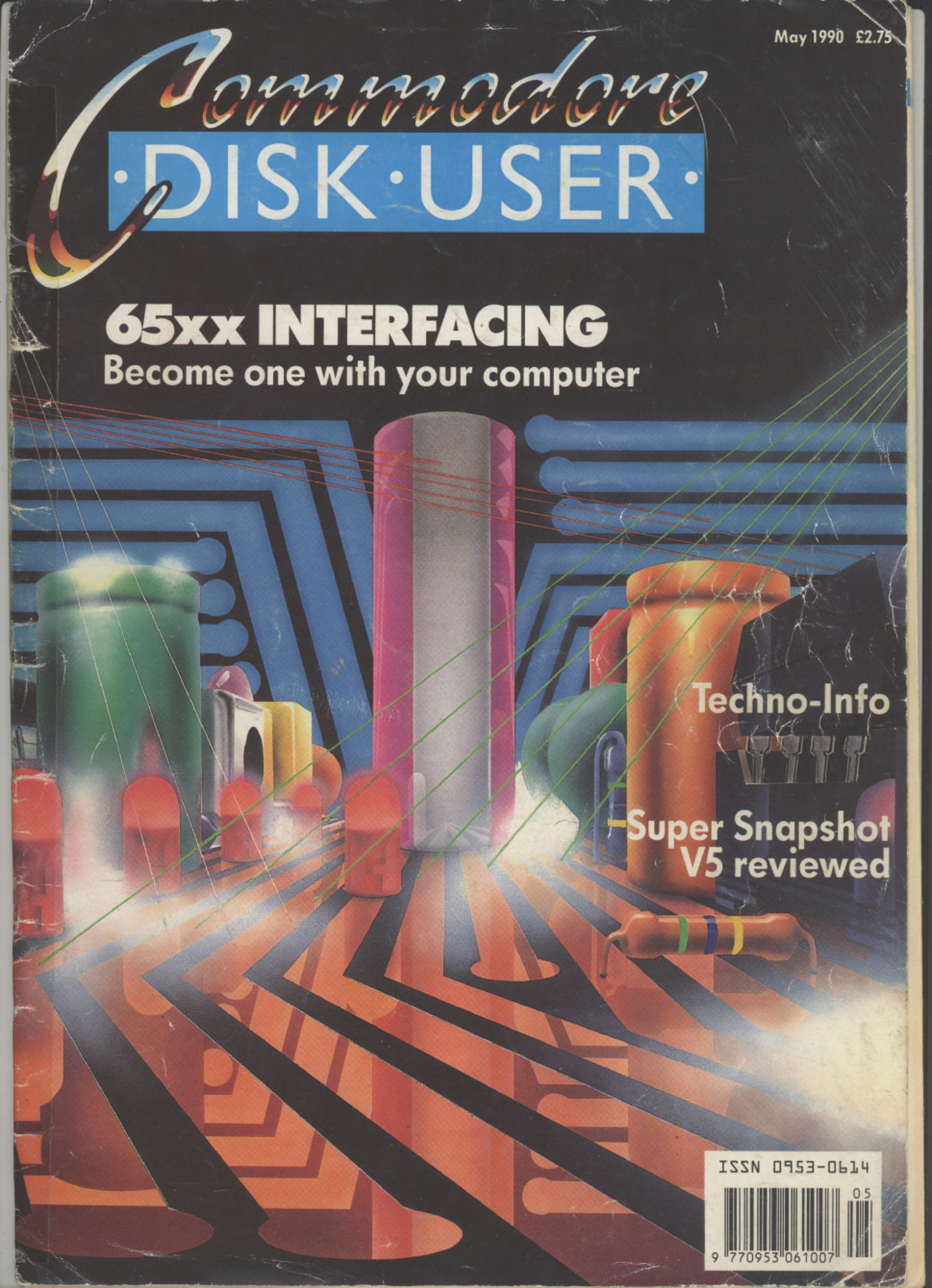
May 1990 £2.75

Commodore DISK USER

65xx INTERFACING
Become one with your computer



Techno-Info

Super Snapshot V5 reviewed

ISSN 0953-0614
05
9 770953 061007

THE ULTIMATE UTILITY CARTRIDGE COMES OF AGE!

NOW ACTION REPLAY Mk VI IS HERE

FOR CBM64/128

THE MOST POWERFUL, FRIENDLY AND FEATURE PACKED UTILITY CARTRIDGE EVER CONCEIVED!

TURBO LOADER — Load 202 block program in under six seconds - world's fastest disk serial loader. On-board Ram and Rom achieves such high loading speeds. Works with 1541/1571/Oceanic/1581.	**TAPE TURBO** — This feature will add Turbo Reload to the programs that you save to tape - no user knowledge required.
INFINITE LIVES GENERATOR — Automatic infinite lives!!! Very easy to use, works with many programs. No user knowledge required.	**FAST FORMAT** — Format an entire disk in about 10 seconds - no more messing about.
PROF MACHINE CDE MNTR — Full 64K Freezer Monitor - examine ALL memory, including stack, I/O area and registers in their frozen state. Ideal for debugging or just for fun!	**PRINTER DUMP** — Print out your frozen screen to printer - MPS 801, 803, Epson, Star, etc. - very versatile.
SPRITE CONTROL — Freeze the action and view the sprites - watch the animation - customize your games - kill sprite collisions.	**CENTRONICS INTERFACE** — For parallel printers, Star, Epson, etc. Print out listings with graphic characters etc. (Cable required for parallel port £12.99).
FREEZER FACILITY — Now you can make your old slow loading programs load faster. Simply freeze the action and save to tape or disk to reload at superfast speed - no more waiting for programs to load.	**SCREEN EDITOR** — Now you can edit the entire frozen screen with this text editor - change names on high scores, etc. Great fun!
DISK COPY — Easy to use disk/file copier. Much faster than conventional methods. Ideal for backing up data disks.	**EXTENDED TOOLKIT** — Many single stroke commands for Load, Save, Dir, etc. Plus range of extra commands, i.e. Auto Number, Old, Delete, Merge, Append, Linesave, etc.

THE REVIEWERS SAID

"I'm stunned, amazed and totally impressed. This is easily the best value for money cartridge. *The Cartridge King!*"
Commodore Disk User

THE ACTION REPLAY MK VI IS NOT ONLY THE WORLDS FASTEST TURBO LOADING CARTRIDGE BUT IT IS PACKED WITH ALL THE FEATURES YOU HAVE EVER NEEDED AND THEN SOME MORE!!!

HOW TO ORDER...

ALL ORDERS DESPATCHED WITHIN 48 HRS

PHONE
0782 744707
24hr Credit Card Line

POST
Send cheques/POs made payable to "Datel Electronics"

FAX
0782 744292
UK ORDERS POST FREE
EUROPE ADD £1
OVERSEAS ADD £3

PRICES AND SPECIFICATIONS CORRECT AT TIME OF PRESS AND SUBJECT TO CHANGE WITHOUT NOTICE

DATEL ELECTRONICS

DATEL ELECTRONICS LTD.,
FENTON INDUSTRIAL ESTATE
GOVAN ROAD, FENTON,
STOKE-ON-TRENT,
ENGLAND.

TECHNICAL PHONE LINE
0782 744324

GRAPHICS SUPPORT UTILITIES DISK

SLIDE SHOW. View your favourite screens in a slide show type display.
BLOW UP. Unique utility allows you to take any part of of a picture & 'blow it up' to full screen.
SPRITE EDITOR. A complete sprite editor helps you to create or edit sprites.
MESSAGE MAKER. Any screen captured with Action Replay or created with a graphics package can be turned into a scrolling screen message with music.

ONLY £9.99

CONTENTS

IN THE MAGAZINE

ON THE DISK

Commodore Disk User
Volume 3 Number 7
May 1990

AN Argus SPECIALIST PUBLICATION

Editor: PAUL EVES
Group Editor: STUART COOKE
Production Editor: HILARY CURTIS
Photography: MANNY CEFAI
Adventure Correspondent: GORDON HAMLETT
Advertisement Manager: PAUL KAVANAGH
Display Sales Exec: MARIA WADE
Classified Sales Exec: TONY FLANAGAN
Designer: MARK NEWTON
Origination: EBONY TYPESETTING
Distribution: S. M. DISTRIBUTION
Printed By: CHASE WEB LTD, ST IVES PLC

SUBSCRIPTION RATES

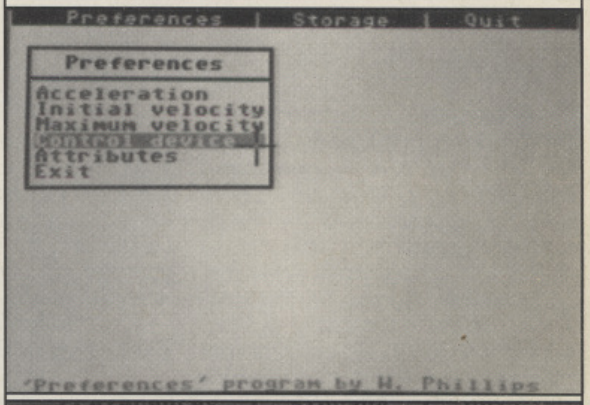
Here are the rates for subscriptions to CDU with effect from November 1989

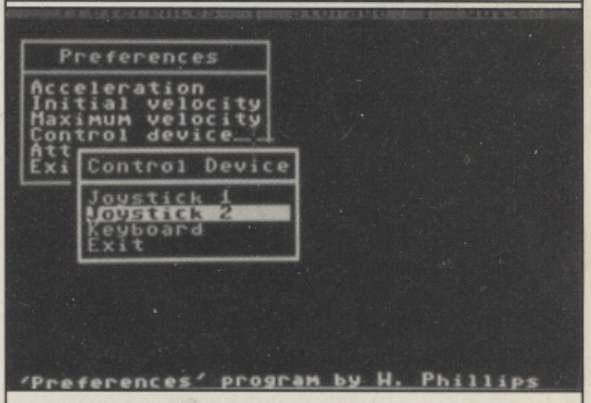
UK	**£33.00**
Europe	**£39.00**
Middle East	**£39.30**
Far East	**£41.60**
Rest of the World £39.70 or	**$69.00**

Airmail rates on request

Commodore Disk User is a monthly magazine published on the 3rd Friday of every month. **Argus Specialist Publications, Argus House, Boundary Way, Hemel Hempstead, HP2 7ST. Telephone: (0442) 66551 Fax: (0442) 66998.**

65XX INTERFACING





Above are 2 examples of interrupt pointers

Editor's Comment

Hopefully, if everything has gone to plan, this issue should be the first of the new style. I hope you like it. We have tried to make the appearance of the magazine more appealing, yet still retain its identity as a serious users magazine.

Since finishing the 'TRIVIA CHALLENGE' series, I have had a small number of queries. It appears that some of you are not managing to save a working version of the game, even after following the instructions correctly. If you are one of these people then please read this before sending in your disks/letters.

If you have a system that has everything bar the kitchen sink hooked up, disconnect EVERYTHING except your disk drive and monitor/tv screen. The program does not like Cartridges, Interfaces, Printers etc etc on-line during its installation process.

The author of the program and myself have in fact found a small bug in the program. SOMETIMES, not very often though, the program produces garbage amongst the questions, or on the categories line. He has tried for 8 months to track this problem down but has had no success. However, because it only manifests itself so seldom, we feel that it is something you can live with.

To save anyone that is having trouble getting an up and running version tearing their hair out, if you are one of these then send me a blank disk to the editorial office, and I will save a copy out for you.

That just leaves me to say, enjoy this months mag. Hope the programs are to your liking. Don't forget, if you have any comments, ideas, suggestions (Keep them clean) or just simply want to air your views, pick up your pen and write to me.

Disk Instructions

We do our best to make sure that CDU will be compatible with all versions of the C64 and C128 computers. One point we must make clear is that the use of 'Fast Loaders', 'Cartridges' or alternative operating systems (Dolphin DOS) may not guarantee that your disk will function properly. If you use one or more of the above and you have difficulties, then I suggest you disable them and use the computer under normal, standard conditions.

Getting the programs up and running should not present you with any difficulties, simply put your disk in the drive and enter the command.

Load "MENU",8,1

Once the disk menu has loaded you will be able to start any of the programs simply by pressing the letter that is to the left of the desired program.

It is possible for some programs to alter the computer's memory so that you will not be able to LOAD programs from the menu correctly until you reset the machine. We therefore suggest that you turn your computer off and then on before loading each program.

How to copy CDU files

You are welcome to make as many of your own copies of CDU programs as you want, as long as you do not pass them on to other people, or worse, sell them for profit. For people who want to make legitimate copies, we have provided a simple machine code file copier. To use it, simply select the item FILE COPIER from the main menu. Instructions are presented on screen.

Disk Failure

If for any reason the disk with your copy of CDU will not work on your system then please carefully re-read the operating instructions in the magazine. If you still experience problems then:

1) If you are a subscriber, return it to:
   Select Subscriptions Ltd
   5, River Park Estate
   Berkhamsted
   HERTS. HP4 1HL
   Tele: 0442-876661

2) If you bought it from a newsagents, then return it to:
   CDU Replacements
   Protoscan
   Burrell Road
   St. Ives
   Cambs
   P17 4LE
   Tele: 0480-495520

(Within eight weeks of publication date disks are replaced free).

After eight weeks a replacement disk can be supplied from Protoscan for a service charge of £1.00. Return the faulty disk with a cheque or postal order made out to Protoscan and clearly state the issue of CDU that you require. No documentation will be provided.

Please use appropriate packaging, cardboard stiffener at least, when returning disk. Do not send back your magazine, only the disk please.

NOTE: Do not send your disks back to the above if its a program that does not appear to work. Only if the DISK is faulty. Program faults should be sent to the editorial office marked FAO bug-finders. Thank you.

Back Issues

Back issues of CDU are available at £3.25 per issue, which includes postage and packing via:

Select Subscriptions Ltd
5, River Park Estate
Berkhamsted
Herts
HP4 1HL
0442-876661

VOL 2 No. 4 MAY/JUN 89
BASE ED – Get organised with this C64 database.
DBASE 128 – 40 or 80 column storage for C128 owners.
6510+ – The ultimate in C64 assembly programs.
SID SEQUENCER – Make Commodore music with ease.
LIBERTE – Escape the POW camp in this 1940's style adventure.
FX KIT – Bangs, Pows and Zaps made easy.

VOL 2 No.5 JUL/AUG 89
FONT FACTORY – Create your own chars.
HI-RES DEMO KIT – Add music to your favourite picture.
ANIMATOR – Get those sprites moving.
BORDER MESSAGE SCROLL – Say what you want along the bottom of the screen.
TYPIT 128 – Create professional text layouts on your C128.
SCREEN COPIES UTILITY – Download your favourite screens.
VIDI-BASIC – Graphic based extension to Basic.
64 NEWS DESK – Become a C64 reporter.

VOL 2 No.6 SEP/OCT 89
MICKMON – An extensive M/C monitor.
SCRAPBOOK – Collectors and hobbyists database.
CELLRATOR – Enter the caves if you dare.
RAINBOW CHASER – Rainbows means points in the unusual game.
HIDDEN GRAPHICS – Utilitise those graphic secrets.
FORTRESS – Save the world. Yet again!!
DISK HUNTER – Keep tabs on your disk library.
SUPERFILE – One more for the record keepers.

VOL 3 No.1 NOVEMBER 89
BASIC EXTENSION – Windows and Icons the easy way
B-RAID – Vertical scrolling shoot 'em up
DISKONOMISER – Prudent disk block saving
HELP – Design your own information help screens
ORSITAL – An arcade style game with a difference
PROGRAM COMPARE – Modifying Basic progs has never been easier
RASTER ROUTINES – A few colourful demos
SPRITE EDITOR 1 – A no nonsense basic sprite editor
WABBIT – Help the rabbit collect his carrots

VOL 3 No.3 JANUARY 90
4 IN A ROW – Connect a row of counters
FROGS IN SPACE – Leap to safety across the space lanes
BLACKJACK – Don't lose your shirt
LORD OF DARKNESS – Defeat the evil lord true adventure style
MARGO – Fly around and collect the jewels
JETRACE 2000 – Have you got what it takes to be best
ULTIMATE FONT EDITOR – Create your own screens and layouts
SELECTIVE COLOUR RESTORE – Design your own start up colours
6510+ UNASSEMBLER – Transform M/C into Source, with labels
TRIVIA CHALLENGE – The first 4 of 12 fonts for Geos users

VOL 3 No.4 FEBRUARY 90
COLOUR PICTURE PRINT – Download your favourite colour screens
BASE-ED 2 – An update to our popular database system
1ST MILLION – Play the market in this strategy game
FM-DOS – Enhance your drives operating system
GEOS FONTS – A further 4 fonts for Geos users
HASHING IT – Relative filing made easy
MULTI-SPRITE – Make full use of up to 24 sprites
DIRECTORIES EXPLAINED – Find your way through the directory jungle
TRIVIA CHALLENGE – The second part of this popular game

VOL 3 No.5 MARCH 90
PLAGUE – Become your planets Guardian and Defender
SURROUND – Reversi on the C64
GEOS FONTS – The last of 12 new fonts
SCREEN SLIDE – Create your own slideshows
JOYSTICK TESTER – Put your stick(s) through the mill
COLOUR MATCHER – Mastermind for the younger ones
SCREEN MANIPULATOR – Full use of the screen now obtainable
TRIVIA CHALLENGE – The last of the 3 files for the game
VIDEO RECORDER PLANNER – Keep tab on your recordings

VOL 3 No.6 APRIL 90
BAR PROMPTS – M/C alternative input routine
HI-LITE BARS – as Bar Prompts but in Basic
TEXAS DEMO – Examples of using Basic for demos
CHARS TO SPRITES – Convert UDG's to sprites
FONT FACTORY – Compliments Chars to Sprites
3D-TEXT MACHINE – Impressive 3D text screens made easy
SCREEN ENHANCER – Makes full use of the screen easy to do
SPREADSHEET 64 – An excellent, easy to use spreadsheet
MINI-AID – 3 short utilities to aid the Basic programmer
C128 COLLECTION – 3 useful C128 programs

All orders should be sent to: Select Subscriptions Ltd, 5, RIVER PARK ESTATE, BILLETT LANE, BERKHAMSTED, HERTS, HP4 1HL Please allow 28 days for delivery.

Interrupt Pointers & Menus

Let your C64 show the 16 bit machines that they do not have the monopoly on Wimps

By William Phillips

If you have ever written a Basic program, you will have noticed that even with a superbly structured program that flows like water, Basic just doesn't offer enough speed for user-friendly applications. The days of long and cumbersome 'Do you want instructions?' programs have long disappeared; we are equipped with a means of controlling a program which everyone can pick up easily and varies little from program to program. It is called Wimps (Windows, Icons, Menu's and Pointers).

So, we know what our programs need; a more user-friendly environment. Yet, in Basic, moving pointers would be very slow indeed, and not all that smooth either. Obviously, a little machine code is necessary, and that, my friends, is why you are reading this article!

The machine code program that I have written gives a Basic programmer control of a moving pointer and menu bar system. This means that you can move a pointer around the screen using a joystick or the keyboard while the program is running, and also tell the program to inverse part of a screen line if the pointer is over it. This is useful for pull down menu's, as you will see in the pointers program. This program is in Basic and was written to illustrate the use of the pointer code. If you play around with the pointers program, it will probably help you to understand the pointer code program. To load it, select Pointers from the menu.

After a small delay, you will be presented with a black screen with a pointer somewhere in the middle, and the top line showing the options available. You will notice that this line has a foreground and a background colour on a low resolution screen. This is unusual and is one of the features of the pointer code.

When the program starts up, the device is set to keyboard. To control the pointer use the keys Q, A, O, and P. Just in case you couldn't work it out on your own, they move the pointer up, down, left and right respectively. Depressing two keys at once will make the pointer move diagonally.

