3FCA
220 VPOKEN,&HF1:NEXT
230 REM
240 X=7:Y=10:A$="A DEMONSTRATION OF":
GOSUB360
250 X=9:Y=13:A$="GRAPHICS MODE I":GOS
UB360
260 REM
270 FORN=0TO255
280 VPOKE&H3C00+N,N
290 VPOKE&H3E00+N,RND(1)*256
300 NEXT
310 GOTO310
320 REM
330 REM This routine prints a string
340 REM at (X,Y)
350 REM
360 ADDR=X+Y*32+&H3C00
370 FORN=1TOLEN(A$)
380 VPOKEADDR+N-1,ASC(MID$(A$,N,1))
390 NEXT:RETURN
```

Scanning The Keyboard

		Bits 0-2 of Port C (#DE)				Keyboard			Joystick
	Bit	0	1	2	3	4	5	6	7
Port A (#DC)	0	1	2	3	4	5	6	7	1 Up
	1	Q	W	E	R	T	Y	U	1 Down
	2	A	S	D	F	G	H	J	1 Left
	3	Z	X	C	V	B	N	M	1 Right
	4	DIER	SPC	CLR	DEL				1 Shot L
	5	,	.	/	Π	Down	Left	Right	1 Shot R
	6	K	L	;	:	]	CR	Up	2 Up
	7	I	O	P	@	[			2 Down
Port B (#DD)	0	8	9	0	-	^	¥	Break	2 Left
	1							Graph	2 Right
	2							Ctrl	2 Shot L
	3						Func	Shift	2 Shot R

Tables similar to this have been published in previous magazines, but never explained very well. If you have read the information on the PPI then you will recognise the ports used above as those involving the PPI.

Only one vertical row of the keyboard matrix above can be read at a time and you must tell the keyboard which row you want to check. To do this you output a number between 0 and 7 to Port C, which is #DE. You can then read from #DC or #DD (depending on which set you wish to check) and examine individual bits. A 0 means a key is down, and 1 means a key is up.

When the matrix is blank, there is no key to check. The value of the bit can be either a 1 or 0 depending on whether you have a soft or hard keyboard (and how old it is), so don't assume that the unused bits will be a 0 or 1. They can be either!

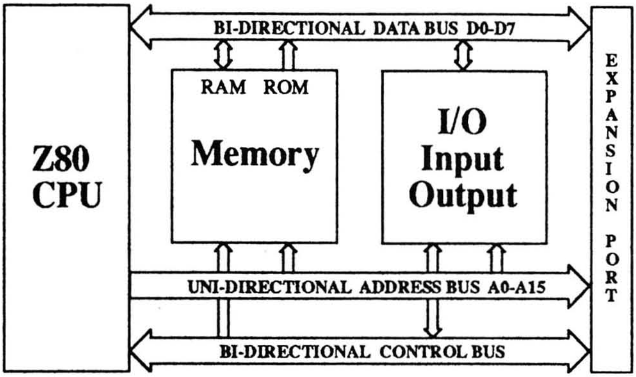
For example if you want to check for the Space Bar being pressed and released in machine code, you could wirte the following ...

```
        LD A,1
        OUT (#DE),A     Set vertical row 1
LOOP    IN A,(#DC)      Read Port A
        BIT 4,A         Check for space
        JR NZ,LOOP      If up then wait
LOOP1   IN A,(DC)       Read Port A
        BIT 4,A         Check for space
        JR Z,LOOP1      If still down then wait
        RET
```

Inside the SEGA

By Michael Hadrup

HARDWARE



The SEGA computer can be divided into three separate parts - CPU, Memory and Input and Output (I/O).

It is important to note that the address bus is used to access memory locations and Input/Output devices. Which type of device is determined by the control bus.

PART 1 - THE BASIC SEGA

The basic SEGA (one with no cartridge or Super Control Station connected) contains the following main chips ....

IC2 - ADDRESS DECODER

This is a special chip that handles a number of functions. Mainly it controls selection of I/O devices. If A4 is low then internal I/O devices are disabled - for use with Super Control Station. The chip also handles disabling of IC3 - System Ram with the DSRAM signal on B3 of the expansion port. It also produces the Non-maskable interrupt when the Reset key is pressed. I/O devices are only addressed using A5-A7 giving 8 possible devices.

The following I/O device is selected given A4-A7

I/O address	A7-A4	I/O Device
#7F	1111	IC4 - Sound Chip
#BE, #BF	1011	IC9 - Visual Display Processor
#DC - #DF	1101	IC5 - Parallel Peripheral Interface

IC4 - SOUND CHIP

This has already been well documented in previous magazines and will not be discussed now. For more information refer to the extracts from Brian Brown's SEGA Programmers Manual. See page 10 of November 1986 / February 1987 issue of SEGA Computer (produced by SEGA Software Support).

IC9 - Visual Display Processor

The VDP in the SEGA is a Texas Instruments 9929A chip (now produced by Western Digital). The VDP has its own 16K of RAM in which display information is stored, called VRAM (video ram). The CPU cannot access VRAM directly but must use the VDP to access VRAM. It accesses through I/O ports #BE (data port) and #BF (command port). The VDP has eight 8 bit write-only registers, an 8 bit read-only status register and a 14 bit auto-incrementing address register.

The address register is 14 bits long because it points to an address in VRAM between 0 and #3FFF (16K). Auto-increment means that when you write or read, it automatically adds one to itself and points to the next byte in VRAM.

The VDP has FOUR basic operations ...

Write to VRAM

Before you can send data to VRAM you must set up the 14 bit address register for writing to a location in VRAM. This involves the output of two bytes to the command port. The first is the low byte and the second is the 6 bits of the high byte with bit 6 set (ie + #40) and bit 7 reset. Once the address register has been set, bytes can be sent to the data port and they are stored consecutively in VRAM.