When the pointer moves, you should notice that it starts off at a certain velocity, and accelerates up to a maximum velocity. The start of initial velocity and maximum velocity are numbers in terms of pixels per second. The acceleration is a delay. If x = the acceleration rate, then x+1/50ths of a second will pass before the present velocity of the pointer is increased by one up to the maximum velocity. If all this seems a little complicated, don't worry. It isn't all that important, but when incorporating the routines into your own programs, it is certainly useful to understand how the pointer works.

Looking back at the pointers program, you will find three options available to you, namely Preferences, Storage, or Quit.

You can select any of these by moving the pointer so that it points to the desired option, and pressing fire/return depending on which device is being used.

1) Preferences

A menu will appear giving you a whole new set of options. You can change the acceleration rate, the initial and maximum velocities, the control device, the attributes, or leave the preferences menu. Upon moving the pointer over one of the options, you will notice that it inverts and changes colour, or 'lights up'. This is another of the features offered by the pointer code. With just a few pokes, you can set the pointer to 'light up' parts or all of the line the pointer is presently pointing to.

All the menu's are reasonably self explanatory – and that is the beauty of Wimps it only takes five minutes to learn how to use a program perfectly, with the minimum of instructions.

2) Storage

The whole point of the preferences program, apart from a demonstration of the pointer code, is that you can set up your preferred device, velocity and acceleration settings and then save it to tape or disk. Working through this set of menu's is very easy too – just type what you wish to call Pointers program, press T or D for the device you wish to save it to, and the program will save the Pointers program under your filename, followed by the pointer code, saving it as "p.code". If you selected tape, a further window will come up and allow you to position the tape before saving.

3) Quit

As you may have guessed, this takes you out of the program. The screen is cleared, the pointer interrupt turned off, and the program ends.

We have now seen the three main uses of the pointer code; the moving pointer, the bar light up system and the background colour on the top line, and all this runs on interrupts! All we need to look at now are the pokes that influence the pointer code.

Firstly, I will introduce you to the pointer code program. On the disk it is called "p.code" and is located from 49152-50013. Load it by typing: LOAD "P.CODE,8,1

Secondly, before we delve deeply into POKE1and, you should know that you cannot use sprites 0 or 1 in your programs if you are using the pointer code, as they are used to put the pointer on the screen.

1) Starting and stopping the interrupts

SYS 49152 initialises the interrupts. It also copies the sprite definitions for sprite 0 and 1 to locations 832-959 where the VIC II chip can 'see' them. For this reason, you should not poke this area while the pointer code is running. If you wish to change the definition of the pointer, you should set it up elsewhere in memory where the VIC II can 'see' it, and change the

sprite pointers 2040 and 2041. Location 2040 should point to the pointer/icon definition and location 2041 should point to the mask (a border around the pointer/icon so that the pointer/icon definition can always be seen, even when surrounded by pixels that are the same colour as it.

SYS 49274 suspends the interrupt. This should be done when saving or loading. To restart the interrupt, use SYS 49152.

2) Top line colour

This feature sets the top 9 pixel lines of the display to a colour of your choice. The colour corresponds to the normal CBM colour range, and its number should be in the range of 0-15 (see over the page for a copy of the CBM colour range). Type

POKE 499332, colour number

to change the colour of the top line background. If this feature is not needed, you should set the colour to the same as that of the background.

i.e. POKE 49332, PEEK (53281)

3) Pointer System

a) Changing device.
When changing devices, you have to poke two locations.

POKE 49884,x

If x is 0 then the device is joystick.
If x is 1 then the device is keyboard.

If you set this location to 0 or joystick mode, you must also set another location to tell the computer which port to read from.

POKE 49337,x
If x is 0 then the computer reads from port II.
If x is 1 then the computer reads from port I.

b) Changing the location of the pointer on the screen.

If you let x = the x position, in the range of 23 to 347 and y = the y position, in the range of 50 to 249, the following pokes apply to set the co-ordinates of the pointer:

POKE 49872,x-256*INT(x/256)
POKE 49873,INT(x/256)
POKE 49874,y

In other words, 49872 is the lower 8 bits of the 9 bit value for x, location 49873 is the last bit (MSB) of this value, and location 49874 controls the y position of the pointer.

These registers can also be read to determine the present position of the pointer on the screen. These locations give you the pixel position. Another two registers hold the screen x and y position.
(0-39 and 0-24 respectively).

PEEK (49886) gives the x position and
PEEK (49885) gives the y position

All of these registers are updated every time the pointer is moved.

c) Setting up a bar light up parameter block.

You have seen how the inverse bar system works in the preferences program. It is controlled by the block of memory from 679 to 767. To set this up, you should firstly poke the dimensions and co-ordinates of the window into memory:

POKE 680, upper Y co-ordinate of window
POKE 681, lower Y co-ordinate of window
POKE 682, left hand side X co-ordinate
POKE 683, right hand side X co-ordinate

In this window, you may not want certain options to be available for some reason. For instance, if you made a sprite editor program one of the menu's may have two options saying 'Hires' option to be available. You could omit that option from the menu, or more simply, you can set up the parameter block so that the bar doesn't light up when you move the pointer over the option. This block of memory is from 685 to 709 and is the block of BAR ACKNOWLEDGE CODES. Location 685 corresponds to the top line of your window, 686 to the second line and so on. If the location holds a 1 then the bar will light up. If it holds a 0 then, you guessed it, it won't.

Location 684 determines what colour the bar will turn when it is inversed. This number is a CBM colour code (0-15).

Location 679 is used to tell the bar light up system whether or not to inverse any bars that are held in the parameter block. If it holds 0, then the bar system will not inverse any more bars it comes across – in effect, it is turned off. If it holds 1, then it is turned on; all bars that are held in the parameter block are inversed.

You should be careful to ensure that the screen does not scroll when a bar is up, or the screen will look a mess!

One last location that may be useful is location 710. This, when peeked will tell you if a bar is lit up even if location 679 holds a 0. It should be poked with a 0 before the interrupts are turned on (as should location 679).

Well, that's all there is to know about the control locations, and I will leave you with a table of the CBM colour codes, and a summary of the locations that are useful. Good luck!

CBM Colour Codes.

Black	0	Orange	8
White	1	Brown	9
Red	2	Pink	10
Cyan	3	Dk Grey	11
Purple	4	Mid Grey	12
Green	5	Lt Green	13
Blue	6	Lt Blue	14
Yellow	7	Lt Grey	15

Useful Locations.
49152 Initialise Interrupts.
49274 Suspend Interrupts.
49332 Top line background colour.
49337 Joystick Port 0 = Port II 1 = Port I
49872 LSB 8 bits of X co-ordinate of pointer.
49873 MSB bit of X co-ordinate of pointer.
49874 Y co-ordinate of pointer.
49885 Y screen co-ordinate of pointer.
49886 X screen co-ordinate of pointer.
679 Bar light up on/off 0 = off 1 = on.
680 Upper Y co-ordinate of window.
681 Lower Y co-ordinate of window.
682 Left hand side X co-ordinate of window.
683 Right hand side X co-ordinate of window.
684 Colour of bar.
685-709 Bar Acknowledge Codes.
710 Is bar lit up? 0 = No.

Finally, experiment to your hearts content, it is the only way of finding out how things work, and it's great fun!

65XX INTERFACING

Steve Carrie begins the third part of his series which deals with interfacing with the 65xx series of microprocessors

Last time, we saw how easily it was to program the parallel port from BASIC to provide a simple but effective data exchange between two machines (Programs 1 & 2).

Now take a look at Programs 3 and 4. Program 3 is basically an assembly language version of program 1 in that it simply takes a string from the keyboard, adds a return character 9CHR$(13)) and sends it out byte-by-byte. The major difference here is the speed! Both routines are written to assemble at $C000 and should be started from BASIC by an SYS 49152 command.

These programs (and all others in the series) were written using my ASM assembler which was published in the June 1989 issue of Your Commodore. If you are using another assembler, it shouldn't be too difficult to convert the source code format.

Program 4 however is a little more complicated and consists of two parts, the setup/foreground section and the interrupt background section. Once the interrupt has been set up, the foreground portion simply loops round waiting for the background portion to signal that a string has arrived. Of course, the foreground section could be doing a lot more than just this but for the purposes of this demonstration we'll just leave it to loop round. When the DATAREADY flag is set, the main section displays the received string and resets the flag ready for the next communication. You may stop the program at any time by pressing the RUN/STOP key or hitting RUNSTOP/RESTORE.

The background section only operates when a byte arrives at the data port (interrupt generated by FLAG). The bytes are stored in memory until a return character (13) is received wherepon the DATAREADY flag is set indicating to the main section that a string is ready.

```
10;     PROGRAM 3
20      ;
30      ORG $C000
40      ;
50      .CIA EQA $DD00
60      ;
70      .START JSR INIT
80      JSR KEYIN
90      JSR SENDOUT
100     JMP START+3
110     ;
120     .BUFFER RES 80
130     ;
140     .INIT LDA #$FF
150     STA CIA+3
160     RTS
170     ;
180     .KEYIN LDY #0
190     .KYE1 JSR $FFCF
200     STA BUFFER,Y
210     INY
220     CMP #13
230     BNE KEY1
240     JMP $FFD2
250     ;
260     .SENDOUT LDY #0
270     .SEND1 LDA BUFFER,Y
280     STA CIA+1
290     TAX
300     .SEND2 LDA CIA+13
310     AND#$10
320     BEQ SEND2
330     INY
340     CPX #13
350     BNE SNED1
360     RTS
370     ;
```

```
10      ; PROGRAM 4
20      ;
30      ORG $C000
40      ;
50      .CIA EQA  $DD00
60      .DATAREADY EQA $FB
70      .BUFPTR EQA $FC
80      ;
90      JSR INIT
100     .WAITLOOP JSR $FFE1
110      BEQ EXIT
120     BIT DATAREADY
130     BPL WAITLOOP
140     JSR OUTPUT
150     LDA #0
160     STA DATAREADY
170     JMP WAITLOOP
180     .EXIT JSR SHUT
190     RTS
200     ;
210     .BUFFER RES 80
220     ;
230     .INIT SEI
240     LDA #$00
250     STA CIA+3
260     LDA #<INTR
270     LDX #>INTR
280     STA $0318
290     STX $0319
300     LDA #$90
310     STA CIA+13
320     LDA #$00
330     STA BUFPTR
340     STA DATAREADY
350     CLI
360     RTS
370     ;
380     .INTR PHA
390     TXA
```

```
400     PHA
410     TYA
420     PHA
430     LDA CIA+13
440     AND #$10
450     BEQ EXIT1
460     LDY BUFPTR
470     LDA CIA+1
480     STA BUFFER,Y
490     INY
500     STY BUFPTR
510     CMP #13
520     BNE EXIT2
530     LDA #$80
540     STA DATAREADY
550     #$00
560     STA BUFPTR
570     .EXIT2 JMP $FEBC
580     .EXIT1 JMP $FE4C
590     ;
600     .SHUT SEI
610     LDA #$47
620     LDX #$FE
630     STA $0318
640     STX $0319
650     CLI
660     RTS
670     ;
680     .OUTPUT LDY #0
690     .OUT1 LDA BUFFER,Y
700     JSR $FFD2
710     CMP #13
720     BEQ ENDOUT
730     INY
740     BNE OUT1
750     .ENDOUT RTS
760     ;
```

PROGRAM 4 OPERATION

The first thing to note about this program is that we are not using the IRQ interrupt as you might expect. The IRQ output of CIA 2 is connected to the Non-Maskable Interrupt (NMI) line of the processor. This requires a slightly different interrupt procedure from what you might expect. In fact, for our purposes it is actually better than using IRQ since the computer's operating system uses this to perform the normal system interrupt for the keyboard scan, cursor flash, etc. A situation may arise whereby the interface may take a back seat to the system.

Using the NMI interrupt, the processor will respond instantly to the arrival of our data, even if it is currently servicing an IRQ interrupt. The reason for this apparently strange design has a lot to do with the pseudo-RS 232 on these computers. Both RS 232 input and output is done by using NMI interrupts.

The INIT routine sets up the NMI interrupt vector and also enables the CIA FLAG interrupt. The main routine loops around checking the STOP key ($FFF1) and also the DATAREADY flag byte. When DATAREADY is set to $80, the program calls the routine OUTPUT to print the received data.

The interrupt routine INTR first checks the ICR to see if our FLAG interrupt has occurred. We do this because the RUNSTOP/RESTORE keystroke also uses the NMI subsystem. If our FLAG interrupt has not occurred, we jump to the (almost) normal system routine at $FE4C. When FLAG does occur, we load the data from PRB and store it in the buffer. We store the buffer pointer and check for a return character (13) which signifies the end of the string. If true then we set DATAREADY to $80 to tell the foreground program that the complete data string is ready.

As a small exercise, you could try rewriting program 3 to operate with a NMI interrupt. It would accept a string as before but would send it out under interrupt. Your main program will have to signal TO the NMI routine that data is ready to go out.

It isn't as difficult as it may appear and when I go on to look at the 6526 serial port and the Plus 4's 6529 port, I'll present routines to do just this, except that you'll have to convert it for the 6526 parallel port!

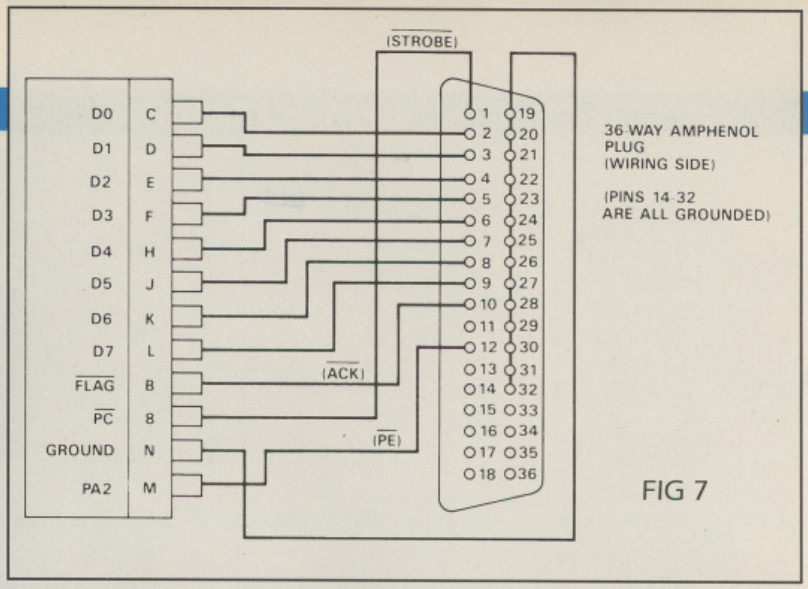


FIG 7

What now?

You may be wondering where all this is leading. Well, one of the annoying things about the Commodore 64 and 128 is their inability to make use of parallel printers directly. There are many inexpensive printers on the market which interface via a Centronics Parallel Interface. Using a method similar to that which I have described above, and with a little clever programming, we can use one of these printers on our machines.

Program 5 does just this for the C64, and Fig 7 shows the necessary wiring from the computer to the printer. The idea here is basically the same as before except that the second computer is replaced by a printer. Once loaded (or assembled) into memory, it may be installed by issuing a SYS 49152 command. Relocate it elsewhere if you need to.

Program 5. Parallel printer driver

```
10      ORG $C000
20      ;
30      .CIA EQA $DD00
40      ;
50      .START SEI
60      LDA #<NEWOPEN
70      LDX #>NEWOPEN
80      STA $031A
90      STX $031B
100     LDA #<NEWCLOSE
110     LDX #>NEWCLOSE
120     STA $031C
130     STX $031D
140     LDA #<NEWCHKOUT
150     LDX #>NEWCHKOUT
160     STA $0320
170     STX $0321
180     LDA #<NEWCHROUT
190     LDX #>NEWCHROUT
```

```
200   STA $0326
210   STX $0327
220   LDA #<NEWCLRCHN
230   LDX #>NEWCLRCHN
240   STA $0322
250   STX $0323
260   CLI
270   RTS
280   ;
290   .RTLF BYT 0
300   ;
310   .NEWOPEN LDX $B8
320   BNE 3
330   JMP $F70A
340   JSR $F30F
350   BNE 3
360   JMP $F6FE
370   LDX $98
380   CPX #$0A
390   BCC 3
400   JMP $F6FB
4120  INC $98
420   LDA $B8
430   STA $0259,X
440   LDA $B9
450   ORA #$60
460   STA $026D,X
470   LDA 4BA
480   STA $0263,X
490   BNE 3
500   JMP $F3FD
510   CMP #$05
520   BNE NOT1
530   LDA #$7F
540   STA CIA+13
550   LDA #$00
560   STA CIA+14
570   STA CIA+15
580   LDA $B9
590   LDX #$00
600   CMP #10
610    BNE 2
620   LDX #$80
630   STX RTLF
640   CLC
650   RTS
660   .NOT1 JMP $F379
670   ;
680   .NEWCLOSE JSR $F314
690   BEQ 2
700   CLC
710   RTS
720   JSR $F31F
730   TXA
740   PHA
750   LDA $BA
760   BNE 3
770   JMP $F2F1
780   CMP #$05
790   BNE 3
800   JMP $F2F1
810   JMP $F2A1
820   .NEWCHKOUT JSR $F30F
840   BEQ 3
850   JMP $F701
860   JSR $F31F
870   LDA $BA
880   BNE 3
890   JMP $F70D
900   CMP #$05
910   BNE NOT2
920   STA $9A
930   PHA
940   LDA #255
950   STA CIA+3
960   LDA CIA+2
970   AND #$FB
980   STA CIA+2
990   PLA
1000  CLC
1010  RTS
1020  .NOT2 JMP $F262
1030  ;
1040  .NEWCHROUT PHA
1050  LDA $9A
1060  CMP #$05
1070  BEQ 3
1080  JMP $F1CD
1090  LDA CIA
1100  AND #$04
1100  BNE BAD1
1120  PLA
1130  JSR SENDCENT
1140  CMP #13
110   BNE NOLF
1160  BIT RTLF
1170  BPL NOLF
1180  PHA
1190  LDA #10
1200  JSR SENDCENT
1210  PLA
1220  .NOLF CLI
1230  CLC
1240  RTS
1250  .BAD1 PLA
1260  JMP $F707
1270  ;
1280  .NEWCLRCHN LDX #$05
1290  CPX $9A
1300  BEQ 3
1310  JMP $F333
1320  LDX #$03
1330  STX $9A
1340  RTS
1350  ;
1360  .SENDCENT PHA
1370  SEI
1380  STA CIA+1
1390  .LP1 LDA CIA+13
1400  AND #$10
1410  BEQ LP1
1420  PLA
1430  RTS
1440  ;
```

Program 5 Operation

What I have done here is to intercept 5 of the system routines concerned with I/O. These are OPEN, CLOSE, CHKOUT, CHROUT and CLRCHN. By alterating the appropriate vectors I was able to force the operating system to recognise device number 5 as a parallel printer. This device number is normally a serial printer or plotter and the routines simply intercept this number. The routines, for the most part, duplicate their kernal equivalents until a suitable cut-off point is found.

You may OPEN a channel in BASIC as follows;

OPEN <logical file #>, 5, <secondary address>

where logical file # is as per a normal printer open statement and secondary address may be 10 (enable line feed send) or anything else for no line feeds. You may list a program as per normal; e.g.

OPEN 1, 5, 10: CMD 1: LIST

The beauty of this system is that if your parallel printer goes offline, the routine will stop sending data and will wait until the printer comes online again. Most parallel printers have data buffers of some kind so there is a possibility that the computer will return to your control long before the listing stops!

You may also notice from the wiring diagram that I have connected an extra line to pin 12 of the Amphenol plug from pin M on the userport. This is line 2 from port A which is otherwise unused. I've used this as a paper-end detector in the software and it will cause a Device Not Present error if it is high.

The routine lives at $C000 and should be transparent to most of your programs. I can't speak for commercial software of course but it should be OK for your own use. Normal serial printing may be done using device number 4.

I hope you find it useful.

Next time we will pick up the thread of our investigation into the facilities of the 6526 CIA.

Oops-a-Daisy

One of the joys of being an Editor is taking all the flak when you make a whopping booboo. When you make several the flak gets really heavy.

Casting his critical eye over the FEBRUARY issue of the magazine he has found faults in **MULTI-SPRITE** (Pages 26-29), **DIRECTORIES EXPLAINED** (Pages 30-32) and **TECHNO-INFO** (Pages 39-40). In the MARCH issue we have found a fault with **SCREEN SLIDE**.

February Issue

MULTI-SPRITE

Page 27, third column, line 14 should read (POKE16383,0)
Page 27, third column, line 34 should start with "SI2"
Page 29, first column, line 11 should start with "DIV"
Page 29, third column, line 15 should start with PRT, MUL, CL1
Page 29, third column, line 32 should read The program "M-S Technical"

DIRECTORIES EXPLAINED

Page 31, first column, line 14 should read OPEN15, 8, 15: OPEN8, 8, 8, "#"
Page 31, first column, line 39 should read GET#8, A$: IFA$ = "" THEN
Page 31, second column, line 5 should start N=ASC (A$)
Page 31, second column, line 11 should be PRINT#8, A$
Page 32, second column, line 20 should read PRINT#15, "B-P: 8"; FN* 32+B
Where 'FN' represents the file number (0-7, remember there are 8 entries per sector) and 'B' is the value listed in the 'Byte by Byte' section of this article.

TECHNO-INFO

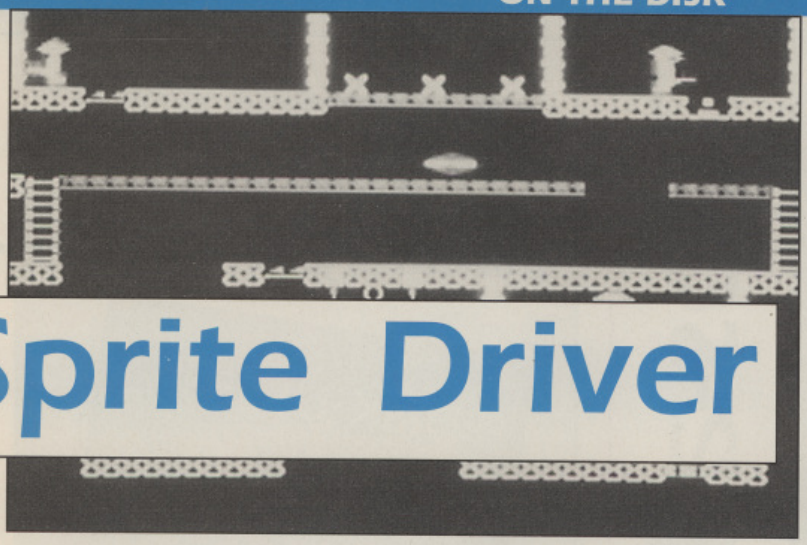
Page 39, third column, line 19 should read (0386) 551353. (My apologies to the gentlemen that owns the car-phone with the printed number.

March Issue

SCREEN SLIDER

Page 27, first column, line 30 has a large piece of text missing. The following needs to be inserted after the word 'downward': 'scroll until you have cleared the screen due to the aforementioned flick. Instead you should give the parameter a value of one. This will keep the interrupts enabled until they are switched off using SYS49152,0. It will also automatically prepare for a'

We apologise to all the readers that may have experienced problems due to the above errors. We also apologise most sincerely to Jason Finch for messing up his otherwise excellent programs.



Sprite Driver

A comprehensive utility for sprite manipulation and games design

By William Christie

Sprite Driver is a powerful utility which will enable the setting up of interrupt-driven sprite movement patterns. 'up to 255 pattern 'screens' can be designed and each screen can be accessed using a single POKE. Briefly, sprite movement patterns – the route a particular sprite will take as it travels around the screen – are entered as basic DATA statements. A call to the Driver routine (SYS 49152) will redirect the NMI interrupt and start decoding and executing the data held in pattern memory. Each sprite (up to 8 can be controlled) is then manipulated according to its pre-defined movement pattern.

The driver provides a large number of features to enable complex sprite patterns to be designed. These features include sprite movement in any one of the 8 'compass' directions, sprite positioning/plotting, movement speed, the setting up of data pointers, sprite colours, sprite on/off, and multicolour on/off. In-addition, a powerful sprite animator is present and control over animation speed and the animation parameters is possible. Sound can also be implemented, and here you can define and play a particular sound effect using any or all of the 3 voices. It is also possible to synchronise a pre-defined sound effect with up to 2 animation frames for each sprite. Voices can be shared between sprites. Sprite animation, sound synchronisation and colour information can be shared between more than one pattern screen. A facility for plotting text/graphics characters onto the screen is also provided.

Remaining facilities include the setting up of loop structures, whereby you can instruct execution to jump back to a pre-set point within a particular sprite pattern. You can also halt execution from a fraction of a second to hours/days, or permanently, or for a random period of time, or conditionally. Lastly, a POKE feature – operating over the whole of the 64's memory has been incorporated together with automatic screen advance and a sprite-sprite 'seek' feature. Each feature is identified by a single number (between 0-28 and between 252-255). I've called these numbers FLAGS as they determine what action is to be taken by the driver. A description of all available features and their correct syntax now follows.

Sprite Driver Features

0 – Instructs execution to jump back to the pre-defined start (see feature 22) of the current sprite pattern.

1,X – Moves sprite right until its X co-ordinate matches X.

2,X – Moves sprite left until it matches X.

3,Y – Moves sprite up the screen until its Y co-ordinate reaches Y.

4,Y – Moves sprite down the screen until it reaches Y.

5,0/1,X,Y – Plot sprite at X,Y setting its Most Significant Bit X (MSBX) to the value held in "0/1". e.g. 5,1,20,150 plots the sprite at 20,150 setting its MSBX to 1.

6,LB,HB – Halts execution of the currently sprite for the delay value held in LB,HB. A value of 255,0 gives a delay of approximately 1.8 seconds. For each increment of the HB value 1.8 seconds are added to the overall delay. Longer delays can be obtained by appending the desired number of "6,LB,HB" after each other or incorporating the delay within a loop. The length of the delay (and the speed of execution of all features) also depends on the setting of the NMI timers (at 56580-56583). These can be altered via the POKE feature.

7,SPEED (0-255) – Sets the movement speed of the current sprite. Due to the way this routine was programmed, a value of 0 represents the slowest speed, and increasing values from 1 to 255 will slow down the sprite.

8,0/>0, (multicolour1,multicolour2) – This sets the multicolour status of ALL sprites. If "0/>0" is zero then multicolour is OFF for all sprites and you insert "8,0" into the DATA statement only. To set a particular sprite(s) to multicolour mode you determine its bit position (sprite0=0,,,sprite7=7) and its decimal equivalent (sprite 0=1, 1=2, 2=4, 3=8,,,7=128). You then add the decimal values for each sprite together and insert it after the "8,". You must then enter the appropriate multicolours. e.g. 8,255,0,2 would turn on multicolour for all sprites and set mcol1 to black(0) and mcol2 to red(2). You would normally enter this feature only once, since it effect remains for all pattern screens (make sure that the first/start screen contains this feature).

9,shape pointer – Sets the current sprite's shape pointer. This feature also turns off the current sprite animation – so you don't need to enter a "12,0" before hand.

10,loop repeat number – Set start of loop and loop execution number (the number of times the loop will be executed). Only one loop can be running (for each sprite) at any one time, i.e. you can't have nested loops. Once the current loop has finished you can then execute another. The demo screens 1 and 4 make use of it to produce a coloured bar effect in the screen border.

11 – Go back to the loop start and repeat for the pre-set value.

12,0/1, (Lowptr, Hiptr, Speed, animation counter/0,0/1) – This is the sprite animator. If the first "0/1" after the "12" is set to 0 the animator

is turned off for that sprite, and you then enter only "12,0". To turn on the animator you enter "12,1, followed by the following parameters:-
Lowptr, Hiptr is the start and end shape pointers for the animation (Lowptr should be lower than Hiptr). Speed is the animation speed, where 1=fastest, 0=slowest, with 1-255 decreasing the speed.

The next parameter, if set to a non-zero value, will instruct the driver to animate in the chosen method for a limited number of times, after which animation is turned off for that sprite. A zero value causes animation to continue always.

The final parameter, "0/1", sets the type of animation. A value of 0 will cause animation to proceed from Lowptr to Hiptr+1. After Hiptr+1 animation is reset back to Loptr and started again.

A "0/1" value of 1 causes animation to proceed from Lptr to Hiptr and then from Hiptr to Loptr and repeat. This is designed to save sprite definition memory and is mainly used when the second half of an animation sequence is identical, but the reverse of the first half. Note, however, that in this case the actual Hiptr value is entered and not Hiptr+1.

13,LB,HB,Poke value (0-255) – This feature is identical to the POKE command in Basic. It takes a number and stores it into memory location HBLB. This is actually a very powerful feature on its own, as it allows pokes to all of the the 64's memory (and even to the driver program itself!). With it you can repeat all of the functions the Basic POKE command allows, eg. alter screen colours, alter memory configuration (bank switching), alter the NMI interrupt speed (poke 56581, value) – thus altering the speed of execution of the sprite driver (and of sprite movements, etc).

14,LB,HB,Colour, Character1, Character 2,,,, Char (n-up to 254), 0 – This feature allows you to print text or any other graphic character onto the screen at screen address HBLB. Each character is given the colour specified. The routine works by using CBM POKE codes – please refer to your computer manual for a table of these codes.

15,0-255 – Turn on/off sprites. This feature affects all 8 sprites in an identical way to feature 8. e.g. 15,0 turns off (disables') all sprites, 15,3 turns on sprites 1 and 2 only.

16, sprite colour – Sets the sprite colour (normal sprite colour register).

17, voice (1-3), Pulose Hi, AttDec, SusRel, Volume, HF, LF< Waveform – This feature defines and plays a sound. Please refer to your computer manual for selecting suitable values for each parameter.

18,X – Move sprite right and up until X is reached.

19,X – Move sprite left and up until X is reached.

20,X – Move sprite left and down until X is reached.

21,X – Move sprite right and down until X is reached.

22 – Define start point. This feature enables you to set the return point for execution to jump back to upon encountering feature 0. This allows you to begin your pattern data for the current sprite with initialising instructions e.g. set sprite colours, speed, animation, etc. you would then insert flag 22 followed by the actual movement data. At the end of this you insert flag 0.

23, Pointer1, Pointer2, Voice (1-3), Pulse Hi, AttDec, SusRel, Volume, HF, LF, Waveform – This feature synchronises a sound effect with animation pointers 1 and 2. This is useful for footstep-type effects, where pointers 1 and 2 would represent the shape pointers corresponding to the sprite's foot hitting the ground as it walks in each direction (e.g. right and left). As this feature is carried over from one screen to the next then careful planning will save memory.

24, Animation Speed (0-255) – This allows the animation speed of a previously defined animation sequence to be altered without affecting or having to re-initialise the animation. The bird in the first screen of the demo shows this feature in action. Notice also how the synchronised sound FX alters to match the speed.

25,animation counter – This allows you to alter the animation counter of a running animation sequence. Its effect is identical to that seen in feature 12, that is, a non-zero value will cause animation to proceed in the pre-determined manner until the counter reaches zero, after which animation is turned off. A zero value causes animation to continue always.

26.LB.HB – This is the random WAIT/HALT feature. It works in an identical manner to the normal WAIT feature (6,LB,HB) except that the LB value is randomized. e.g. 26, any number,0 will cause execution of the current sprite to halt for between 0 and 1.8 seconds. On the other hand, 26,LB,HB causes a halt of between HB*1.8 seconds and HB*1.8 plus 0-1.8(LB) seconds. This routine, integrated into the normal WAIT routine, makes use of the CIA timer.

27,LB,HB – This is the conditional version of the WAIT feature. In this case, execution of the current sprite will halt at this point if the memory location HBLB contains zero. Any value other than zero will cause the following feature to be executed. As a result, you should use the POKE feature (13) to set that location to the appropriate value. This is also a very useful command as it allows you to set up situations where, for example, an 'alien' is blocking a particular exit out of a screen. Only by completing a task (and POKEing that location with a non-zero value) will you be allowed to exit that screen. Another use could be where certain sprite patterns are only used if you have successfully completed a number of levels in a game. Demo screen 5 makes use of it to synchronise the movement of all sprites before they move off the ladder.

28,SPRITE (0-7),LB,HB _ With this feature you can instruct any sprite to seek out the current sprite. To use it you should first set location HLBL to zero with the POKE feature. Execution will continue at that point until HBLB is set to a non-zero value. Look at DEMO screens 4 and 5 to see it at work.

The final 3 features enable automatic access to different sprite screens.

252,screen number (1-255 – never zero!) – This instructs the driver to begin execution of sprite patterns comprising the specified screen number.

253 – Instructs the driver to move to the current screen+1

254 – Instructs the driver to move to the current screen-1.

255 – This is a permanent halt feature and causes execution of the current sprite to halt permanently. It can be

used in situations where sprites are not moving around the screen, or in cases where you are not using all 8 sprites for a particular screen. In fact, if you are not using a particular sprite you must insert this flag otherwise the driver will attempt to execute the next byte (usually a pointer to the next sprite) and may crash. (normally, though, if the driver comes across an unrecognised flag it will report this to you – see Error Debugging later).

Entering Patterns:-

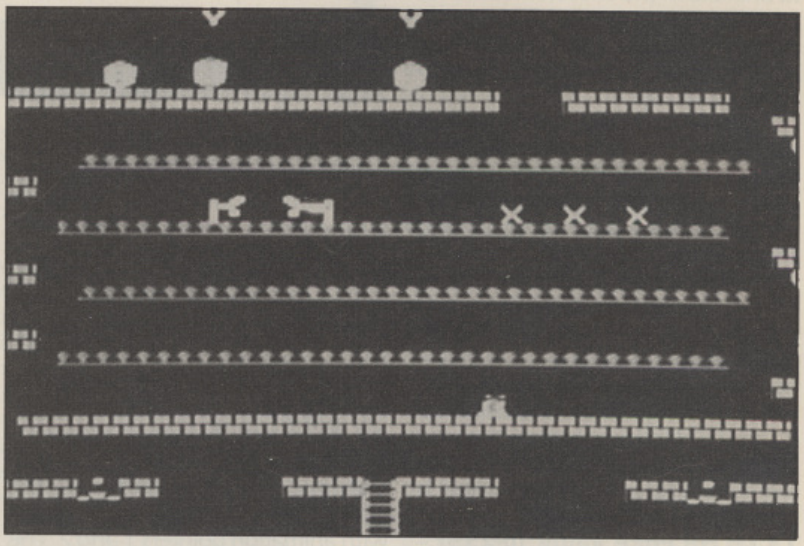
Now that you know the range of features offered how do you go about entering them into basic DATA statements, and how do you allocate each feature to each sprite? The Basic entry program, SPDRIVER.BAS, supplied on the disk should allow you to achieve this with some considerable ease!

Certain rules apply when entering pattern data, and this program contains a number of error-checking lines that help you keep to the expected entry format. This means that you will have to structure the data statements when dealing with each sprite, but you'll find this to be of great advantage later on.

To begin with, the DATA statements are at the end of the program. Each sprite's pattern data is entered one at a time and separated by a "-1". Sprite 1 is always the first sprite pattern defined, followed by sprites 2-8, in that order. When the last pattern (sprite 8) has been entered you insert a "-2" instead of a "-1". It makes the layout much easier to follow if you select suitable line numbers for each sprites data. I would suggest that sprite 1 is given lines 1100-1199, sprite 2, 1200-1299, etc. In fact, the error-checking lines report the current sprite number and screen number if an error has been detected (missing "-1", "-2", sprite data) and so, by following this layout you should find few problems in debugging errors of this type. So, to recap, you should end up with the following:-

```
1100-1198 DATA,,,, sprite1 pattern,,,
1199      DATA -1
1200-1298 DATA,,,, sprite2 pattern,,,
1299      DATA -1
.
.
etc for sprites 3-7
.
.
1800-1898 DATA,,,,sprite8 pattern,,,
1899      DATA -2
```

Remember, if you are not using a particular sprite you should insert a "255" and then a "-1"/"-2", as appropriate. So, having defined the first screen of sprite patterns and terminated with a "-2", all further screens are entered in a similar manner. Remember, also, that the first DATA statements of each screen refer to sprite 1, and the following refer to sprites 2-8, respectively.



The numbering of each screen is such that each screen is always the previous screen+1, and that the first pattern screen defined is always screen number 1. When you have defined the last screen you then enter a "-9" after the "-2".

Example:

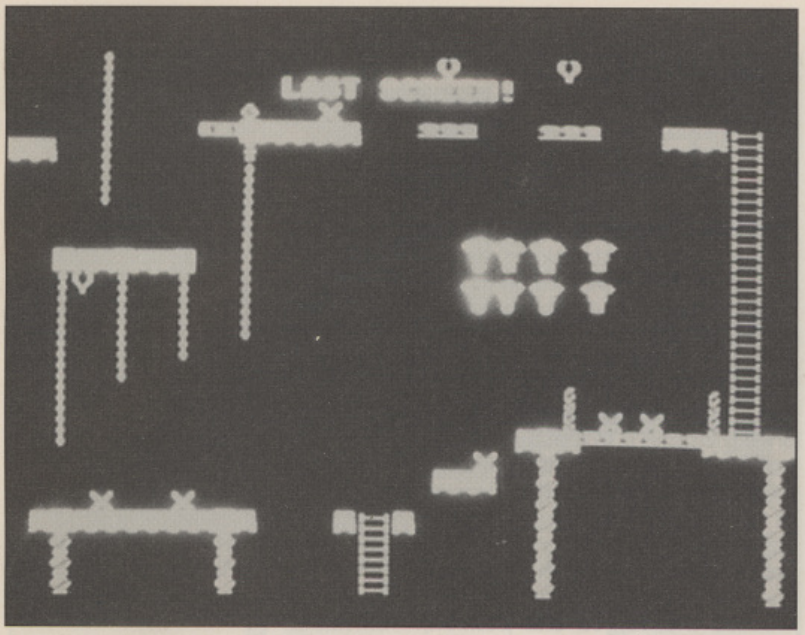
Suppose you wanted a sprite to start at 50, 100 with its MSBX set to 0, and travel to the right until it has reached 20, 100 with its MSBX set to 1 (rightmost side of the screen), and then travel back and repeat. You could enter something like the following:-

```
1100 DATA 15, 1 :REM turn on sprite
1 (and off all others).
1110 DATA 8,1,0,2 :REM Set multicol-
our on for sprite 1 (off for all others)
and set mcol1 to black (0) and mcol2
to red(2).
1120 DATA 16,3 :REM set sprite col-
our to cyan(3).
1130 DATA 7,5 :REM set sprite speed.
1140 DATA 12,1, 128, 133, 10, 0,0
:REM set up animation.
1150 DATA 23, 129, 132, 1, 0, 3, 0,
15, 5, 5, 129 :REM synchronise SFX
with shape pointers 129 and 132,
using voice1.
1160 DATA 5, 0, 50, 100 :REM plot
sprite at 50, 100 and set its MSBX to 0.
1170 DATA 22 :REM set return point.
1180 DATA 1,20 :REM move sprite
right until x=20 – this will automati-
cally set its MSBX to 1.
1190 DATA 2, 50 :REM move sprite
left until x=50 – its MSBX will be set to
0.
1195 DATA 0 :REM go back to pre-set
return point.
1199 DATA -1 :REM no more pattern
data for that sprite.
```

As you can see, the actual movement segment represents the last 6 bytes of the pattern, with the remaining 32 bytes representing those features that are needed to initialise the sprite. You can also see why feature 22 (set return point) and 0 (go back to return point) are needed – you don't want the sprite to be plotted at 50, 100 all the time (it won't get anywhere, otherwise!), and the animation will be reset each time.