Read from VRAM

Before you can read data from VRAM you must set up the 14 bit address register for reading from a location in VRAM. This involves the output of two bytes to the command port. The first is the low byte and the second is the 6 bits of the high byte and bits 6 and 7 reset. Once the address register has been set, bytes can be read from the data port in consecutive order from VRAM.

Write to VDP write-only Register

These registers control the VDP operation and determine they way VRAM is allocated. To change a VDP register involves the output of two bytes to the command port. The first is the data byte (the new value) and the second is made up of bits 0-2 (which register number 0-7), bits 3-6 reset and bit 7 set. (ie + #80). Note that the address register is destroyed when a VDP register is changed.

Read from read-only VDP Status Register

This register contains flags on interrupts, sprite collision and the fifth sprite. It is also used to reset the transfer of bytes. For example if you were changing a VDP register and the transfer of the first data byte was complete (but not the second) and an NMI interrupt occurs (reset key pressed), the VDP is left hanging. The next byte sent would be interpreted as a register destination. A read of the status register resets the VDP, so that the next byte will be interpreted correctly.

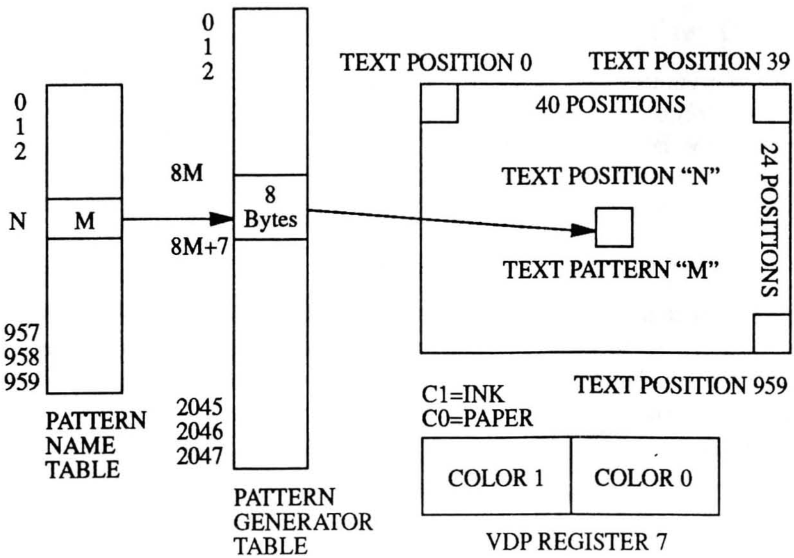
OPERATION		0	1	2	3	4	5	6	7
Write to VRAM									
Byte 1	Address Setup	A7	A6	A5	A4	A3	A2	A1	A0
Byte 2	Address Setup	0	1	A13	A12	A11	A10	A9	A8
Byte 3	Data Write	D7	D6	D5	D4	D3	D2	D1	D0
Byte N	Date Write	D7	D6	D5	D4	D3	D2	D1	D0
Read from VRAM									
Byte 1	Address Setup	A7	A6	A5	A4	A3	A2	A1	A0
Byte 2	Address Setup	0	0	A13	A12	A11	A10	A9	A8
Byte 3	Data Read	D7	D6	D5	D4	D3	D2	D1	D0
Byte N	Date Read	D7	D6	D5	D4	D3	D2	D1	D0
Write to VDP Register									
Byte 1	Data	D7	D6	D5	D4	D3	D2	D1	D0
Byte 2	Register Select	1	0	0	0	0	R2	R1	R0

The Four Screen Modes

Text mode

In this mode, the screen is divided into a grid of 40 text positions across by 24 down. (normally only 38 columns are visible). Each of the text positions contains 6 pixels across by 8 pixels down. The tables in VRAM used to generate this display are the Pattern Name Table and the Pattern Generator Table which occupy 3008 VRAM bytes.

TEXT MODE MAPPING



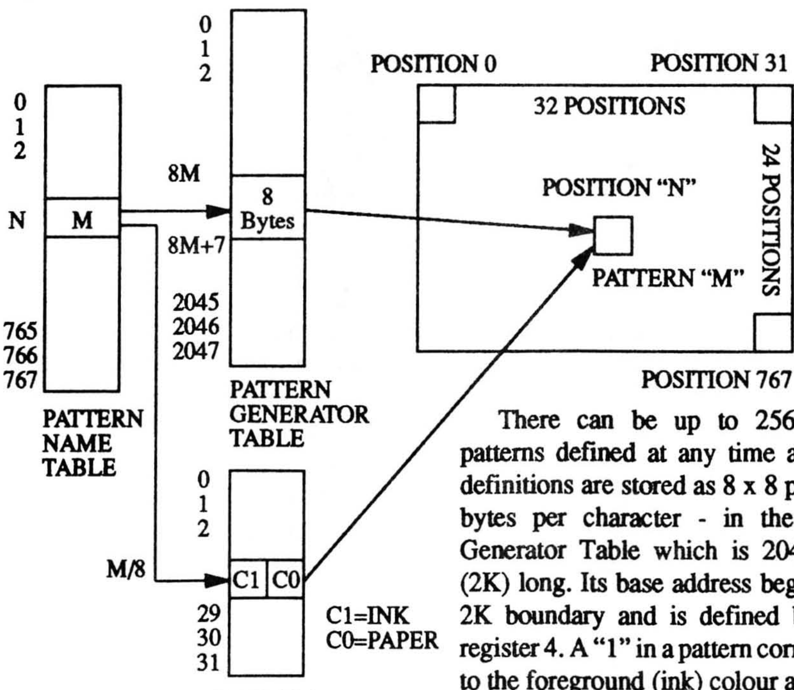
There can be up to 256 unique text patterns (characters) defined at any time and their definitions are stored as 8 x 8 pixels (bits 0 and 1 are ignored) - 8 bytes per character - in the Pattern Generator Table which is 2048 bytes (2K) long. Its base address begins on a 2K boundary and is defined by VDP register 4. Only two colours are available for the whole screen including the backdrop (border) and they are defined by VDP register 7 (background+foreground colour*16). A "1" in a text pattern corresponds to the foreground (ink) colour and a "0" corresponds to the background (paper) colour.

The Pattern Name Table holds a map of which text pattern (character) is to be displayed and is 960 (40x24) bytes long. Its base address begins on 1K boundary and is defined by VDP register 2. The first 40 bytes correspond to the top row, the next 40 to the second row and so on. Sprites are not available in the Text mode.

Graphics Mode 1 - Not available in Basic

In this mode, the screen is divided into a grid of 32 columns across by 24 rows of pattern positions. Each of the pattern positions contains 8 pixels across by 8 pixels down. The tables in VRAM used to generate this display are the Pattern Name Table, Pattern Generator Table and Colour Table which occupy 2848 VRAM bytes.

GRAPHICS MODE 1 MAPPING



There can be up to 256 unique patterns defined at any time and their definitions are stored as 8 x 8 pixels - 8 bytes per character - in the Pattern Generator Table which is 2048 bytes (2K) long. Its base address begins on a 2K boundary and is defined by VDP register 4. A "1" in a pattern corresponds to the foreground (ink) colour and a "0" corresponds to the background (paper) colour.

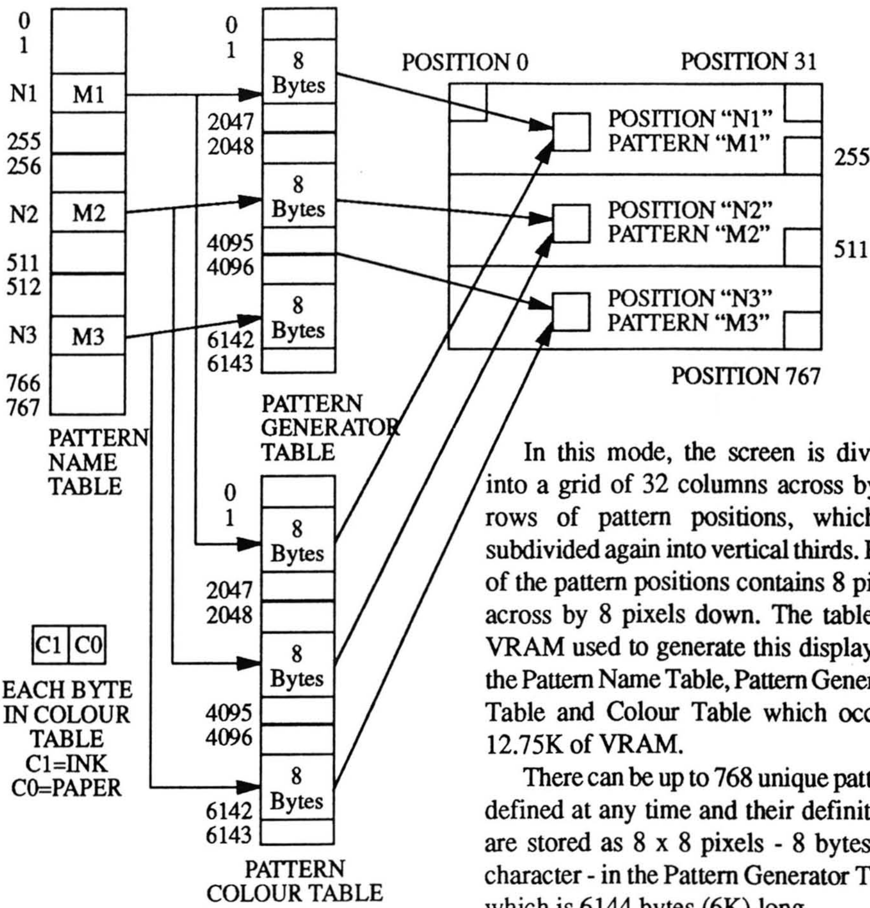
The colours of the "1"'s and "0"'s are defined by the Colour Table which has 32 entries, each of which is one byte long. (background*16+foreground colour). The first entry in the colour table defines the colours for patterns 0-7, the next for patterns 8-15 and so on. Its base address begins on a 64 byte boundary and is defined by VDP register 3.

The Pattern Name Table holds a map of which pattern is to be displayed and is 768 (32x24) bytes long. Its base address begins on 1K boundary and is defined by VDP register 2. The first 32 bytes correspond to the top row, the next 32 to the second row and so on. Sprites are available in Graphics Mode 1.

Graphics Mode 2 - The normal Graphics Screen

This mode is similar to Graphics Mode 1 except it allows for more patterns - 768 (32x24) and additional colour information is included for each pattern.

GRAPHICS MODE 2 MAPPING



In this mode, the screen is divided into a grid of 32 columns across by 24 rows of pattern positions, which is subdivided again into vertical thirds. Each of the pattern positions contains 8 pixels across by 8 pixels down. The tables in VRAM used to generate this display are the Pattern Name Table, Pattern Generator Table and Colour Table which occupy 12.75K of VRAM.

There can be up to 768 unique patterns defined at any time and their definitions are stored as 8 x 8 pixels - 8 bytes per character - in the Pattern Generator Table which is 6144 bytes (6K) long.

It is separated into three blocks of 256 patterns, each a vertical third of the screen so that patterns in the top third are found in the first 2048 bytes and so on for each third. Its base address begins on a 8K boundary and is defined by VDP register 4. It may be located in the lower or upper half of VRAM. VDP register 4 contains 0 for the lower or 255 for the upper half. A "1" in a pattern corresponds to the foreground (ink) colour and a "0" corresponds to the background (paper) colour for that pattern.

The colours of the "1"'s and "0"'s are defined by the Colour Table which is 6144 (6K) long. Each of which is one byte long. (background*16+foreground colour). The first entry in the colour table defines the colours for the corresponding pattern in the Pattern Generator Table. Its base address begins on an 8K boundary and is defined by VDP register 3. It may be located in the lower or upper half of VRAM. VDP register 3 contains 0 for the lower or 255 for the upper half. (opposite to Pattern Generator Table).