Running the Demo

The Basic program supplied on the disk contains some demo screens incorporated into the data statements. This program also has a number of other useful routines to enable the design of each pattern to be made more easily. This aspect is discussed



later on. However, for DEMO purposes, I will describe the necessary steps to take to get it up and running.

First, load in the program – type LOAD "SPRITE DRIVER", or select it from the menu, then press RETURN. After the program has loaded, type RUN. After the main entry program has loaded you will be asked if you require certain programs and data. You should type "Y" at this point. You will then, in turn, be asked if you want to load in each file. Again, you should type "Y" after each prompt. Next, you'll be asked if you need to enter the contents of the data statements (containing the DEMO patterns) into pattern memory (POKEd into $6000-$6FFF). Type "Y" here. After a pause the program will ask you if you want to save the pattern data. If you enter "Y" the program will ask for a file name and save pattern memory (not the basic DATA statements – the driver acts only on pattern memory). It is only that amount of pattern memory used up by your sprite patterns (the DEMO patterns, in this case) that is saved. If you do type "Y" ensure, before hand, that an appropriate disk is in the disk drive. The program appends a ".PD" onto the end of the file name. The choice, here, is up to you, but saving pattern memory will allow you to use it later on in your own applications without having to go through the process of loading in and running the basic entry program. Next, you will be asked if you wish to run the pattern constructor. Type "N" here – the full operation of this module is detailed later. The program will then ask you if you wish to run the sprite driver. Enter "Y" here. finally, you will be asked if you wish to run the Sprite Driver with the DECRUNCH program (see Screen Designer). You should enter "Y" here also. If you answer "N" here you will exit this basic program and the Sprite Driver will be running under interrupt. You will then see the first screen (screen 1) of sprite patterns being executed. To access each further screen simply type "poke 824,screen number 1-5, for the DEMO screens.

If, on the other hand, you enter "Y" to this last question then you should see the first graphics screen together with the first sprite pattern screen being run. The following controls are now provided:-

Pressing "F1" will access the next graphics screen and sprite pattern screen.

Pressing "F3" will access the previous graphics and sprite pattern screens. (In the DEMO, 5 graphics screens and 5 sprite pattern screens are provided).

A number of other control keys are provided, and these allow you to construct your own sprite patterns more easily by allowing you full movement control over sprite 1 and allowing control over the shape pointer and colours of that sprite. In the DEMO, however, sprite 1 isn't used until screen 5 and so no effect will be seen here until this screen is reached. For DEMO purposes, press 'F1' or 'F3' only. You can, if you wish, exit this program by pressing the 'run/stop' key. If you do this then typing POKE 824, pattern screen number will access the selected pattern screen. To see the DEMO graphics screens type POKE 1022,1:POKE 820,LB (1-5 for the DEMO):POKE 821, HB (0 for the DEMO): SYS 52480 (to DECRUNCH and print that screen). Pressing "RUN/STOP" ≠ "RESTORE" will exit the Driver.

Using the Sprite Pattern Constructor

Part 1.

Despite its name, this module does not actually insert patterns into DATA statements. It does, however, provide full control over sprite 1, allowing you to move it around the screen and alter things such as sprite colours, multicolour on/off, and the shape pointer. It's an invaluable aid, though, especially when used with the Decrunch program.

The disk program "SPDRIVER.BAS" contains pattern data for demo purposes in its data statements. You should first delete these by removing all lines past 1000. The easiest way to do this is by loading in a utility program that has a delete function (e.g. the Devaid utility by Paul Eves, published in the March/April 1989 issue of CDU). Then load in the SPDRIVER.BAS program (this, later referred to as the Entry program, is loaded into $0801 since we are not using the Loader program). Next, delete all lines past 1000. Now disable the utility program – most are disabled by typing "RUN/STOP" + "RESTORE", otherwise, things such as re-defined function keys will affect the program. You should now alter the file names in lines 36, 40, and 44 to match your own previously saved sprites, re-defined characters and graphic screens, respectively (alternatively, you can use the default names to load in the demo files). Next, save the program using a suitable name (I would suggest you add a ".blank" at the end of the

name). You should now proceed as follows. Load in the program called "SPDRIVER.LOADER". Now list the program and alter the name of the program in line 20 to match your previously saved Entry program above – if required. Next, run the program. This places the main program at $7001, avoiding memory clashes with sprite, character, etc data.

You will now be asked which files to load in. answer these questions as appropriate to your needs. Next, you are asked if you want to enter the pattern data into pattern memory. Type "N" here since you are still in the process of designing these patterns. You will then be asked if you wish to run the pattern constructor – type "Y" here. Finally, you are asked if you want to use the Decrunch program also. Type "Y" if required, making sure the "decrunch.mc" and graphic screen files are in memory. If you are not using the Decrunch program then type "N". Either way, you will be presented with the first graphics screen or a blank screen, as appropriate. You will also see a single sprite, coloured white. From here, the following controls are provided:-

F1: Will access the next graphics screen.
F3: Access the previous graphics screen.
M: Toggle multicolour on/off for sprite 1.
C: Increase sprite colour register. Colour will wrap round from 15 to 0.
F5: Increase multicolour1 and wrap round if required.
F7: Increase multicolour2 and wrap round if required.
+: Increase sprite shape pointer (+1)
–: Decrease sprite shape pointer (–1)
.: Move sprite right.
,: Move left.
A: Move up.
Z: Move down.
1: Move up+left.
2: Move up+right.
Q: Move down+left.
W: Move down+right.
I: Increase ink (info bar) colour.
R: Toggle ROM/RAM characters.
SPACE-BAR: Print sprite X,Y co-ordinates, shape pointer, sprite colour, mcol1, and mcol2 – in that order – at the top of the screen. This key must be pressed after each sprite movement or other alteration to see the most up-to-date information.

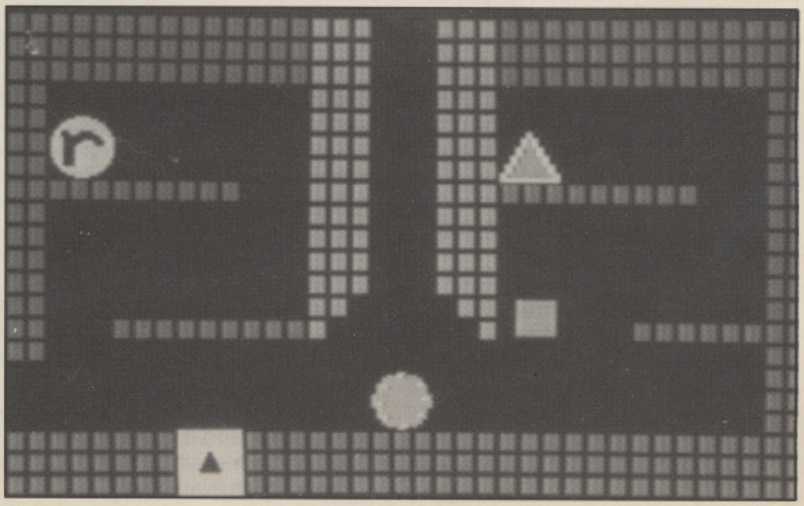
To use this module as an aid to constructing sprite movement patterns, first select the required shape pointer (+/– keys). If you intend to use animation then select a suitable frame/pointer within that animation. Next, choose appropriate colours for the sprite (make sure multicolour is set if needed). Now move the sprite to the start position within the screen and press the space-bar. Note down the information presented – you will insert these into data statements later on. If the sprite is to remain in a fixed position throughout that screen then move to PART 2. Otherwise, proceed as follows. Move the sprite using the appropriate keys to the end of the first part of the required pattern (i.e. move it in one straight line until you have reached the desired end position). Now, look up the feature table and write down the appropriate flag – e.g. if the sprite was moving right you would write down "1". Having done this, press the space-bar and write down the appropriate X or Y co-ordinate corresponding to that flag/feature – write this down next to the flag. Remember, if the sprite is altering its MSBX bit then you may have to move it again (sprites are moved until their X co-ordinates are the same as the final destination co-ordinate, irrespective as to whether this occurred in the MSBX=1 or 0 region of the screen). Now, repeat this procedure until the required pattern for that sprite has been completed.

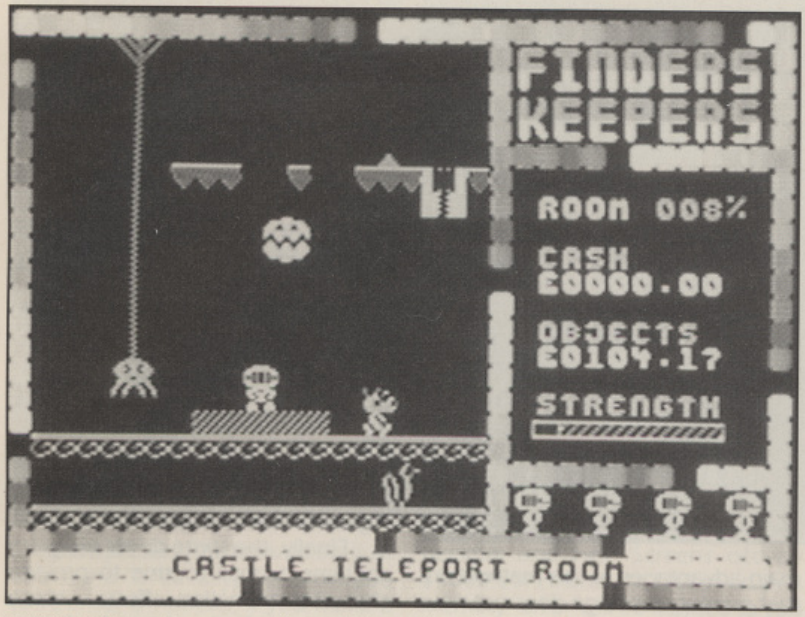
Part 2.
Having encoded the pattern information for one sprite you can now go on to build up each pattern for the remainder sprites in the same way. Remember, if you are not using a particular sprite you must write down (and later insert) a "255" in its pattern area. Once one screen of patterns has been defined you can then, if you wish, enter further pattern screens in an identical manner to the above.

Part 3.
With all pattern information for each sprite now written down you can begin to enter them into DATA statements. Simply follow the instructions given in the section "Entering Patterns". Remember, also, to insert the appropriate sprite and multicolour on/off values.

Sprite/Character data Incompatibility

Programmers who use sprites and redefined characters will be aware of the clashes of memory between these and basic program space. If you look at the memory map you will see that the regions between $1FC0-$3FFF (sprite and character data), $4000-$5FFF (crunched graphics screens), $6000-$6FFF (pattern memory) and $C000-$CFFF (the Driver and Decrunch programs) have been pre-allocated. Note that pattern memory can extend up to $FFFF – a lower limit of $6FFF is imposed by the entry program to allow the entry program it-





self, to sit in the area above this. Taking the above into account, I have provided a loader program, called "SPDRIVER.LOADER", which sets the start of Basic to $7001. the program then loads in and runs the main Sprite Driver. Bas program, where you can now run the DEMO patterns or construct your own. As a result, problems of memory clashes with data and the Basic program should not now occur. The only problem now is that you only have a maximum of $3000 bytes for the basic entry program (containing your pattern data in its Data statements). Taking memory and screen pointers into account (see Memory Format later) this should allow you about 2800-decimal bytes of actual pattern data before getting an 'Out Of Memory' error. This should be enough for most purposes – if you look at the sprite patterns for many platform games, for example, you'll notice that their movement patterns are very simple and in many of their screens only a few sprites are used (and, thus, the minimum amount of memory possible is used up – while at the same time maintaining the game's interest). On the other hand, it would be easy, at first, to go overboard using this utility and design patterns as complicated as possible. The best advice here is to try to achieve a balance between pattern complexity and memory usage. If you do find that you need more memory then there is a solution – load in the Spdriver bas program without going through the loader. This places the program into $0801 (normal basic start point). You can now use memory up to $5FFF – though you cannot use sprites, re-defined characters, graphics screens (and the Decrunch program). Also, you cannot now use the loader program to load your entry program into $7001 – you'll get an Out Of Memory error if you do. In this case, to see your patterns at work you should first save the entry program and then run it. You can now proceed to load in all required files and run the Driver and Decrunch – all necessary routines are at the start of the entry program and are thus unaffected by graphics and pattern data. Your sprites and re-defined characters will, however, overwrite the latter half of the entry program (containing the data statements) and this is why you must save the entry program beforehand.

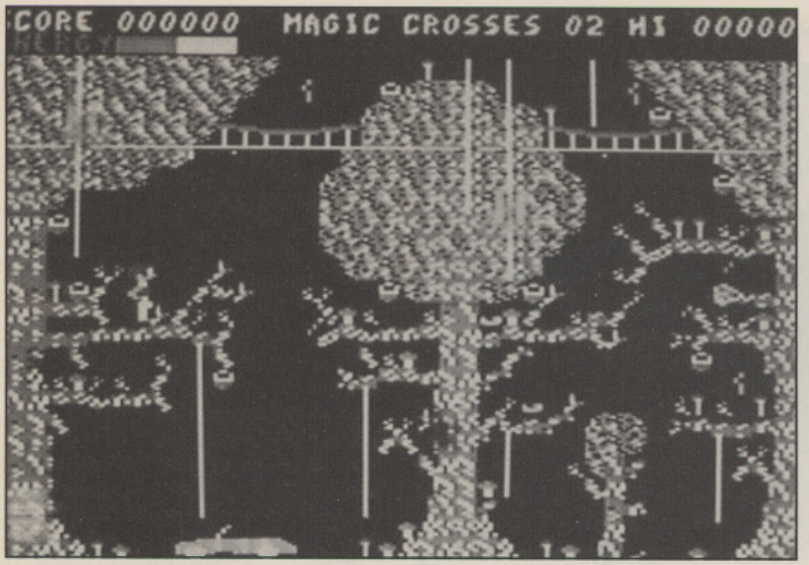
Re-defined Characters and Sprites

These have to be pre-defined, either manually or using one of the many available sprite and character editors. I use the excellent "3 into 1 plus" graphics editor by Tony Crowther, published in the Nov/Dec issue of CDU. Sprites and characters must occupy only the area $1FC0-$3FFF. The DEMO sprites use the area between $1FCO-$377F, while the re-defined characters occupy $3800-$4FFF. The Decrunch program contains a subroutine, called via SYS 52858, which copies ROM characters to the block $3800-$3FFF. If you use this facility for your own characters then you should switch in these characters using POKE 53272,31.

Error Debugging

Earlier on I mentioned that pattern entry format errors are easily detected by the Basic entry program. The error-checking lines here should ensure that you end up with pattern data for each sprite in the correct format. In-addition to this, the Sprite Driver, itself, has a built-in error-reporting routine. This routine is only entered when the Driver comes across an unrecognised flag (if you remember, the first number of each feature is referred to as the flag, and it tells the Driver which particular feature to execute). If an unrecognised flag is seen then the following occurs: interrupts are first turned off, then if you look at the top of the screen you will be first told that an unknown flag has been detected. Then you are told what actually caused the error – PEEK (250), the current sprite being looked at – PEEK (251), and the current screen number – PEEK (252). You'll find these three pieces of information, together with the format of the data statements, to be of great value as you proceed to debug the offending flag. To pin-point this flag within your data statements you do the following:-

1. Take a note of the screen number. You will now be able to determine that screen of pattern data from the data statements.
2. Note down the current sprite number. A value of 0-7 will be seen here, where 0 refers to those data lines containing the pattern for sprite 1, a value of 1 refers to those containing the pattern for sprite 2, etc – within the pattern screen.
3. Now, note the offending flag and proceed to look for it within those lines. Having located the flag you can then correct the fault. As you can see, you should be able to correct such errors quickly and with considerable ease. In most cases the error will be as



a result incorrectly incorporating parameters associated with a particular feature e.g. "8,0" and "8,3,2,0" are perfectly valid, where the first version switches off multicolour for all sprites – you DO NOT now add mcol1 and mcol2 after this, as in the second version (if you think about this then this actually makes sense, since it saves memory!). Pay particular attention to the animator, sound and character plot features. With practice, you should see very few of these errors cropping up.

Pattern Memory Format

The format I initially used was to allocate a fixed amount of memory for each pattern, giving 80 bytes max. However, this allowed for only 6.4 screens of pattern data per $1000 (#4096) memory locations – not a lot!. And also, it led to considerable memory wastage if you didn't use the full 80 bytes. It was, on the other hand, easy to program and comprehend during the development stages. However, a much better technique is used in the final version. The new method makes use of memory pointers (a similar technique is used in the Crunch/Decrunch programs). Here, each pattern screen begins with a pointer to the first byte of the next screen. Each individual sprite pattern also begins with a pointer, but this time it points to the next sprite pattern. Accessing each screen and sprite pattern is thus very fast (simply a case of switching pointers). However, the main advantages of this method are that memory wastage does not now occur, and you can have individual sprite patterns of any length – from one byte to several thousand bytes, if needed. On the disadvantage side, there is an overhead of 18 bytes per pattern screen, but this is a minor point since the advantages far outweigh any disadvantage.

These pointers are inserted by the Basic entry program – this is why you must specify the end (and, thus, start) of each screen and sprite pattern through the use of "-1" and "-2" in the data statements.

Memory Map (in hex):-

0801-2092 Screen Designer.
IFC0-377F Sprites.
3800-3FFF Re-defined characters.
4000-5FFF Crunched screen data.
6000-6FFF Sprite Pattern data.
7000-9FFF Sprite Pattern Entry prog.
8000-8490 Crunch/Decrunch.mc
A000-BFFF RAM under ROM UNUSED.
C000-C99D Sprite Driver.mc
C200-CC18 Getchar.mc routines.
CD00-CED0 Decrunch.mc
CED1-CFFF UNUSED.
D000-FFFF RAM under ROM UNUSED.

Zero page locations to avoid (hex):-
50-57,2 used by Crunch.
50-5D,2 Decrunch (both versions).
FB-FF, 2, 4E, 4F Sprite Driver.
Free Memory for use with conditional features (e.g. poke, conditional wait, sprite-sprite seek):-
CED1-CFFF, and any other area not used by your programs. Note that you cannot use the GETCHAR.MC program with the Sprite Driver, or the CRUNCH/DECR.MC file with the sprite pattern entry program.

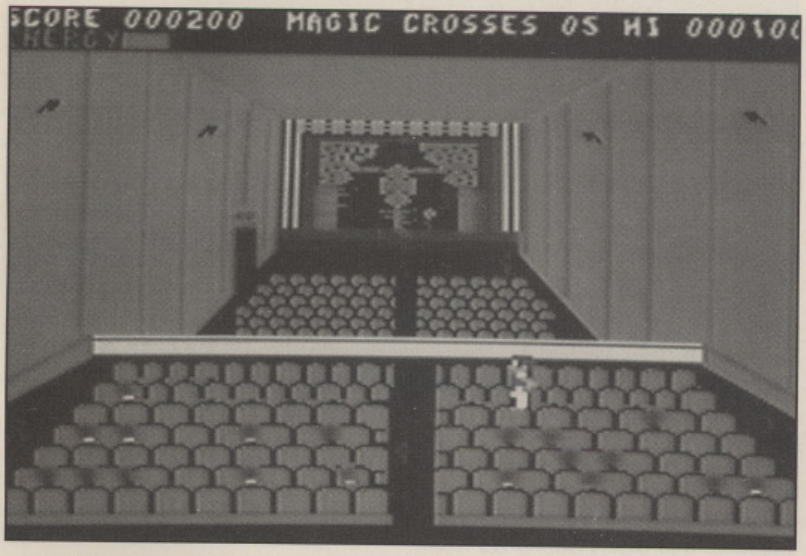
When using this utility SYS 49152 will, as mentioned before, begin execution of the first pattern screen. However, SYS 49157 will re-execute the current screen. This allows you to re-set the current pattern screen after e.g. a 'death' sequence.