The Pattern Name Table holds a map of which pattern is to be displayed and is 768 (32x24) bytes long. It is segmented into three blocks of 256 names so that names in the top third point to patterns found in the first 2048 bytes in the Pattern Generator Table and so on. Its base address begins on a 1K boundary and is defined by VDP register 2. The first 32 bytes correspond to the top row, the next 32 to the second row and so on. Sprites are available in Graphics Mode 2.

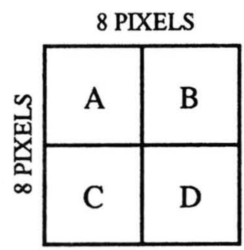
MultiColour Mode - Not available in Basic

The MultiColour mode provides an unrestricted 64 x 48 colour square display. Each colour square contains 4 x 4 pixels. The tables in VRAM used to generate this display are the Pattern Name Table and Pattern Generator Table which ouccupy 2816 bytes (2.75K) of VRAM.

The Name Table is the same as in the other modes, consisting of 768 entries for each of the 32 x 24 positions. It points to an 8 byte segment in the Pattern Generator Table. Only two bytes are used and these specify the colour of a 2x2 block area (8x8 pixels).

BYTE 1	COLOUR A	COLOUR B
BYTE 2	COLOUR C	COLOUR D

2 BYTES FROM
PATTERN GENERATOR TABLE



MULTICOLOUR
PATTERN

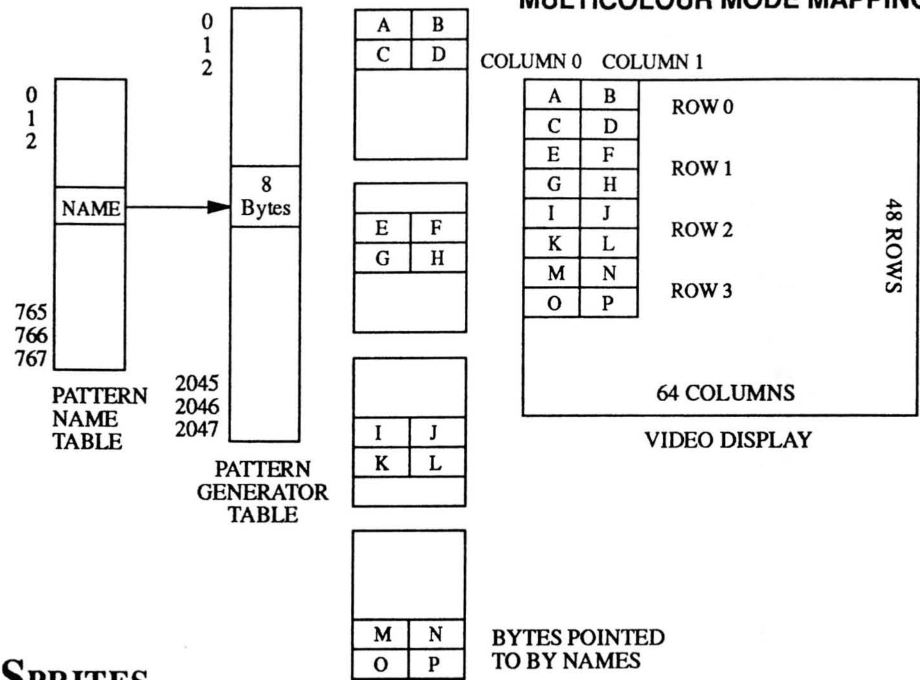
The location of the 2 bytes within the 8 byte segment pointed to by the name is dependant upon the screen position where the name is mapped. For names in the top row (names 0-31), the 2 bytes are the first two. The next row of names (32-63) uses bytes 3 and 4 within the 8 byte segment. The next (64-95) use bytes 5 and 6, while the last row uses bytes 7 and 8. This series repeats for the remainder of the screen.

Thus the colour displayed from a 8 byte segment of the Pattern Generator is dependant upon its position on the display.

Because the colour displayed can differ depending upon the position, the mapping can be simplified by using duplicate names in the Name Table. Each group of 4 rows points to the same set of multicolour patterns as follows. Names 0-31, 32-63, 64-95 and 96-127, point to the multicolour patterns 0-31 and Names 128-159, 160-191, 192-223 and 224-255, point to the multicolour patterns 32-63 and so on.

This now requires only 1536 bytes for the Pattern Generator Table. Its base address begins on a 2K boundary and is defined by VDP register 4. The base address of the Name Table begins on an 1K boundary and is defined by VDP register 2. Sprites are available in the Multicolour Mode.

MULTICOLOUR MODE MAPPING



SPRITES

The video display can have up to 32 sprite planes. The location of a sprite is defined as the top left hand corner. A sprite can be moved by redefining the sprite origin. Sprites are transparent outside of the pattern itself. Sprites are not available in the text mode.

The blocks in VRAM that define the sprites are the Sprite Attribute Table and Sprite Generator Table. As there are 32 sprites there are 32 entries in the Attribute Table each occupying four bytes. This table is 4x32 = 128 bytes long and is located on a 128 byte boundary defined by VDP register 5.

The first two bytes determine the position of the sprite.

The first byte holds the vertical position from the top of the screen such that -1 (or 255) puts the sprite at the very top. The second byte holds the horizontal position such that 0 is at the left edge of the screen.

When the position of a sprite extends into the backdrop area that portion of the sprite is not displayed. This allows sprites to move on to the screen from behind the backdrop or border.

- Vertical values from -8 to -1 allow a sprite to blend in from the top of the screen.
- Vertical values from 183 to 191 allow a sprite to move off the bottom of the screen
- Horizontal values from 248-255 allow a sprite to move off the right of the screen

There are no horizontal values to allow a sprite to blend in from the left of the screen. However a special bit is provided called the Early Clock Bit. When this is set a sprite jumps 32 pixels to the left. To make a sprite blend in from the left we set the EC bit and move it horizontally from 1 to 31, then reset the EC bit and set the horizontal position to 0.