Finally, the sprite driver program contains a small routine to provide character animation effects – these can be seen in the demo (e.g. conveyor belt). You can use this routine in your own programs if you wish – the re-defined characters correspond to SHIFT+(N,M,O,P, and T) in bank $3800-$3FFF. Character animation is turned on by incorporating "13, 16, 192, 1" into your pattern data, and turned off via the use of "13,16,192,0". Bear in mind that if the routine is turned on then memory corresponding to these characters will be corrupted.

Screen Designer

This utility allows the user to design a series of game screens. These can be saved and then re-loaded at a later date for further additions or modifications. Finished screens can then be used in your own programs.

To use this utility load in and run the program "SCREEN DESIGNER". You will then be asked to enter in a screen number between 1 and 255 (the crunch/decrunch programs actually address two-byte screen numbers but you'll find it virtually impossible to store more than 255 screens in memory). The exact number of screens allowed will depend on the complexity of each screen. Note that if you are using this program with the sprite driver make sure that each graphics screen number is the same as its corresponding sprite pattern screen. With all numeric inputs in this



program pressing the back-arrow (just above the "CTRL" key) will allow you to re-enter a number if you make a mistake.

Once a screen number has been entered you will be presented with a blank screen and a cursor at the centre. Screens are now built up character by character using the keyboard. From here, the following features, which act only on the current screen, are provided:-

F1 – Increase border colour and wrap around if required.

F2 – Increase background colour.

F3 – Changes the colour of every occurrence of the character under the cursor to that of the cursor. You will not be able to select colours currently used by any occurrence of that character in the current screen – this enables the preservation of screen features present as a result of colour differences between identical characters.

F5 – All occurrences of the current character under the cursor will be swapped with the character selected by the next key-press. Control characters, such as CRSR movement and colour will be ignored until a valid character has been entered.

F7 – Pressing this key gives access to the main screen processing options. From here the following are provided:-

LOAD – Loads in previously saved screens, character sets, or the disk directory if you enter "$".

SAVE – Saves all screens in memory.

MEM – Displays where the next screen will be placed in memory in both decimal and hex. Screens occupy $4000-$5FFF and an "Out of Crunch Memory" error will be flagged to you if your last screen tries to exceed this.

LIST – will list all screen numbers as they are stored in memory. This is useful for checking that there are no two screens with the same screen number. A maximum of 140 screens can be listed at one time. If you do manage to squeeze in more than 140 screens then pressing any key will list those remaining. The current screen will be cleared so make sure that it has been CHRUNCHed previously.

NUM – Allows you to alter the current screen number.

VIEW – Allows you to quickly view a range of screens.

GOTO – Will decrunch and print the specified screen.

BGN – Selecting this option will reset crunch screen memory back to the start.

WIPE – Use this option to clear the current screen.

COPY – Copies the character set to $3800-$3FFFF. Use this feature to make a **RAM** copy of the character set before loading in your re-defined characters (which must, of course, be compatible with this bank). By the way, copying screens is very simple. Simply GET the required screen, then set its NUMber and finally CRUNCH it.

RAM – Turns on the RAM copy of the character set (this is done by poking 53272 with 31).

MCOL12 – Pressing "O" will toggle character multicolour on/off for the current screen. Multicolours 1 and 2 can be altered by pressing keys "1" and "2". 'Press RETURN – To enter the screen editor from here. By the way, to print multicolour characters onto the screen simply press the CBM logo key+keys 1-8 to turn on multicolour and select an appropriate colour for colour RAM. Now, all characters typed will be in multicolour mode. To print in normal colour mode press "CTRL"+keys 1-8 to select a colour. Multicolour and normal colour characters can exist on the same screen at the same time. You may want to consult your computer manual for more information about this mode.

CRUNCH – Crunches the current screen and colour memory into a compact form and appends it onto the end of the last screen crunched. Each crunched screen is also given a header which contains the current screen number, background and border colours, multicolour status and multicolours 1 and 2

DEL – This option allows you to delete a screen or range of screens. Here, the following are valid if you wish to delete more than one screen: N1-N2 deletes all screens within the range N1-N2 inclusive, -N and N- will delete all screens up to and including N and all screens after and including N, respectively.

RENUM – Will renumber all screens in memory (the first screen is always given the number 1). You are required to enter a suitable increment number, though in most cases this number will be 1.

ALT – This useful option allows you to alter the number of any screen in memory. You are asked to enter the target screen number (the one to be altered) and then the new screen number you wish it to take.

Both the INSerT and DEL:ete key routines have been re-written to prevent scrolling problems; you cannot push or pull characters from one screen line to the next. The cursor colour can be altered in the normal way i.e. via the CTRL and commodore keys. You cannot, however, clear a screen using the CLR key (it would be too easy to press this key by mistake). To clear a screen you should use the Wipe option from the main menu (F7).

Custom character sets must be pre-defined and should be compatible with bank $3800-$3FFF. Here,

you should bear in mind that the F7 menu is printed with the assumption that the memory area defining the alpha-numeric characters should be compatible with the normal ROM version – don't define graphic characters here.

I mentioned above that if you require to reduce the amount of memory your screens take up then you should make them less 'complicated'. The next section gives you an idea of how to do this by explaining the memory format and how the crunching and decrunching process works.

Memory Format

The format chosen is based very closely on the way in which Basic programs are stored. Experienced Basic programmers will know that a system of memory pointers is used, where each basic line is preceeded in memory by a pointer to the next line. Line numbers are implemented to allow specific access to each line. A zero byte terminates each line and the end of a program is signalled when a pointer points to two consecutive zero bytes. With this in mind I came up with the following system. The base address was set to $4000 (this is easily altered, though). At locations $4002/3 a pointer is inserted which points to the end of the first screen. There then follows the screen number, in LB/HB format, and then the screen data, in a crunched format. This, then, comprises the first screen. Further screens are added in the same way. Each screen pointer points to the LB of the next screen pointer. As this system is so close to basic I thought that I could use the basic ROM routines to do things like accessing (GETing) any screen, delete screens and then use the LNKPRG routine to re-chain the memory pointers, etc. However, one main problem was encountered here. This problem was eventually traced to the fact that the basic ROM routines were written in such a way that a maximum of 256 bytes (including zero) could be addressed at each basic line, and this is the reason why each basic line is limited to 256 bytes. So, there was no alternative but to re-write the delete and re-chain routines. These had to be able to allow deletions to be made on screens which were more than 256 bytes in length. The root of this problem, though, lies in the method of screen crunching, which produces crunched screens of variable length. It is this variability which leads to the use of a system of memory pointers (pointers also allow fast access to each screen!). In contrast, if each crunched screen had a fixed length then manipulations such as accessing and deleting a screen would be relatively easy by comparison. Such a system is used with 'Block Screens' – which I'll say little more about!.

Anyway, back to the format used with this utility. Routines such as delete, get, list, etc were eventually perfected and are part of the disk file GETCHAR.MC.

A modification to this format is implemented where locations $4000/1 hold a pointer to the end of the last screen. This tells the Crunch program where to append the next screen and also allows the Screen Designer option MEM (display where next screen is placed in memory) to be programmed easily.

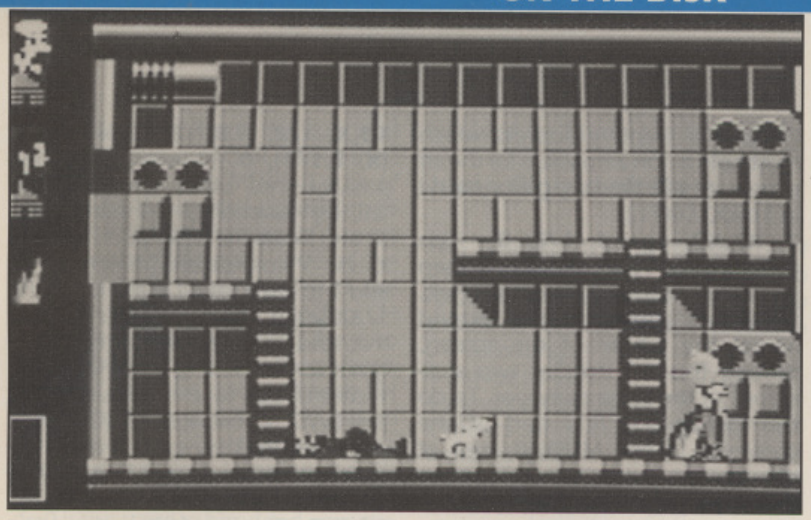
Now that you know how screens are stored and how a system of pointers is utilised, how are screens actually crunched?

The technique used here is a modification of a method based on character repeats. With this method the crunch program scans a screen from the top left to the bottom right and determines if any character sequence consists of the same character. If it doesn't then a specific 'flag'/byte is inserted into memory followed by a straight 'dump' of the mixed character sequence. A zero byte then determines the end of that sequence. If a repeat is found then another flag/byte is inserted into memory followed by the character code and then the number of repeats in LB, HB format. The whole screen is scanned and converted in this way until the complete crunched version is produced. Then, since the program knows the current memory location (which is where the next screen will go) it can now insert that location into the pointer at the start of the newly crunched screen. It also enters this location into $4000/1 (thus, updating it). What, then, about colour memory? Well, at the same time as scanning screen memory the crunch program also scans colour memory. In this case, though, a colour flag followed by a colour code is entered into the current screen crunch position only if the current colour memory location holds a colour that is different to the previous location – and only if the previous screen location is not a space character. The net result is that colour memory is integrated into screen crunch memory. In this way, a large memory saving is achieved. The best way to explain this is via an example. Suppose the screen is blank except for the word "LAST SCREEN" is in purple. We would then have as the crunched version:-

LB, HB, LB, HB, 255, 1, 1, L, A, S, T, , 255, 5, S, C, R, 0, 2, 3, 2, 0, 1, N, 0, 2, , 221, 3

The two pointers point to where the next screen will be added. The 255,1 then changes the colour to white. 1,text,255, 5, text, 0 then signifies a sequence of mixed characters with a colour change to purple at the word LAST, and a mixed character termination at the zero byte. There then follows 2, E, 2, 0 which represents the double E in SCREEN. The last letter, N, is then represented by 1, N, 0. Finally, the remaining spaces comprising the rest of the screen is represented by 2, , 211,3

The above system will produce a reasonable memory saving, but by modifying this method further we can save even more memory. The following, then, is the final outcome of these modifications:-
ch, LB=character repeat.
254, ch, ch,,,ch (n), 0=mixed characters.
255, C=alter colour.
The above version makes use of the fact that character repeats of over 255 occur very rarely in most games (and more so in platform-type games). Thus, we save one byte since a Hi-byte counter is not used. However, should a repetition of over 255 occur then a further two bytes would be used to count the remaining repetitions (representing an increase of one byte over the old system). The 'ch' above is the CBM screen character/poke code. You may notice a number of apparent problems with this system. First, by allowing a colour-change flag (byte 255) to be integrated into a mixed character sequence (saving



more memory) then we cannot now use character 255 (normally a reverse-CBM logo+B). Secondly, the use of the number 254 as a mixed character identifier means that we cannot now use this character (reverse-CBM logo+V). Lastly, since zero is used as a mixed character terminator then we cannot now use its character equivalent (normally a "@"). To prevent this use if your screens have been designed using the "GETCHAR.MC" routines (used by the Screen Designer) then you will be unable to select either of the characters "@" or reverse-CBM logo+B/V (poke codes 0,254 and 255). I hope, though, that this will prove to be only a minor problem.

If you are using the crunch/decrunch programs on their own then the crunch routine also makes a pre-check for these characters. If found then their screen locations are highlighted.

The Decrunch program then reverses the above process (simply a matter of acting appropriately to each flag). Note that this program does not clear the last screen before printing the next – it prints on top of the last screen (after all, there's no need for a CLR). This means that screens are printed faster and also gives a smooth transition between the last screen and the next.

Both the decrunch and crunch programs can be used on their own. The file "CR/DECR.MC" should be loaded in. You should then set the first screen to the start of memory using POKE 1022,0. POKE 820,LB: POKE821,HB then sets the screen number. SYS $8000 will crunch the current screen (and also sets 1022 to 1 enabling the next screen to be appended onto the end of the first screen). Further use of SYS $8000 (and POKE 821/1 with appropriate screen numbers) will cause all further screens to be added onto the end of the previous screens. POKE 820/1, screen number followed by SYS $8300 (or sys $CD00 if you are using the file DECRUNCH.MC) will decrunch and print out the chosen screen.

Memory location 2 is used by these programs to pass on information to the user according to the following:- if it contains 0 then all is well. A 1 means that the decrunch program has failed to find a specified screen. If the location contains 2 then an illegal character (the three mentioned above -codes 0, 254 and 255) has been used in the current screen. ~In this case the screen will need to be crunched. Finally, a 128 means that no more memory is available for crunched screens.

One possible use for this utility is with loading the disk directory. Load in any directory and list it. Now poke1022,0: poke820/1, number (screen/listing number) and SYS$8000 to crunch it. If the directory is too large to fit onto one screen then list the reaming part and crunch it again (don't poke 1022,0 this time). You can repeat this with more than one directory if you wish. You can even write comments onto the screen and alter screen colours before crunching. From now on, to look at any directory simply POKE1022,1: POKE820/1, screen number: SYS$8300. POKEing 1022 with 1 only has to be done once – it tells the decrunch program that there are screens in memory.

You can also use the 'GETCHAR.MC" file on its own. Look at lines 290-365 on the Screen Designer for a list of SYS calls and their actions. You should also list each routine in the program to tell you of any parameters you should set up before hand. The F1-F7 keys work as for the Screen Designer, though the F7 key returns control back to basic – you'll have to write your own routines to give access to each of the SYS calls from here. Pressing F7 also removes the centre 4 lines (using the routine at 51218). To return these lines and colour information you should SYS 51243.

Loading the Demo Screens

Five example screens are held on the disk in the file "SCREEN 5". To load them in press "F7" and then "L". Now press "D" to load from the disk. Next, type in the name of the file and press return. Control will then return to the editor. You should now load in the re-defined characters – but before doing this, press "F7" and then "P" to make a RAM copy of the character set. Now proceed to load in the characters using the filename "CHARS". To view each screen use the GET option from the menu. Five screens are provided (numbered 1-5). The re-defined characters correspond to keys SHFT+A-U.

Finally, you can easily alter the amount of memory available for your crunched screens. Load in the file "CRUNCH/DECR.MC" and POKE 33464/5 with the new end of crunch memory address. This value should be greater than 4096*4 ($4000) – the start of crunch memory.

Platform Encoder

This final module will allow the player to take control of sprite 0 and move it around the screen. Screen features, such as ropes/ladders, objects, walls, etc, are detected in an encoded format. This means that all screens are required to be encoded and entered as basic DATA statements.

As with the Sprite Driver, each feature is identified by a unique flag/number. All available features together with their correct syntax now follows.

1,X1,X2,0 – This defines a space in a platform, with X1 and X2 defining the left and right hand edges of the space. At this point I should mention that all features described relate to a platform Y co-ordinate (see DATA format

later). Note also that all features end with a zero and that all parameters must have values between 1 and 255.

2,X1,X2,Yend, 1/4,0 – Rope/Ladder – X1 and X2 are the left and right edges, Yend is the Y co-ordinate of the end of the ladder. If "1/4" = 1 or 2 then the ladder extends upwards, with a value of 1 allowing the player to move up or down and a value of 2 allowing movement upwards ONLY. A value of 3 or 4 applies to ladders extending downwards, and 3 allowing movement down or up and a value of 4 allowing movement down ONLY. To encode a rope simply make X1 and X2 the same value (you will be unable to move left or right now – meaning you won't fall off the rope). Unfortunately, you cannot have ladders that cross the boundary of MSBX=0 into MSBX=1 (make sure that ladders which fall in the area of the screen covered by sprite X=1-255, MSBX=0 also end in that area, and similarly with X=1-255, MSBX=1). When defining ropes/ladders if you wish the player to fall off the bottom when moving down then either define an "invisible" platform at Yend which consists of a space extending across the whole screen, or if a previously defined platform is the same as Yend then make that part of the platform corresponding to the rope/ladder X1 and X2 a space. If on the other hand, you wish the player to stop at the bottom of the rope/ladder (i.e. and not fall off) then make sure Yend is not a platform Y co-ordinate. To prevent the player from moving up off the top of a rope you should define the rope extending down from within a space.

4,X, Yend, 0 (no feature 3, 5, 9, 10, 11) – Defines a wall. Yend is the minimum Y co-ordinate you must reach to jump over the wall. If the wall is >2 pixels in width then you may have to define the wall twice, one each for walking into each side of the wall (x values will be different).

6, X1, X2, Yend, 1/2, 1/2, (LB/HB), Speed, 0 – This defines a lift. Yend is the destination Y co-ordinate you (sprite 0) will reach. If the first 1/2 is a 1 then the lift moves up, a value of 2 moves you down. If the second 1/2 is 1 then you leave out the LB, HB values and enter the speed (1=fast, 255=slow). If 1/2 is 2 then you enter LB,HB – a memory address which must hold a non-zero value for the lift to operate (free memory can be found in $9B00-$9FFF).

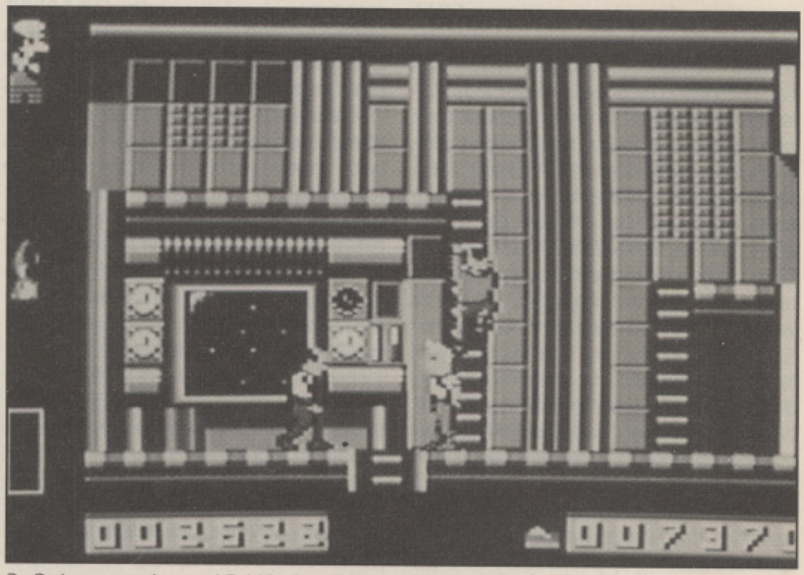
7, X1,X2, Speed, 1/2, 0 – This defines a conveyor, If 1/2 is 1 then you move right; a 2 moves you left. Note that conveyors can extend across the whole of the screen, though this will require two definitions of the same conveyor. To do this define the first half of the conveyor in the MSBX=0 area of the screen with X2=255, then define the second half in the MSBX=1 area with X1=1 (make sure you enter an "11" as mentioned before.

8, X, 1/2, (LB/HB) – This defines a ground object. If 1/3=1 then you lose a life if you reach this point. You must also leave out the LB, HB values. If 1/3=2 then you insert LB,HB. Reaching this point now sets HBLB to 1 (use with conditional features). A 1/3 value of 3 clears location HBLB (sets it to zero) – remember to include the address. These last two features can also be used to communicate with conditional features of the sprite driver.

12, X, 1/2, (LB,HB), Xnew, Ynew, 1/2, 0 – This defines a transport (moving within a screen). If 1/2=2 then LB, HB must be entered and that location set to 1 to operate. With a 1/2 value of 1 you leave out LB, HB. Xnew and Ynew are the destination sprite X and Y co-ordinates. If the final 1/2=1 then sprite 0's MSBX is cleared. A value of 2 sets it to 1.

13,X, 1/2, (LB,HB), SX+1, 1/2, (X, Y,1/2), 0 – This feature allows passage into other screens. "1/2, (LB,HB)" function as per feature 12. SX and SY are the destination screen's X and Y co-ordinate (remember to add 1 to each value). If the following 1/2 is set to 1 then you enter "0" after this, and you are transported to the given screen, setting sprite 0's X and Y co-ordinate to the default values (see DATA format later). If 1/2=2 then you enter the new sprite X and Y co-ordinates followed by 1/2 9MSBX=0 or 1, respectively).

14,X,0 – This final feature should be entered in the last screen. Upon reaching X the game is reset back to the beginning. All collectable objects reappear. The current score is carried over/not reset.



Collectable Objects

Collectable objects are defined separately in line numbers 6000 onwards, and consist of a single character. A maximum of 7 objects are allowed per screen. They take the following format:-

X1, X2, 0/1, Y1, Y2, CH, CHr, Co1, LB, HB, Sc/0

X1, X2 and Y1, Y1 define the left, right, top and bottom edges of the object (X1<X2,Y1<Y2) – as the sprite hits it. Use the Screen Designer to determine these values by moving the sprite firstly to right until it just hits the object. Now press the spacebar and note down the X co-ordinate. Repeat this hitting the object from the

right and then from the top and bottom of the object. If 0/1=0 then the object is in the MSBX=0 area of the screen. If this value equals 1 then the sprite's MSBX must equal 1 to hit it. CH is the CBM poke code of the original character while CHr is the code of the character replacing CH one taken (Thus you don't have to leave a blank space once the object is taken). Co1 is the colour of both CH and CHr. LB and HB is the screen address of the object. This is calculated automatically by using the "X" function of the Screen Designer. Sc/0 if >0 is the score to be added once the object is taken. IF THIS VALUE IS ZERO, however, then you gain a life once you take the object (useful in later screens of a game). If you do not use the full 7 objects in a screen then you must pad out the remaining objects with 11 zero's per unused object.

Data Format

Each screen is given a header of 5 bytes. The first two bytes are the screen X and Y values (note that you enter the actual number – don't add 1). The next three bytes are sprite 0's default X and Y co-ordinates and its MSBX value (0/1). These are the co-ordinates the sprite is given if the player loses a life or enters another screen with the sprite X and Y values unspecified (see feature 13). You then enter your encoded screens platform by platform in the following manner:-
Line No, Y, 888, data, 11, data, 0 (or 9,999)
Line No is the current Basic line number. Y is the platform Y co-ordinate (sprite Y value standing on the platform).
You must then enter "888" – this is where the pointer to the next platform is inserted (automatically). After this you enter all those features on that platform that can be hit by the sprite when its MSBX=0 (ie 1st 4/5 ths of the screen from the left). Remember to enter - after each feature. If you do not have any features after this point on the screen (ie MSBX=1) you enter another 0 and begin encoding the next platform (the last feature thus ends with two 90's. If, however you do have features that can only be reached when MSBX=1 then you enter 11 after the last 0. You then enter the remaining features of that platform followed by 0 (from last feature), 0.

Jump Right/Left Data

This data defines the movement of the sprite as it jumps. You are free to re-define this table. To do this you should select appropriate values from the following:
0 moves sprite Y-1
1 moves sprite X+/-1
2 moves sprite X+/-1 and Y+1
3 moves sprite Y+1
4 UNUSED
5 moves sprite X+/-1 and Y-1
6 signals the last byte.
These values should be entered in place of those currently present in lines 1340-1359.
To begin encoding your own screens you should first load in the program "PLAT.BAS" the delete lines 1400-1870 (leaving the "-1"), and also delete lines 600-8999 (leaving the "-2"). You should now save the program using a suitable name e.g. PLAT.BLANK.

Example

Imagine a screen (number 1,0) that consists of two platforms – one with sprite Y co-ordinate of 100 and the other with a Y value of 170. The screen also has one ladder with X1 and X2=50 and 64 in the MSBX=0 area. The ladder starts on platform Y=170 and extends up to platform Y=100. On platform Y=170 there is a conveyor that extends from X=200 (MSBX=0) to X=20 (MSBX=1). There is also a door to screen 2, 0 on platform Y=100 at X=25 (MSBX=1), and you wish the player to start at X=100, Y=100, MSBX=0.

These co-ordinates will have been previously determined by using the Screen Designer and moving the sprite over these objects, pressing the space-bar and noting down the relevant values.

To represent this in DATA format you first enter the 5 header bytes i.e. screen X,Y, Sprite X, Y, 0/1. The actual values entered would be "1,0, 100, 100, 0". On a new line you then enter all objects on the first platform (at the top of the screen). This line would appear as:-

100, 888, 2, 50, 64, 170, 3, 0, 11, 13, 25, 1, 3, 1, 1, 0, 0

The first 100 is the platform Y co-ordinate followed by 888 (pointer). "2, 50, 64, 170, 3, 0" defines the ladder extending downwards – every ladder, thus, has to be defined twice if you want the player to be able to walk onto both ends of a ladder. If, however, the player can jump onto a ladder then you can, if you wish, define the ladder only once. The next byte ("11") tells the computer that features after this point can only be hit when the sprite's MSBX=1. The final feature defined, "13, 25, 1, 3, 1, 1, 0, 0" allows the player to enter screen 2, 0. Note, however, that one is added to each screen number – this avoids the use of zero's, which would confuse the program into thinking that the end byte of that feature has been reached. Not that this feature is unconditional (no LB, HB) and that default sprite X, Y values are to be used when entering the next screen.

This then is the encoded version of the first platform (remember to add an extra zero after the last feature defined on each platform). The next platform (Y=170) would appear as:-

170, 888, 2, 50, 64, 100, 1, 0, 7, 200, 255, 30, 1, 0, 11, 7, 1, 20, 30, 1, 0, 9

As before, you enter the platform Y then 888. The ladder is now defined a second time with Yend=100 and the ladder direction set to 1 (allowing movement up or down). The conveyor is now defined in the MSBX=0 area. Note that X2=255. The remaining segment of the conveyor is now defined in the MSBX=1 area of the screen (via the use of "11". Note that X1=1.

Since this is that last feature on the last platform of the screen you enter a "9" after the last zero.

This then is the encoded version of the whole screen. further screens can be added next (or at a later date – after saving the program).

In General

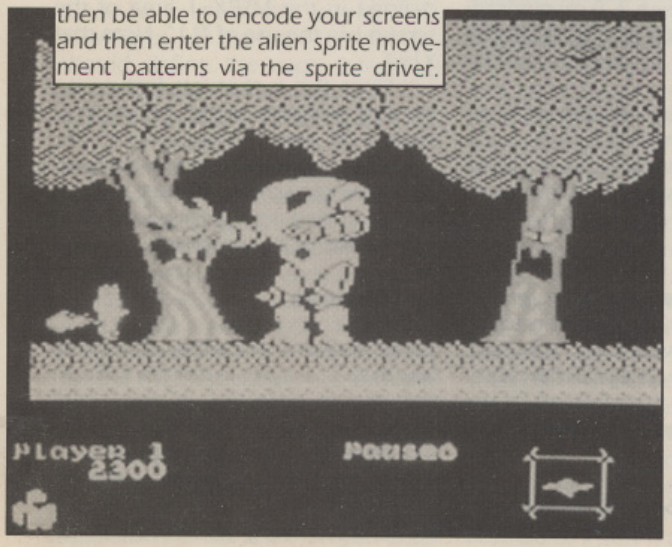
Each platform encoded should be less than 253 bytes in length, though this should pose few, if any, problems.

There is no limit to the number of features you can have per platform/screen, though the gameplay will slow down if too many features are pres-

ent. Games can be made more complex by the use of conditional features e.g. walking over a switch which allows another feature e.g. lift to become functional. Note that the player can also communicate with the sprite driver by the use of "poke" function of feature "8" and any conditional feature in the driver. The driver can also be made to affect the player in that a sprite can switch on/off any of the conditional platform features. A note of warning here; when using conditional features be sure to use free RAM/memory (see the memory map) and to keep track of memory locations used, and for what purpose.

Error debugging here can, at first, be a difficult process. The main points to remember are to define each platform clearly (don't define two platforms on one line), and to pay close attention to the parameters associated with each feature and any multi-function feature. Separate each screen with REM lines. You can also include REM's between platforms to make future alterations easier.

When designing games you should start with the screen designer. This is the easiest module to use since all functions are automatic. Make sure, though, that no lifts or rope/ladder extend across the MSBX=0 to MSBX=1 region of the screen – you'll have to use the 'pattern constructor' module of the sprite driver to check this. Do not enter collectable objects (platform.mc does this) but do note down their screen addresses – given in their LB, HB format. You should then be able to encode your screens and then enter the alien sprite movement patterns via the sprite driver.



Both the platform encoder and sprite driver will take some practice to get used to, however, once mastered it shouldn't take you long to complete one game screen.

Note that due to memory limitations imposed by the object table of plat.mc you are limited to a maximum of 32 game screens.

Finally, due to its complexity the chances of encountering problems when using this system for the first time is probably quite high. To start off with, make your screens simple, with few platforms and features. Then gradually build up to more complex screens as you become more familiar with the format of each feature.

Memory Map

$0800-onwards Plat.bas
$8000-$8F9D Plat.mc
$9000-$999F Object table (32 screens)
$99A0-$9A7F Object condition table
$9A80-$9AFF Jump right/left data.
$9B00-$9FFF Free memory for use with condition features.

Note that Plat.bas will, if large enough, extend into the sprite and character definition areas. If this happens (you'll notice corrupted sprite shapes) you should load in your sprites after running Plat. Bas. Note also that there is no provision made to allow you to save a complete game – this would require the saving of the whole of memory from $0800-$D000. This would mean very long saving and loading times. You would not be able to use a Disk Turbo program since most use memory within the area occupied by the game. As a result, to run a game you should load in the Spdriver.Loader program and run it, then press RUN/STOP+RESTORE and finally load in PLAT.BAS and run it. This will give you a SYS address. Simply move the cursor over it to run the game. For those fortunate enough to own a "back-up" cartridge e.g. the Expert/Action replay you can make a back-up copy of the game by breaking into the initial screen. You should now be able to load in the game using the cartridge's fastloader.

Screens

Add an extra 2K of screen storage to your C64 with this handy utility

By Phillipe Bastings

Working on a big computer might not be so exciting as working on my good old C64, but at least, it gives me ideas of programs to develop for the C64!

One feature that is very useful when developing a program is the possibility of scrolling up or down the screen or to restore the previous screen to see what you typed some minutes earlier and then to come back to the current screen.

From now on this feature is available on your Commodore 64 thanks to this handy utility.

SCREENS adds 2 Kb of screen memory to the usual 1 Kb (1024-2023). Every time the screen scrolls up, SCREENS will store the line that is going to disappear, the top line of the screen, to another place in memory and gives you the possibility to bring that line back to the current screen simply by pressing a function key and without losing any information of the current screen.

Note that only the characters are saved, but not the colours. This should not disturb you as long as you don"t change the current colour every time you enter a new program line!

As mentioned above, a simple key press will bring the previous text back to the current screen. The 8 function keys are redefined in this way:

F1 => PREVIOUS LINE
F2 => PREVIOUS PAGE
F3 => NEXT LINE
F4 => NEXT PAGE
F5 => LIST
F6 => LIST + CARRIAGE RETURN
F7 => CLEAR CURRENT SCREEN + CURSOR HOME (same as SHIFT/CLR-HOME)
F8 => CLEAR ALL SCREENS + CURSOR HOME

How it Works

At initialization (SYS 51152) the KERNAL ROM is copied into RAM (PEEK (1)=53). This is necessary to redirect the usual scrolling up routine (59626) to a new routine.

Every time the system is calling the scrolling up routine, SCREENS scrolls up screen 3 losing line 0, then stores line 0 of screen 2 to line 24 of screen 3, then scrolls up screen 2, then stores line 0 of the current screen to line 24 of screen 2 and finally returns control to the system's own scrolling up routine.

As you might guess, this causes a lot of memory moving every time the system performs a scrolling up of the screen. You will be aware of this when listing a program, as you will notice that the speed is reduced. Figure 1 shows what is going on.

When you press function key F1, the opposite is happening, except that nothing is lost. Line 24 of the current screen is stored to line 0 of screen 3 using a 40 bytes long buffer. (See Figure 2).

The same is happening when using function keys F2 or F4, except that in this case a 1 Kb buffer is used. This buffer uses the 1 Kb of RAM just before the Basic interpreter (39960-40959). This should not disturb your basic programs as long as they are not longer than 37 Kb. The only thing you have to be aware of is that Basic stores strings variables starting from the top of the free basic RAM. So if you type in DIRECT MODE, A$="YOUR COMMODORE", then type PRINT A$, the string A$ is displayed. But if in the meantime you press function key F2 or F4, the variable A$ will be destroyed by the buffer. This is not true when a program is running because the function keys are disabled and are automatically re-enabled when the basic programs stops!

You will notice when using the function keys, how fast everything happens although 4 Kb of memory are moved every time. To do this I have used machine language of course, but also a routine of the Basic interpreter (41919) that moves one block of memory from one place to another. This is the reason why I could not use the RAM under the Basic ROM for the storage of the two more screens and for the 1 Kb buffer.

SCREENS redirects the scrolling up routine of the KERNAL, but also the IRQ vectors (788-789) for execution of the SCREENS routines when a function key is pressed. When a basic program is running, SCREENS deactivates the function keys, but when the basic program stops, SCREENS will automatically reactivate them.

How Memory Is Used

49152-50151 : SCREEN 2
50152-51151 : SCREEN 3
51152-52140 : ML + IRQ ROUTINES

828-882 : TRANSFER FROM ROM TO RAM. THIS AREA IS ONLY USED AT INITIALIZATION

39960-40959 : BUFFER

Some zero page locations are also used as working storage areas. These locations are 2 and 251 to 254.

How to Start with Screens

When you want to work with SCREENS, select the program from the menu. The machine code will be poked into memory and initialization will take place. A message will appear on the top of the screen when the program is ready.

Load a basic program or type in a program.

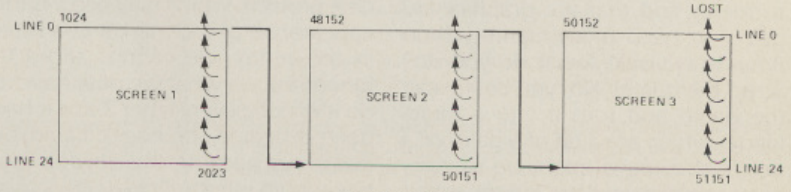


Figure 1. Scrolling

If you press RUN/STOP RESTORE, SCREENS will be deactivated. To reactivate, type SYS 51152. Note that when SCREENS is initialized, all screens are cleared. (Same as F8).

REMARK : SCREENS might not work correctly with some cartridges!

Data Compression

Data compression techniques explored in an easy to understand method

By Neil Higgins

In this article I would like to examine a technique which is primarily concerned with reducing the memory used by a block of data, and programs stored on disk. Even though this method has been written and tested on the Commodore 64 it will easily convert for use on any other computer, especially if you have a knowledge of machine code.

In general, you will find that most computer files contain large amounts of repeated numbers, this is particularly so in programs containing graphics data such as hires screens, character sets and sprites. We can exploit this fact, and in some cases depending on the contents of the file, achieve savings of up to 50% and more. You may be thinking, what is gained from making a file smaller? Well the most obvious advantages are that it will occupy less disk space, and the time taken to load the file could be considerably reduced (especially under normal load conditions on a 1541). Imagine you are writing the next block busting arcade conversion and you suddenly find that the graphics are going to need more memory than what is available, what do you do? Scrap the game? No, you could split the game and turn it into a multi-loader which are a bit of a pain, or a better alternative might be to compress some of the data and squeeze it into one file.

Probably the most common method of compression, which relies on the data containing long runs of repeated characters, Run Length Encoding. If you take a look at Diagram 1(a) you will see a string of characters, which could be encoded more compactly by replacing each repeated part with a count of the number of times it is repeated, followed by the character itself. We could say that we have 4 B's followed by 5 A's etc, which continuing to the end of the string would compress into that of Diagram 1(b), and as you can see we have achieved a saving of 10 characters (48%). Note, it is pointless to encode a run of less than three characters, since two characters (count, char) are needed for encoding. The major fault with this routine is that the string to be encoded must only contain letters or else when we come to decompress the string back to its original state we won't be able to distinguish whether we have a normal digit or a count for the next character. We can solve this problem quite easily by using another character in the encoding process, for simplicity we will call this the 'marker', now each appearance of this character signals that the next two characters consist of a (count, char) pair, just like the previous method. To make this a little clearer take a look at Diagram 1(c) in which we have used the letter Z as the marker to encode the string in Diagram 2(a). With the use of a marker though, we have reduced the saving from 10 to 6 characters (29%) but the method is more efficient because we can now use digits or any other character in the string. One problem which may occur is the appearance of the market character in the string, we cannot afford to ignore this, so what we now have to do is encode every letter Z as if it had been repeated, or else it could be disastrous because the decode routine would interpret every letter Z as a marker. If you take a look at Diagram 2(a) you can see our string contains five 'Z's after encoding we end up with Diagram 2(b) in which the first Z, even though it is not repeated has been encoded with a count of one. Another method that can be used to encode a marker, is to use a count of zero, that is Z0 would represent any occurrence of the letter Z, but as you can see in Diagram 2(c) after encoding we have ended up with string three characters bigger than the original 2(a) which defeats our objective. At this stage I must point out that in some cases you will end up with a bigger string even after compression, otherwise you could continually apply the methods until the string is very small, which is just not possible! So it is entirely up to you as to which method you prefer to encode the marker, but for the rest of this article we will stick with the first of a (marker, count, marker). There is a simple way we can find a marker which would result in the most efficient compression, but could add a considerable over-head in time, especially on large blocks of memory or files. Can you guess what it is?

Keep reading and all will be revealed (ha!).

Okay, thats most of the theory out of the way, we shall now take a look at converting all this information to work on the 64. In fact we do not need to change much, its just a case of using numbers instead of characters, as most of you should know each memory location in the 64, can only hold a value in the range 0-255 ($00/$FF hex). Knowing this, lets imagine we have a block of memory containing a small program and some graphics data that we we want to compress, first of all we need to choose a suitable number to use as the marker, and as I mentioned before there is a way of finding one. What we have to do is search the whole block for any number that does not appear, or appears the least times. However, if we were doing this on a large block of memory, even using machine code, it could take a long time, so a quick alternative is whats needed. The number I tend to use as a marker is 239 ($EF), for the only reason that it is not used as an op code in the 6510 instruction set, so the chances of it being in the object code are pretty remote, of course we have no way of telling what numbers make up the graphics data, unless the search method is used. If you take a look at Diagram 3(a) you will see sixteen hexadecimal numbers just as they might be displayed using a monitor, they have been compressed into Diagram 3(b) using the marker $EF, and thats it, just a basic change from characters to numbers.