BYTE			
0	VERTICAL POSITION		
1	HORIZONTAL POSITION		
2	NAME		
3	EARLY CLOCK	0 0 0	COLOUR BITS 0-3

SPRITE ATTRIBUTE TABLE ENTRY

Byte 3 of the Attribute Table holds the name of the sprite pattern held in the Pattern Generator Table.

Byte 4 contains the colour of the sprites in bits 0-3 and holds the Early Clock bit in bit 7.

The Sprite Generator Table contains 256 patterns of 8 bytes and is 2048 (2K) long. It base address begins on a 2K boundary defined by VDP register 6.

There is a maximum limit of four sprites that can be displayed on a horizontal line. If this rule is violated then the four highest priority sprites are displayed and fifth and subsequent sprites are not. The fifth sprite bit in the VDP status register is set and the number of the violating fifth sprite plane is placed in the status register.

Larger sprites than 8x8 pixels can be used. The MAG and SIZE bit of VDP register 1 are used to select various options.

MAG 0, SIZE 0	No options selected. (8 byte pattern)
MAG 1, SIZE 0	The size of each pixel is doubled when displayed creating 16x16
MAG 0, SIZE 1	32 bytes (4 patterns are used to display 16x16 pixels).
MAG 1, SIZE 1	32 bytes are used and each is doubled when displayed giving 32x32

PATTERN A
PATTERN B
PATTERN C
PATTERN D

SPRITE GENERATOR PATTERNS

A	C
B	D

SIZE 1 SPRITES

The VDP also checks sprite collision (called coincidence checking). The coincidence flag in the status register is set if any two sprites have bits active at the same screen location.

Sprite processing is terminated if a value of 208 is found in the vertical position of any sprite Attribute entry. If all sprites are to be blanked, then simply place a 208 in the vertical position of the first sprite.

A total of 2176 bytes in VRAM are required for Sprite generation. If all 256 sprite patterns are not needed then tables can be overlaped to reduce the amount of VRAM required.

The designing of sprites has been described before, so I won't go into any more detail about sprites.

Write Only Registers

Register 0	Contains one VDP operation bit. All the other bits must be zero and are reserved for future use.
BIT 0	M3
Register 1	Contains 7 VDP operation bits
BIT 0	MAG bit, 0 selects 1X, 1 selects 2X magnification
BIT 1	SIZE bit, 0 selects 8x8 bits, 1 selects 16x16 bits
BIT 2	Reserved must be zero
BIT 3	M2
BIT 4	M1
BIT 5	IE (Interupt enable), 0 disable interrups, 1 enables interrupts
BIT 6	BLANK, 0 blanks display, 1 enables active display
BIT 7	4/16K VRAM, 1 selects 16K VRAM (always use 1)

M1	M2	M3	
0	0	0	Graphics Mode 1
0	0	1	Graphics Mode 2
0	1	0	Multicolour Mode
1	0	0	Text mode

Register 2	Name Table Base Address = #400 muliplied by value in register 2
0	#0000
1	#0400 ...
14	#3800
15	#3C00
Register 3	Colour Table Base Address = #40 muliplied by value in register 3
0	#0000 - Special case for Graphics Mode 2
1	#0040 ...
#FE	#3F80
#FF	#3FC0
Register 4	Pattern Generator Table Base Address = #800 muliplied by value in register 4
0	#0000 - Special case for Graphics Mode 2
1	#0800 ...
6	#3000
7	#3800

Register 5	Sprite Attribute Table Base Address = #80 muliplied by value in register 5
0	#0000
1	#0080 ...
#7E	#3F00
#7F	#3F80
Register 6	Sprite Generator Table Base Address = #800 muliplied by value in reg 6
0	#0000
1	#0800 ...
6	#3000
7	#3800
Register 7	Bits 4-7 are text colour (foreground)
	Bits 0-3 are text colour (background) and backdrop colour

Status Register (Read only)

A read of the status register always clears the address register and all flags.

Bit 7	Interrupt flag - Set at end of each raster scan pending an interrupt. This must be read every interrupt in order to clear the interrupt and receive the new interrupt for the next frame. This is why we must disable interrupts before writing or reading to VRAM. At the end of every interrupt the status register is read and the address register is cleared, resulting in some strange effects if not disabled.
Bit 6	Fifth sprite flag - This is set to 1 whenever five or more sprites occur on a horizontal line. The number of the fifth sprite is placed in the status register whenever this is set to one.
Bit 5	Coincidence flag - This is set to 1 if two or more sprites collide. Coincidence occurs if any two sprites have an overlapping pixel. Transparent sprites as well as those partially or completely of screen are also considered. Spritres beyond the Attribute Table Terminator (a 208 in the vertical position) are not considered.
Bits 0-4	Fifth Sprite Number

IC5 - Programmable Peripheral Interface

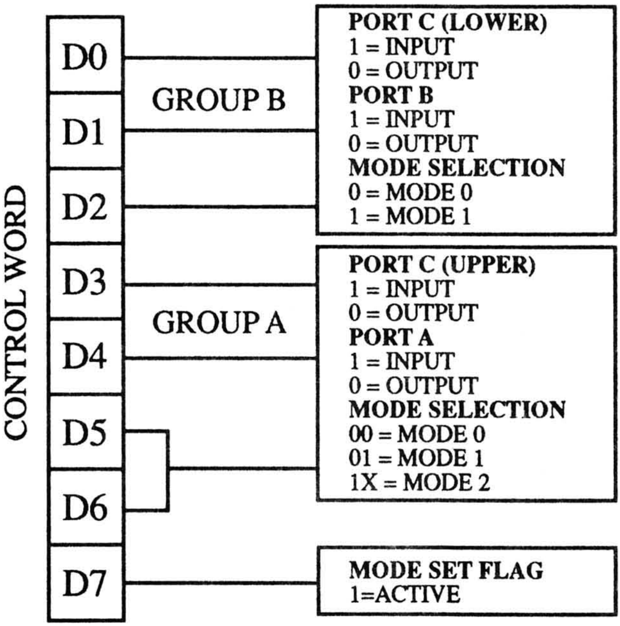
The 8255A is a general purpose programmable I/O device. It has 24 I/O pins which may be individually programmed in two groups of twelve and used in three modes of operation. In Mode 0, each group of twelve pins may be programmed in sets of 4 as input or output.