On the disk is a machine code

program called compression that puts all this theory to the test, in that it will compress and decompress any number of bytes (default is 16). The program was written using the 6510+ assembler but should be compatible with most assemblers that use Basic source files. It is well documented but I would first recommend you to read it thoroughly (get a listing if you have a printer) before trying it out, that way you can be sure how it really works. The routines were actually written for a program I was developing which compressed data as it was loading from disk, and as you know the disk only outputs one byte at a time, hence the need to store the first, input the second byte, compare them for any repeats, and so on. You might like to put all this knowledge into practice and use the routines as a base for writing your own program compressor/compactor, if you do decide to have a go then I wish you good luck.

(a) BBBBAAAAADDDECCCCCCFF

(b) 4B5A3DE6CFF

(c) Z4BZ5ADDEZ6CFF

Diagram 1

(a) ZCCCCBBEZZZZEEEE

(b) Z1ZZ4CBBEZ4ZZ4E

(c) Z6Z4CBBEZ0Z0Z0Z0Z4E

Diagram 2

(a) AC 00 00 00 00 00 7F BC FF FF FF FF EF 70 6E 55

(b) AC EF 05 00 7F BC EF 04 EF EF 01 EF 70 0E 55

Diagram 3 **0 = zero character**

Hires Animator

Get those single or multicolour images animated just like the movies with this easy to use utility

By K. Hall

Creating Hi-res images that have animation is not the easiest of tasks on the C64. This utility, which is a concept I have had for a long time, should take some of the hard work out of it.

To start designing your character for further use you should run the basic programme DESIGNER OFFSETS which pokes into the memory a series of offsets for the X and Y coordinates during the drawing routine, after loading the ANIMATEMC programme.

There is a simple demo programme included which uses the file created from the designer (this is called TEST1 and is suffixed with .ANM) and a file called SPRITE FILE 1.

The demo contains three main routines which show a Hires character operating in interrupt-driven mode, a hires plot routine showing an abstract simulation of resonance, and a basic driven Hires character showing how information about the characters position can drive a hardware sprite.

The Hires animator program was created to take the heartache out of designing a sequence of animated images in single or multicolour mode. The concept originally came about after I thought that it might be possible to create the illusion of changing the position of an actor's body from a range of photographs to give the impression of doing a variety of stunts while the actor was safely drinking gin and tonics in the local pub. All the hard work would be done by computer to generate a scene in the film. From this rather grandiose idea came this rather simplified programme and, obviously due to the restrictions of memory and the C=64s 320 * 200 resolution, a lot cruder than any that would be required for sequences in a film. (Since writing this programme I have watched a video of "The Running Man" where a similar idea to the one I originally had was used in the story to create the effect of Ben Richards fighting with another man, when it was in fact a stunt man with a computer generated overlay of a digitised photograph). Perhaps someday I might, with a different algorithm to the one I'm using here, be able to create something a lot more realistic than this for a computer to use, (I'm thinking of the Amiga next) using perhaps a collage of digitised images and a Video to bring this about. However until then you might find these a useful series of routines until both you and I are slightly richer.

The programme will create a series of images and reverse them from one single image drawn using a joystick (or mouse in joystick mode) in either two colours or multicolour mode using bit mapped memory, and allow you to save them for use in your own programmes. Each image can be half the height of the screen and up to quarter of the width and the amount of apparent movement can be varied for both the arms and the legs of the character created.

When I first came up with the idea for this programme, I wrote the algorithm to recreate the images in Laser Basic, and the method employed to do this was to scan the image bit by bit and manipulate the legs and arms accordingly. As you can imagine this was a rather slow and tedious business and took something like 15 to 20 minutes to generate just 7 images (yawn). Then when I decided to see how well I could cope with this idea in assembly language, it being the first major piece of work I have attempted in this medium, I went back to the drawing board and worked on creating the other images whilst the first was being drawn and then the final images being reversed only on completion of the first. This all began something like three years ago and I finished the bulk of the source code then. I would have had the whole thing completed and debugged a long time ago but for the fact that in the meantime I have completed an A level in Computer Science, City and Guilds Cobol (Yawn again. Cobol is a bit like having an attack of verbal diorrhea, when you've grown up with Basic which is more concise and to the point. Unlike me!) and Systems Analysis whilst my wife has brought two children into the world. Its only recently that I've had a good bash at trying to debug the main animation routines, such as when the character moved from right to left all I seemed

to get was a screen full of garbage (this took me ages to figure out what was going wrong and it was something quite simple at the end of the day). Finally the technicolour version arrived and here it is.

Designer instructions

First of all you will need to load a short Basic programme. This asks whether you will want to design in multicolour mode or not. You will then be asked for the background and other colour(s). Next you will be asked for the offset factor for the character designer. The greater the value for this, the more extreme the movement you will get for the character you will design. A value of 0 will give a statue, but remember, to avoid overstepping the characters dimensions the programme will reduce the amount of drawing space available to you with a high offset factor which I have restricted to 2.5. The programme will then poke into memory the data needed to adjust the legs of your creation. Next you will have to repeat the operation for the arms (you will draw the main torso, head and legs in one go then add the arms afterwards). Again the data is poked into memory and finally you will get the chance to access the main design menu, which has six options.

Option 1)
This allows you to draw the main part of the body and legs and has slightly different key commands depending on whether you're in hires or multicolour mode.
In hires mode, pressing E will toggle the mode of drawing between erasing and drawing the foreground colour. The crosshair shows the point at which drawing will occur, this and the border will switch between red and green depending on this mode. Pressing C will clear the screen and character data.
In Multi colour mode pressing keys 1,2,3 and 0 will switch between colours and C will also clear the screen and character data.
Pressing RETURN in both modes will return you to the main menu (after a very short delay in which the character data is reversed.)
You should always draw your character facing from left to right as the programmes offsets are set to manipulate the data for figures facing this direction. On pressing RETURN the characters Hires Data are reversed. Actually this is not strictly true as in Multi-colour mode there are two bits to represent each pixel. Therefore with a normal reverse routine colour 1 would become colour 2 and vice versa. (Colours 0 and 3 would remain unchanged). There are two different routines to deal with two colour or multi colour modes, though you actually need not worry about this as they're called automatically depending on which one you're in.
Option 2)
This allows you to generate the arm movement of either one or both arms. You will probably want to create the effect of just seeing one arm. Key commands are the same for option 1.
Option 3)
Now is your chance to see what you've created. Move the joystick left and right to animate your character. Press the fire button to exit.
Option 4) and 5)
These allow you to save and load your design to either tape or disk. Type in T or D for device of your choice. You will only be allowed 10 characters for the filename. An extension of .ANM is added to the filename automatically in either load or save modes so that you can distinguish these files later.
Not only is the design saved but also whether it was multi colour and the colours originally picked by the user
Option 6)
Restarting the utility allows you to change the offset values for the different frames. However this will also clear the existing design in memory, so be careful.

Using Your Design From Basic

The animation routines can be run as an interrupt driven routine or directly by a loop routine from basic. Since the designer was written using a plot pixel routine I have given the basic programmer the facility to plot and unplot points on the hires screen.
Sys Routines and Poke or Peek locations.
ABASE:51290
Location holding Hires screen vector LO-BYTE. For those unfamiliar with HI and LO-BYTE vectors then you will need to realise that the Commodore uses sixteen bits to find any location in memory, which is two bytes. It is standard format to store vectors as LO byte, HI byte, in fact this is how the machine is wired up for indirect addressing routines in machine code. E.G. Location 49152 is $C000 in Hexadecimal or Base 16. Therefore to address this location from a two byte vector the lo byte would contain 0 i.e. 49152-(INT(49152.256)*256), and the hi byte 192 ($C0 in Hexadecimal) i.e. INT(49152)/256.
ABASE+1
Hires screen vector HI-BYTE
AER.49174
Routine to print standing figure (either facing to the left (RILE=1) or to the right (RILE=0).
ALOOP:49837
Routine to exclusive – or an 80*96 pixels block of memory from start address held in lo hi format at locations $FD and $FE (253 and 254 decimal) to $FB and $FC (251 and 252 decimal).
ANIM:49450
Interrupt driven animation routines. Both animation routines (this one and NANIM) will read the joystick port 2 automatically.
AOFF:49475
Routine to switch off interrupt driven character routine.
ASTE:49890
Routine to restore variables for ALOOP routine.
CB:254
Bottom of memory clear vector HI-BYTE
CLS:49235
Routine to set video matrix to start at $5C00.
CMEM:49246
Clear memory of video matrix to colours set by user during his design of character. This routine reads 49157 which hold the nybbles for the video matrix (now at $5C00) and 49158 for the colour ram ($D800 to $DBFF).
COLOUR:50982
When in four colour mode, setting this location to 0,1,2 or 3 will plot location in background colour or colours 1,2, and 3 respectively
CT:50236
Top of memory clear vector HI-BYTE
DX:49359
Location storing direction in x axis of joystick in port 2. (1 for right move-

ment. 255 for left movement)
DY:49360
Location storing direction in y axis of joystick in port 2. (Either 1 or 255)
ERASE:50981
When plotting pixels in two colour mode, setting this location to 1 will erase pixels, setting to 0 will draw pixels.
FIRE:49361
Location holding the value of 1 when joystick fire button pressed.
HL:49919
Extreme left hand side co-ordinates for Hires Character. HI-BYTE.
HR:49921
Extreme right hand side co-ordinates for Hires character. HI-BYTE.
HX:50983
Position in x axis to plot hires co-ordinate. LO-BYTE (i.e. 0-255). Note in 4 colour mode, the co-ordinates still run from 0 to 319 which the plot routine converts this to a two bit value.
HX+1
Position in x axis to plot hires co-ordinate. HI-BYTE (i.e. positions 256 to 320 minus 256)
HY:50985
Position in y-axis to plot hires co-ordinate.
INIT:49362
Routine to initialize a character at the left hand side of the screen. This can take a lot of pain out of setting up a character for animation.
LL:49918
Extreme left hand side co-ordinates for Hires character. LO-BYTE.
LR:49920
Extreme right hand side co-ordinates for Hires character. LO-BYTE.
MODE:49711
Location to hold RTS or JMP machine code instruction depending on whether routine is to be interrupt or non interrupt driven i.e. 96 for non interrupt or 108 for interrupt.
MULTI:49152
This location stores whether your hires character is in multicolour mode(1) or two colour (0). Poking this location with 1 or 0 will be used in the PLOT routine
MVE:49917
Location telling you whether the hires character is moving (1) or standing still (0).
NANIM:49490
Non interrupt animation routine. Can only be used after setting MODE to 96.
PAUSE:50192
Location to hold value PV to slow down animation routine (0=fastest, 255=fastest).
PLOT:50901
Routine to plot pixel at HX,HY and in colour COLOUR (4-colour mode) or ERASE (2-colour mode).
PV:16
Value held at location PAUSE to slow down animation routine.
PX:49924
Location holding number of character blocks moved by hires character.
PY:49925
Location holding number of character blocks moved down by hires character
RILE:49916
Location telling you whether the character is moving right to left (1) or left to right (0).
SBANK:49205
Routine to set bank that the VIC-II chip will rend to bank 1. Again for those unfamiliar with the banks used by the commodore 64, I will explain. The VIC-II chip can only "see" 16K at any one time. Therefore anything that the programmer wishes to fit on the screen including screen memory itself, sprites and character set must all be in that 16K block. (This is not quite true but for the purposes of brevity you will have to take it for granted.) The programmer therefore has the choice of 4 banks of memory to choose from of which the default is 0. Memory register $DD00 (56576), bits 0 and 1 are used to set the Bank and $D018 (53272) to set the screen (or video matrix) memory and the character dot data base (or hires pixel data)
SCHBASE:49224
Routine to set character base to $6000 which will be used for the hires screen.
SULUE:49912
Location of top left hand corner of character on screen LO-BYTE
SULUE+1
HI-BYTE of top left hand corner of character.
WPE:49297
Routine to clear memory from CB*256 to CT*256.
YD:49926
Switch on movement in y axis of hires character (0=off, 1=on).
SYS 50222 can be used to wipe out memory from $6000 to $BFFF.
HINTS and TIPS.
Memory map.
$C000 to $CE00. Main animation and design routines.
$A000 to $BFFF. Reversed character images.
$8000 to $9FFF. Hires character images.
$6000 to $7FFF. Hires Pixel memory.
$4000 to $4800. Tables to set offsets of character animation design.
$4000 to $5BFF. Sprite Definitions during animation.

For ease of programming when using Hires Animator it will probably be more convenient to set up the character using the SYS INIT routine. Alternatively you will have to set LL, LR, HL, HR, RILE (0), MVE (0), SVLUE and SVLUE+1, then call AER, PX and PY routines. See the demo routines to see how it's done.
ASTE, ALOOP, probably need not be used but I've left them there for anybody who wants to use them for more complex routines. Note that with the new bank now set to 1 and the video matrix at $5C00 then the sprite pointers start at location 24568 ($5FF8).

MODE is set for interrupts as a default and will only need to be changed when using NANIM. If your using RILE as part of an initializing routine from basic, then it can only be set to 0 as the reversed characters (i.e. those facing Right to Left) are stored under basic ROM and this needs to be switched out via a machine code routine.

The co-ordinates for the 16 characters created by the designer are $8000, $8050, $80A0, $80F0, $$9040, $9090, $90E0, $A0F0, $A0A0, $A050, $A000, $B130, $B0E0, $B090, $B040.

Finally for anyone interested in changing the animation effects created by the designer then the data lines in the DESIGN OFFSETS programme are set up as follows.

The values are in groups of four, and represent how each pixel from row 47 downwards on the leg and row 21 downwards for the arm are moved in each frame from the standing figure you draw, to create two arms and two legs. The first two are for one arm and leg (X and Y) and the second two for the second arm and leg. These offsets are not required after you have designed the character.

Rotatron

Bored? Fed up of your games? Then give yourself some relaxation, load up Rotatron, sit back and enjoy

A lot of people these days forget one all important factor when they switch on their computer. A home computer is essentially an entertainment machine. Oh yes, it's nice to sit there for hours and program your next blockbusting utility.

Maybe you stare at your screen for hours whilst you are working out your bank balances, or maybe you write thousands of letters on your word processor. All these activities are very rewarding to the dedicated computer user. However, they all suffer from the same side effects, that is to say, boredom sets in. All work and no play makes Jack a dull boy. Enter ROTATRON.

Rotatron demonstrates the use of pre-calculated rotation tables to produce some mathematically complex rotational effects.

It was written by me (Under the alias Thunderdog,) in 1989, and contains handling for all parameters: major, minor and tertiary orbits.

Major orbits are like Earth's orbit around the sun, in that they are the primary direction in which the sprites move. Minor orbits are like the moon's orbit round the earth, in that this is where the sprite rotates around that axis. And tertiary orbits are like a satellite's path around the moon, in that they orbit round the minor orbit.

For the mathematically minded, here are the formulae to work out a sprites' position at any time, given that A_1 to A_3 are the angles (from 360°) of major, minor and tertiary in x, and A_4 to A_6 are the angles of major, minor and tertiary in y.

$$A_1 + 30 \cos A_2 - 15 \cos A_3 + 160$$

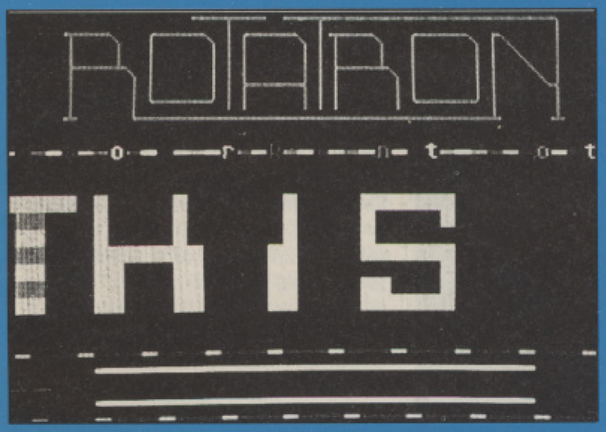
$$y=100 \sin A_4 + 30 \sin A_5 - 15 \sin A_6 + 160$$

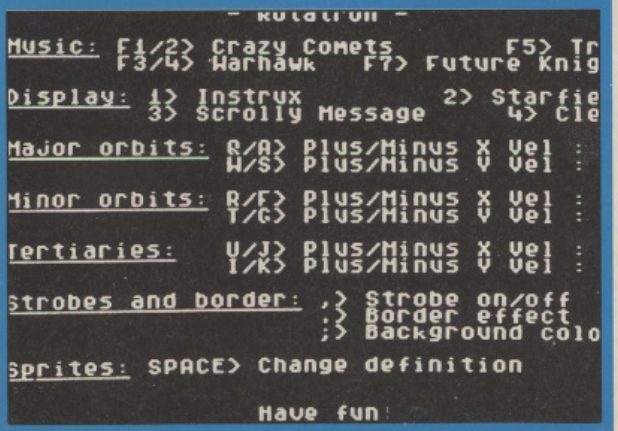
For those not of a mathematical nature, all you need do is remember what I explained about the orbits before!

To load the demo, use the CDU menu, or type:

```
LOAD "ROTATRON",8
RUN
```

Once the title screen appears, you can press SPACE to start the demo. After that, the keys are displayed on a help screen, along with the current rotation speeds, which can be accessed at any time by pressing "1". (Except on the title screen!)





Maze Generator

If you enjoy mazes but would prefer to construct your own then this program is for you

By C. Makepeace

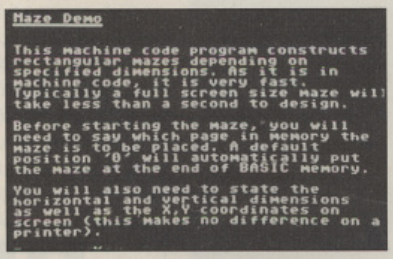
If you enjoy solving mazes, this will allow you to make a limitless amount of your own. The mazes are designed in machine code and are thus very quickly done. You can have a maze of any dimensions designed from tiny 2x2 to huge 63x255 mazes which will keep you stumped for hours!

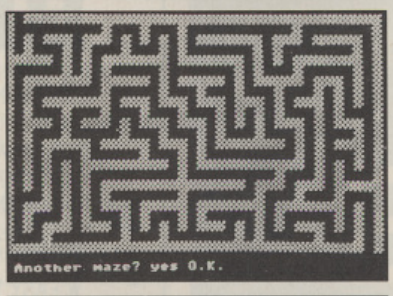
The mazes start at the top-left of the maze and end at the bottom-right. The way the mazes are designed is such that there is only one possible route through. In fact, between any two points, there is only a single route.

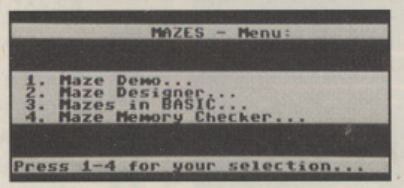
Once the computer has finished the mazes, you can either print them on the screen or to the printer. If your printer has this facility, you may condense the characters to subscript size and print very dense mazes.

Everything you will need to know about using the maze program is contained on the disk in the form of a computer tutorial which you run and experiment with different sizes of mazes. Within this tutorial there is also a short game where you have to guide a little blob around the maze and collect five diamonds randomly scattered about. Once these have all been collected, you must make for the exit at the bottom-right of the screen. If you wish, you can have another blob moving around the maze which you must avoid.

How the mazes are created: On the disk is a BASIC equivalent of the maze designer which may help you understand how the mazes are constructed. The array which contains the data for printing out the final maze is stored in M%(H,V). The computer moves randomly around this array making a path through it and as it does so, it keeps a 'logbook' (in the array P%(P)) of all its moves using the numbers 1 to 4 to represent Up, Down, Left and Right. Inevitably the computer will run into a dead-end where it can to nowhere. When this happens, it uses the logbook in P%(P) to move back along its own path until it finds part of the maze it has not been to before. The computer does this until all the array (M%(H,V)) has been used up. When this happens, the computer will come to its last dead end, then back-track all the way to the start, upon which it will print out the resulting maze. The key to only having one path is that the computer never doubles back on itself thus making a loop within the maze.