In Mode 1, each group may be programmed as 8 lines of input or output (strobed). The remaining pins are used for handshaking and interrupt control signals.

In Mode 2, a bi-directional mode, which uses 8 lines for the bi-directional bus and five lines (one from the other group) for handshaking.

Port	Description
#DC	Port A - One 8 bit data output latch/buffer and one 8 bit data input latch
#DD	Port B - One 8 bit data input/output latch/buffer and one 8 bit data input buffer
#DE	Port C - One 8 bit data input/output latch/buffer and one 8 bit data input buffer (no latch for input). This port can be divided into two 4 bit ports under the mode control. Each 4 bit port contains a 4 bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with A and B.
#DF	Control register (write only)

MODE DEFINITION FORMAT



Initially when RESET each port will be set to input (high impedance state). The modes for Port A and Port B can be defined separately, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers including the status flip-flops, will be reset whenever the mode is changed.

Single Bit Set/Reset Feature

Any of the eight bits of Port C can be Set or Reset using a single output. When Port C is being used as status/ control for Port A or B, these bits can be Set or Reset using this operation just as if they were data outputs.

Mode 0 Port Definition Chart

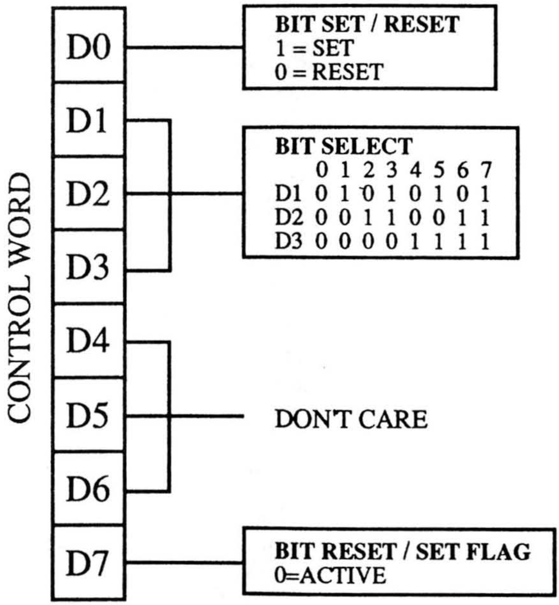
	A		B		Group A		Group B	
Hex	D4	D3	D1	D0	Port A	Port C Upper	Port B	Port C Lower
#80	0	0	0	0	OUTPUT	OUTPUT	OUTPUT	OUTPUT
#81	0	0	0	1	OUTPUT	OUTPUT	OUTPUT	INPUT
#82	0	0	1	0	OUTPUT	OUTPUT	INPUT	OUTPUT
#83	0	0	1	1	OUTPUT	OUTPUT	INPUT	INPUT
#88	0	1	0	0	OUTPUT	INPUT	OUTPUT	OUTPUT
#89	0	1	0	1	OUTPUT	INPUT	OUTPUT	INPUT
#8A	0	1	1	0	OUTPUT	INPUT	INPUT	OUTPUT
#8B	0	1	1	1	OUTPUT	INPUT	INPUT	INPUT

	A		B		Group A		Group B	
Hex	D4	D3	D1	D0	Port A	Port C Upper	Port B	Port C Lower
#90	1	0	0	0	INPUT	OUTPUT	OUTPUT	OUTPUT
#91	1	0	0	1	INPUT	OUTPUT	OUTPUT	INPUT
#92	1	0	1	0	INPUT	OUTPUT	INPUT	OUTPUT
#93	1	0	1	1	INPUT	OUTPUT	INPUT	INPUT
#98	1	1	0	0	INPUT	INPUT	OUTPUT	OUTPUT
#99	1	1	0	1	INPUT	INPUT	OUTPUT	INPUT
#9A	1	1	1	0	INPUT	INPUT	INPUT	OUTPUT
#9B	1	1	1	1	INPUT	INPUT	INPUT	INPUT

The **normal SEGA** is setup in Mode 0 with control word #92 (the disk drive uses control word #90 - more on this later). So I am only going to discuss Mode 0 for now.

DC (A) Input, Side A of keyboard
DD (B) Input
Bits 0-3, Side B of keyboard
Bit 4, External B11 on connector (No connection)
Bit 5, Fault signal SP-400
Bit 6, Busy signal SP-400
Bit 7, Cassette tape input
DE (C lower and upper) Output
Bits 0-2, Keyboard setup
Bit 3, No connection
Bit 4, Cassette tape output
Bit 5, Data bit SP-400
Bit 6, Reset signal SP-400
Bit 7, Feed signal SP-400

BIT Set / Reset FORMAT



An example of the Bit Set/Reset feature can be found in the tape routines. When the tape output is to be zero we use

```
LD A,8:OUT (#DF),A.
```

When the tape output is to be one we use

```
LD A,9;OUT (#DF),A.
```

Previously a way of disabling break (and the whole keyboard) was to use OUT &HDF,&H9B in Basic. We can now see that this changed everything to Input, which does in fact disable the keyboard circuitry.

Sprite Collision - Using Our Knowledge of the VDP

This routine checks for collision between sprites and any dot on the graphics screen. - it only works for MAG 0 sprites

```
                  ORG #EC00
F5F3      Check   PUSH AF:DI           ; Keep sprite number
DBBFF1            DI:IN A,(#BF):POP AF ; Restore sprite number
FE20D0            CP 32:RET NC         ; Return if not 0-31 (invalid)
8787              ADD A,A:ADD A,A      ; Multiply by 4
D3BF              OUT (#BF),A
3E3BD3BF          LD A,#3B:OUT (#BF),A ; Set up for read from Sprite Attribute Table
0000              NOP:NOP
0000              NOP:NOP
DBBE              IN A,(#BE)
3C57              INC A:LD D,A         ; Put Y-coordinate + 1 into D
00                NOP:NOP
DBBE5F            IN A,(#BE):LD E,A    ; Put X-coordinate into E
000000            NOP:NOP:NOP
DBBE47            IN A,(#BE):LD B,A    ; Put pattern name into B
000000            NOP:NOP:NOP
DBBE4F            IN A,(#BE):LD C,A    ; Put EC bit and colour into C
7AFEC0            LD A,D:CP 192        ; Check vertical is between -7 and 191
3804              JR C,Check1
FEF9              CP -7
3FD0              CCF:RET NC           ; Return if sprite not on screen
CB792808  Check1  BIT 7,C:JR Z,Check2  ; Check EC bit
7B                LD A,E               ; If EC set, shift to left 32 pixels
D6205F            SUB 32:LD E,A
FEF9              CP 249
3FD0              CCF:RET NC           ; Return if not on screen
682600    Check2  LD L,B:LD H,0        ; Calculate address of Sprite Pattern
2929              ADD HL,HL:ADD HL,HL
29                ADD HL,HL            ; Multiply by 8
7DD3BF            LD A,L:OUT (#BF),A
7CC618            LD A,H:ADD A,#18
D3BF              OUT (#BF),A          ; Set for read from Sprite Pattern
7BE607            LD A,E:AND 7         ; Number of bits to shift pattern right
3259EC            LD (JUMP),A          ; Self modifying code for the JR below
D5                PUSH DE              ; Keep coordinates safe for later
11E0EC            LD DE,STORE          ; Area to store shifted data
0608              LD B,8               ; 8 scans of data
DBBE      Check3  IN A,(#BE)           ; Get byte of Sprite Pattern
6F                LD L,A               ; HL is used as a shift register
2600              LD H,0               ; from 8 bits to 16 bits
18FE              JR $:JUMP EQU $-1    ; Jump over unnecessary shifts
2929              ADD HL,HL:ADD HL,HL  ; Each add is a left shift
2929              ADD HL,HL:ADD HL,HL
2929              ADD HL,HL:ADD HL,HL
2929              ADD HL,HL:ADD HL,HL
EB                EX DE,HL
7223              LD (HL),D:INC HL     ; Store the shifted pattern
```

```
7323                LD (HL),E:INC HL       ; Which is now 16 bits
EB                  EX DE,HL
10EB                DJNZ Check3            ; Repeat for each scan
D1                  POP DE                 ; Restore coordinates
210008              LD HL,#800             ; Initially 8 scans and 0 skiped bytes
7A                  LD A,D                 ; Get Y-coordinate
D6B93815            SUB 185:JR C,Check4    ; Jump if completely on screen (vert)
3CED44              INC A:NEG
E60767              AND 7:LD H,A           ; We now check 8-(Y-184)MOD8 scans
7AD6F9              LD A,D:SUB 249
380A                JR C,Check4            ; Jump if sprite is moving off bottom
3C                  INC A                  ; Sprite is moving onto top of screen
67                  LD H,A                 ; We now check (Y-248) scans
1600                LD D,0                 ; Set Y coordinate to top of screen
ED44E607            NEG:AND 7              ; We must skip some data stored in STORE
876F                ADD A,A:LD L,A         ; We will skip 2*[8-(Y-248)MOD8] bytes
E5          Check4  PUSH HL                ; Keep this for later
CDCFEC              CALL ADDR              ; Calculate address in VRAM
E3                  EX (SP),HL             ; Swap address wih num of scans and skips
44                  LD B,H                 ; B will be the scan counter
7BFEF9              LD A,E:CP 249          ; Is the sprite fully on screen (horiz)
3806                JR C,Check5            ; Jump if it is - otherwise set some flags
CB79                BIT 7,C                ; Allow for the EC bit
2002                JR NZ,Check5           ; If set, moving on from left, so skip left side
CBF1                SET 6,C                ; Moving off right of screen, so skip right side
2600        Check5  LD H,0                 ; L holds the number of bytes to skip
11E0EC              LD DE,STORE
19                  ADD HL,DE              ; Skip the bytes
D1                  POP DE                 ; Restore address in VRAM
CB79        Check6  BIT 7,C
200B                JR NZ,Check7           ; Jump if we skip left side (in broder)
7BD3BF              LD A,E:OUT (#BF),A
7AD3BF              LD A,D:OUT (#BF),A     ; Set for read from graphics screen
0000                NOP:NOP:NOP
DBBE                IN A,(#BE)             ; Read byte from VRAM
A6                  AND (HL)               ; Use shifted sprite pattern as a mask
37                  SCF                    ; Signal collision
C0                  RET NZ                 ; If any bit set then there is a collision
23          Check7  INC HL                 ; Move to next byte in STORE
CB71                BIT 6,C
200D                JR NZ,Check8           ; Jump if we skip right side (in border)
7BC608              LD A,E:ADD A,8
D3BF                OUT (#BF),A
7AD3BF              LD A,D:OUT (#BF),A     ; Set for read from next scan on the right
000000              NOP:NOP:NOP
DBBE                IN A,(#BE)             ; Read byte from VRAM
A6                  AND (HL)               ; Use shifted sprite pattern as a mask
37                  SCF                    ; Signal collision
C0                  RET NZ                 ; If any bit set then there is a collision
23          Check8  INC HL                 ; Move to next byte in STORE
1C                  INC E                  ; Move down a scan
7BE607              LD A,E:AND 7
```

```
2005                JR NZ,Check9        ; Jump if still in same row
7B                  LD A,E
D6085F              SUB 8:LD E,A        ; Restore to start of row
14                  INC D               ; Move to next row
10D1       Check9   DJNZ Check6         ; Repeat for required scans
A7C9                AND A:RET           ; No collision
                                        ;
62         ADDR     LD H,D              ; Copy Y-coordinate to H
CB3CCB3C            SRL H:SRL H
CB3C                SRL H               ; H=INT(H/8)
7A                  LD A,D              ; This is Y-coordinate
E6076F              AND 7:LD L,A        ; L=YMOD8
7B                  LD A,E              ; This is X-coordinate
E6F8                AND 248
B56F                OR L:LD L,A         ; L = L+INT(X/8)*8
C9                  RET
00         STORE    DS 16,0             ; Define space for 16 bytes of storage
                                        ;
3E00       BASIC    LD A,(STORE)        ; Get which sprite to check
CD00EC              CALL Check          ; Call check routine
9F                  SBC A,A             ; If CARRY = 0 then 0, CARRY = 1 then 255
32E0EC              LD (STORE+1),A      ; Store the result for BASIC program
C9                  RET
```

BOO BOO's

There were problems with the scroll routines from the last issue. There were numbers in some cases, where there should have been question marks. Each call instruction was supposed to be calculated by you and the numbers added in. The calls should have been read as

```
CD????    CALL Up1
CD????    CALL Down1
CD????    CALL Left1
CD????    CALL Right1
```

This is because the addresses in the call instructions were all wrong. However most people managed to figure it out eventually.

IN THE LAST ISSUE

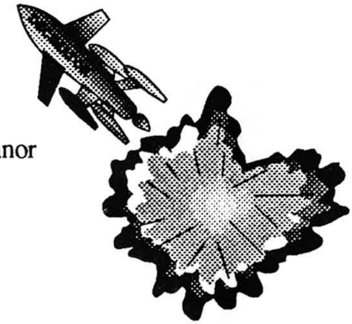
Information from the Disk Drive, about
RS-232, centronics and Floppy Disk Controllers
ie. The stuff I couldn't fit in this issue

Due out about November - December (after Finals)

Poseidon Software

NZ SEGA DISTRIBUTORS

FREEPOST 243
PO BOX 277
TOKOROA
NEW ZEALAND



Decimator	*32K only*	$27.95
Poker	*32K only*	$24.95
Delta Fighter	*32K only*	$24.95
Burgular Bill	*16K / 32K*	$17.95
Carverns of Karanor	*16K / 32K*	$19.00
Vortex Blaster	*32K only*	$12.00
Aerobat	*32K only*	$19.95
Orb of Power	*32K only*	$12.00
Castle of Fear	*32K only*	$12.00
Castaway	*32K only*	$12.00
Burgular Bill and Caverns of Karanor	*on one disk*	$50.00
Michael Howard's - More than 50 Programs		$5.00
Book and Tape - Teach Yourself BASIC Programming		$6.00
LSV, Print 64 and Pattern Paint	*Disk / Tape versions*	$15.00
Magazine programs on cassette		$20.00
Machine Code Summary Sheets		$1.00

	Australia	**New Zealand**	
Magazine subscription (6 issues)	A$26	NZ$25	GST incl

Name ______________________ **Phone** __________

Address ______________________________

____

Item	**No. of items**	**$ Total**

Add Postage **$2.50**

Payment by (circle one) **Total Enclosed $**__________

Cheque Cash Bankcard Visa

Orders over $20.00 only

Credit Card No ______________________

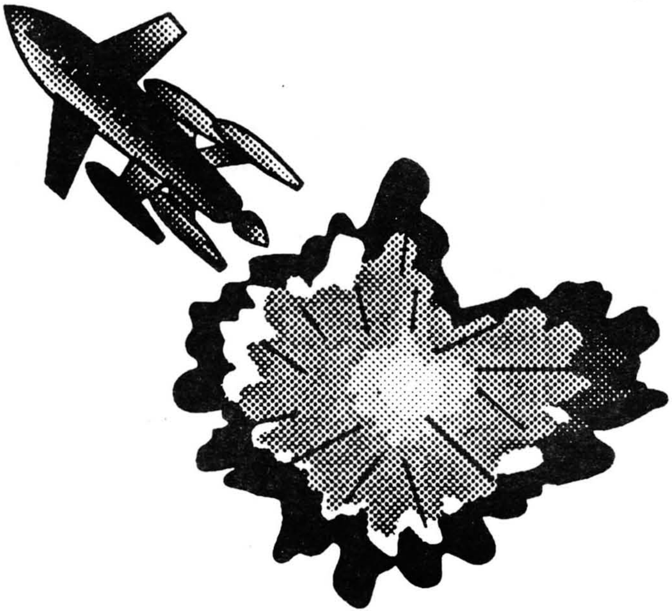
Signed ________ ________ Expires / /

VISA

BANKCARD MasterCard Welcome here

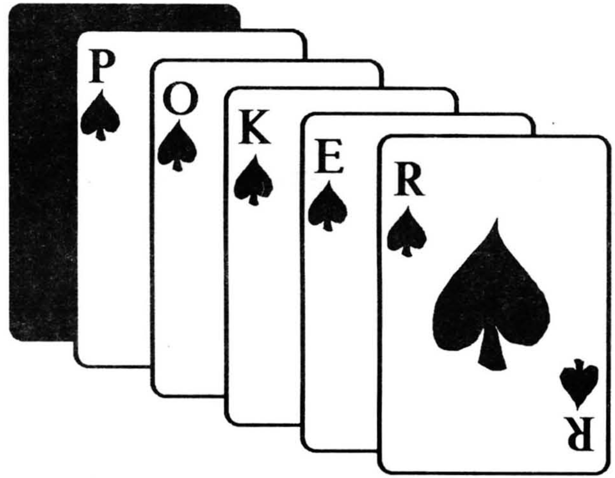
DECIMATOR

32K only



Written by
Michael Boyd
For Poseidon Software

POKER



Written by T. R. Speirs
for Poseidon Software