Technical note: Within the tutorial, the term 'page' is used. The C64 has, as you know, 64K of memory. This memory is split up into 256 'pages' each 256 bytes long. For instance, the screen memory starts on page 4 hence: POKE 4*256,1 will put an 'A' at the top-left of the screen.

So, if you wish to have the maze memory start anywhere other than the start of BASIC (which is most convenient), you will have to decide which page you are going to put it on. Avoid the areas above page 160 (hex: $A0) as this is where the BASIC ROM is stored and the program won't work. Also avoid any pages below page 8 as these contain information which will crash the computer if changed.

How to calculate how much memory your maze is going to use: The amount of memory used depends mostly on the horizontal and vertical dimensions, the larger they are, the more memory is used. To find out how much, the horizontal dimension must fit into one of these categories:

0-63, 0-31, 0-15, 0-7, 0-3, or 0-1

i.e. a 28x10 maze would be (to the computer) a 31x10 maze and a 8x3 maze would be a 15x3 maze. Once you have the computer-dimensions, add 1 to both, find the product and then round up to the nearest 256 bytes (i.e. 400 goes to 512 and 750 goes to 768). This will tell you how many bytes the maze will use.

Some examples:

A 20x10 maze:
This fits into the 0-31 category so:

(31+1)*(10+1) = 32*11 = 352 bytes.
352 bytes to the nearest 256 gives 512 bytes or 2 pages.

The biggest possible is a 63*255:

(63+1)*(255+1) = 64 * 256 + 16384 (64 pages)

This is a huge maze (about 4 A4 pages of compressed subscript!) hence 16K of memory being used. (If a BASIC equivalent of the maze designer were to be used, it would require about at least the full 64K of memory!).

A word of caution: if you intend to design large mazes, a lot of memory may be required. To allow for this, do not have a large BASIC program in memory as there may not be enough space. If the computer accidentally starts to write over $A000 (the ROM) then the end of the maze will be corrupted or the computer may crash. If in any doubt, work out how much memory is to be used and then check whether it will fit in. Also on the disk is a program to calculate the amount of memory to be used by a maze, filed as MAZE.MEMORYCHK.

Character Extractor

A simple but powerful utility that will scan the whole of memory for character sets, and allow you to save them to disk

By Neil Higgins

This is a simple but powerful utility that will scan the whole of memory for character sets, and allow you to save them to disk. Although I do not expect you to go around pinching every character set you can find, it can be used to see how certain fonts and backgrounds have been defined for idea's in designing your own graphics.

Since the extractor needs to view all the memory I have supplied two versions, one sits in low memory from: 4096-6163 ($1000-$17FF) and the other in high memory from:36864-38911 ($9000-$97FF). You will of course only need to load one version at a time, but if you cannot find a particular set, lets say using the low version, then you should re-load the program being scanned, followed by the high version, that way you can be sure of scanning the whole of memory. Both versions transfer the character set into the default Bank 0, so that no memory is destroyed using bank switching and moving the screen, the only area of memory that cannot be seen is from 0-2047 ($0000-$07FF) which contains zero page and screen memory, it is very rare that a character set is placed here anyway. Experienced programmers will have noticed that both versions sit in an area which is a character rom image, which means there is less chance that they will overwrite any character sets, even so I have still allowed you to look at three areas because there may be sets just being stored.

So the only character sets that cannot be seen by each version are as follows:-

LOW VERSION
$0000-$07FF = Zero page/screen
$3000-$37FF = Character Set store
$1000-$17FF = Extractor Code

HIGH VERSION
$0000-$07FF = Zero page/Screen
$3800-$3FFF = Character Set Store
$9000-$97FF = Extractor Code

When you have loaded and started one of the extractors you will see in the top half of the screen all available key actions, while the middle of the screen displays the character set with its original address in decimal and hex. All key actions are exactly the same in both versions and do the following:-

F1 = Multicolour #1 ($D022)
F3 = Multicolour #2 ($D023)
F5 = Screen Colour ($D021)
F7 = Character Colour
F8 = Exit to Basic
M = Toggle Multicolour On/Off
+ = Next Character Set
– = Previous Character Set
S = Save Current Set
D = Disk Directory

If you exit to Basic by pressing F8 the extractor will still be in memory and can be restarted using the appropriate SYS command. All disk operations are to device 8, if you want to save the current set then press S then enter a filename and press return.

LOADING THE EXTRACTORS
First of all load in and RUN a program you wish to look at, now reset the computer (a reset button is provided on most cartridges) finally load one of the character extractors with the following:-

LOW VERSION
Enter LOAD "CHAREX/HIGH",8,1 (Return) then start it with SYS 36864 (Return)

Alternatively, select one from the disk menu. So that no memory is corrupted, it would be wise to load them straight from the disk as above, instead of using the menu.

Flexible Line Distancing explained, in laymans terms

By Jason Finch

Nudge

SCREEN SLIDER, published in the March 1990 issue of CDU, was an introductory utility to test the feasibility of a program such as NUDGE, my latest offering to readers of CDU.

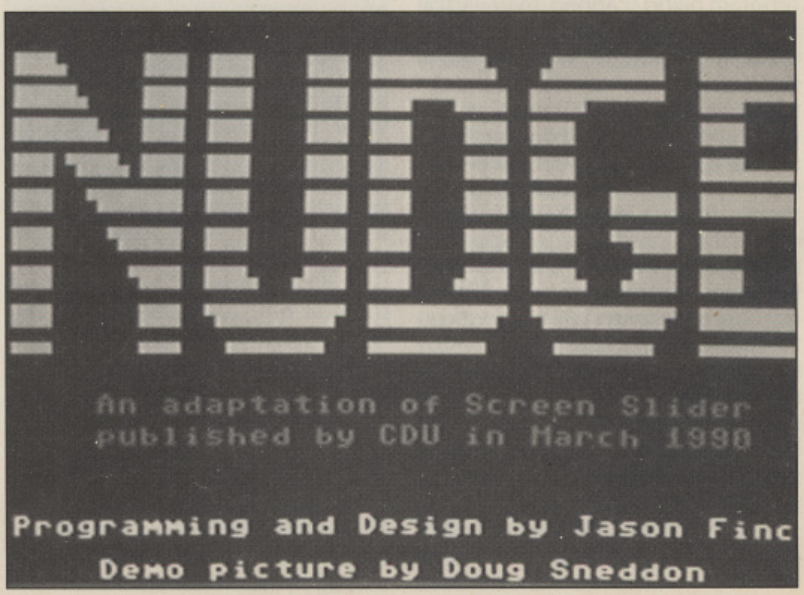
Like SCREEN SLIDER it relies upon the development of a technique known as Flexible Line Distancing. However, the main section of Nudge involves much more complex programming methods and, unlike the former utility, requires extremely precise timings. If the routine is slowed down by just one timing cycle (very fast indeed – in fact, approximately one millionth of a second. There's a little proof of this later!) then the display will flicker and the slide will not work. For this reason the main keyboard is not scanned whilst the slide is in progress and control is not returned to BASIC until it is complete. It is also necessary to disable sprites whilst the interrupts that make the effect possible are in operation.

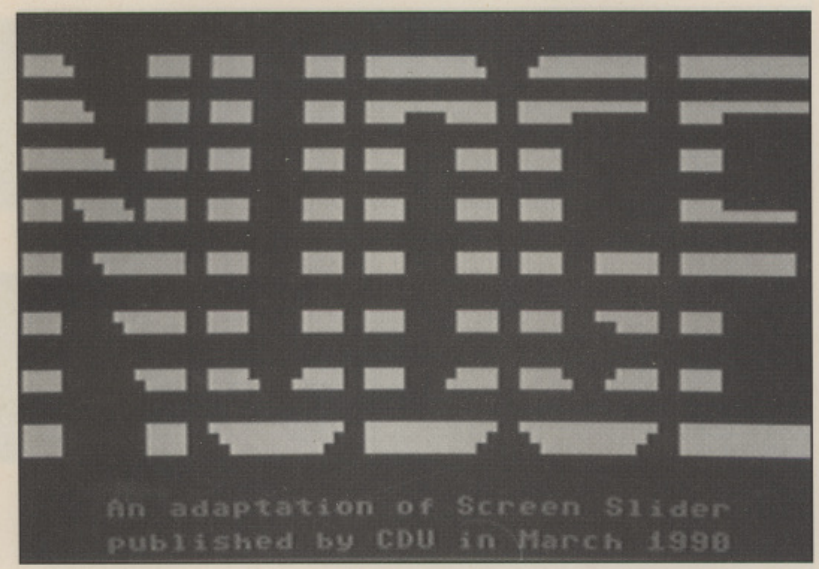
The slides must first be prepared for in the same way as those in SCREEN SLIDER. This allows a flicker-free display prior to an upward scroll but is also convenient for a downward scroll. You must bear in mind that with the former, the screen that is about to be scrolled is theoretically visible in its entirety and the sections that you cannot see are merely positioned out of the display window.

You must set up the computer for the scroll using one of the three simple commands that will be described later. It is then that you display your screen however you would as if it were to simply appear. This may be by using the standard PRINT command or POKEing to the screen. You would also change to multicolour, bitmap or whatever mode you wished to use at this point. If you plan to change location 53265 then you should read carefully the section later on, that discusses this. You would then use one of the other commands to initiate the scrolling – the computer will return to BASIC the moment that it finishes. Despite there being only three commands the routine is amazingly versatile, allowing you to change not only the direction and speed of the slides but also what will happen afterwards and the spaces between each line. Now we get to the point of Nudge and so before explaining the operating instructions a few words on what the routine actually does and what you should see on the screen.

As you will know (hopefully), the screen consists of 25 rows of 40 characters. Each row is made up of eight pixels vertically, creating an overall height of 200 pixels. A line scans the display many times a second to build up the picture. This is called the raster line and can be intercepted at any position by machine code routines to change the screen display at any vertical position. This is how affects such as split-colour borders are achieved and more recently programmers are discovering increasingly impressive effects that can be created. 'never mind if you don't understand machine code – follow my simple instructions and you can't go far wrong.

I am sure you will have seen games that scroll screens vertically very smoothly. These do not use the same technique – Nudge allows one screen to be cleared from the display or one to enter it a little differently to the norm, and completely independently of colour, mode and bank number. Each of the 25 lines of the screen can be introduced or made to fall away separately at a designated speed, or larger sections of the screen can be moved together. To understand this better you should see the





demonstration program. To help you a bit first, I have dreamed up an appropriate analogy.

Imagine twenty-five people lined up along a wall, all facing one way – at ninety degrees to you who is looking through a window with only the people and the wall visible. Each person represents one line of the screen and the wall the background, although we have to rotate the whole situation through ninety degrees. Each person in turn waits a short while before walking off out of your field of view, creating a gap between each person. When all have gone you will not see the people although they all still exist – all you see is the background. You can instruct these people as to how fast they should walk, the length of the 24 gaps between them and what they should do afterwards – should they all return to their original positions or should they remain out of view. These could then be replaced by 25 different people who could then run back into your field of vision at a different speed, with different spacings and so on. You should now be prepared for the operating instructions.

The code on the CDU disk, filed as "N.CODE", loads to 49152 in memory, its end address being 50007. The various operations are performed via three SYS calls from BASIC. These are as follows: SYS49330,a,d will prepare the interrupts for a scroll in direction 'd' with 'a' representing what will happen afterwards. Each one must be given a numeric value and if both are zero then all interrupts will be switched off. This should be done before you access the disk drive or cassette deck. The same operation will be performed after the proposed scroll if 'a' takes a value of zero. If 'a' is a one then the interrupts will remain enabled. This means that screens about to be scrolled upward can be forced not to appear for a split-second before they are actually scrolled. Finally, 'd' must be a one for an upward scroll and a two for a downward scroll. In the former case the display will apparently disappear. 'It can be thought of as being positioned below the display area. `in the latter case there will be no apparent change.

SYS49333,x will start the slide in the direction specified by the SYS49330 instruction and with a speed represented by 'x'. This is a numeric value between zero and nine inclusive with nine being the fastest. 'If you have not prepared the interrupts beforehand with the SYS49330 call then a Syntax Error will be generated. Otherwise, all sprites are disabled and control is not returned to BASIC until the slide has finished.

SYS49336,n1,n2...n23, n24 will set the spacings between the lines as they are scrolled into our out of the display for the next and subsequent scrolls. They represent the actual number of physical screen lines (sets of eight pixels). If 'n1' takes a value of 255 then the computer will use the default set of values and the other numbers can be omitted. The only limitations with this command are that 'n24' must be a non-zero value and the total n1+n2+...+n23+n24 must be eight or more. The lowest value any one number can be is zero and the highest is 26, one more than the total number of lines on the screen. If any of these rules are not adhered to then an 'illegal Quantity Error will be generated by the routines. The only other limitation, of course, is the physical length of a BASIC line – 80 characters.

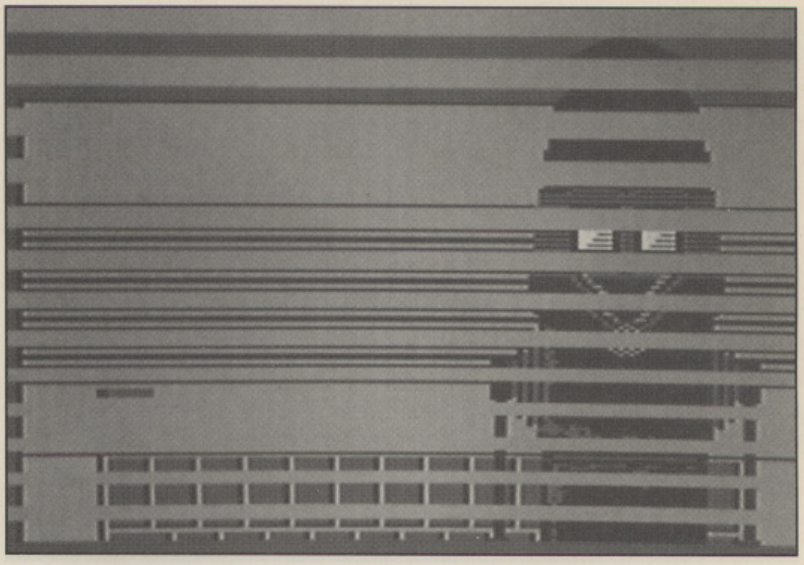
One other thing that you should be aware of is related to the way in which the VIC-II chip works when shifting the screen vertically by one pixel. So far as graphics are concerned, the memory is divided into four blocks of 16K. On power-up the computer uses block zero which occupies locations zero through to 16383 inclusive. When you are using the routine, unless the last address in the block of memory that is being used is a zero, black lines will appear in between lines as they are scrolled. Therefore you should ensure that location 16383 holds a value of zero (POKE16383,0). However, with this routine 'i have included this in the code and so when a scroll is prepared and again when it is initiated the computer checks to see what block of 16K is being used and stores a value of zero in the appropriate location automatically. This should not create any problems unless you have a large BASIC program that would usually use location 16383. The only other address that may need to be altered by you is 1536 bytes "earlier" in memory than that needed usually. For block zero this is memory address 14847 ($39FF). This will only need to be altered if you want to use extended background mode and is not altered automatically by the code – you will need to do it yourself.

The last point is to do with location 53265. This is the location that controls bitmap mode, 24/25 rows and is the location around which the whole method for shifting the screen in the way that Nudge does is based. When a scroll is prepared with the SYS49330 command the contents of bits 3-6 of location 53265 are stored into location 49453. Any attempt after that to change location 53265, for example to display a bitmap picture, will be pointless. Once a scroll has been

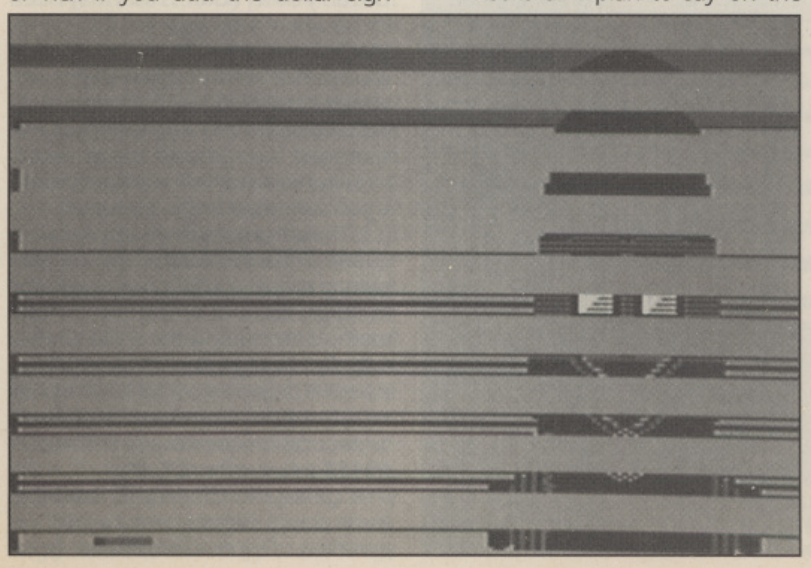
prepared you must alter location 49453 and not 53265. You should store only bits 3-6: POKE49453, (X AND 120) where X is your desired value for 53265. You can see this in action when the bitmap picture is displayed and extended background mode used in the Nudge demonstration that is on the disk (see lines 45 onwards).

Now more about the demonstration program. To load it, select the program NUDGE from the menu. The code will then be loaded together with some graphics created by a friend of mine, Doug Sneddon. The demo is a simple BASIC program that illustrates what can be done using Nudge. Also on the disk is a file called "N.SAVE-REL" which will not only allow you to save the main code to your own disk but it will also give you the option of relocating the code. You are somewhat limited to the number of places to which you can relocate it because of branch instructions in the machine code. If one has to cross a certain boundary then an extra timing cycle is required and the routine is thrown out of sync.

In the memory of the Commodore 64 there are 256 blocks of 256 bytes of memory. Each of these is called a page. Page zero is locations 0-255, page one is 256-511 and so on. You can only relocate code with the start address being the first in a page. The computer will ask you for the start address of the code or the page number which can be given in decimal or hex if you add the dollar sign prefix. A number greater than 255 will tell the computer that you wish that to be the start location and a number less than that will signify that this is your desired page number. The code can be situated in the BASIC programming area or in the 4K at 49152 onwards. The only restriction is that the start address is equivalent to the start of a page. The call address (usually 49330) will be displayed and you will be asked to confirm your selection. The code will then be saved to disk device number eight. If you only want to save the code and do not want to relocate it then simply type 49152 for the start address or 192 for the page number.



That is all I plan to say on the operation and theory behind the slides. Just load up the demo and list it if you like to see the methods used to achieve the different effects. But more importantly, experiment with it yourself and find a use for it in your own programs!



I bet you thought that I had forgotten my proof of the speed of one memory cycle. Well, although it doesn't really have anything to do with the program, here goes! You will need a monitor for this – use the fill instruction to fill the memory from $0810 to $9800 with $EA bytes (F 0810 9800 EA). This is the machine code NOP instruction and takes 2 cycles to perform. Return to BASIC and type POKE38911,96. This will merely allow the computer to return from the machine code routine. Now type the following line:

```
TI$"000000":SYS2064:PRINT TI
```

The computer will pause for a short while whilst the 36847 instructions are carried out. Each takes 2 cycles – a total of 73694 memory cycles. The value contained in the variable TI is the number of 60ths of a second that it took. You should get a value of five or six. Divide 73694 by your answer and then multiply by sixty to obtain the approximate number of cycles in one second. You should get a value in the region of about three quarters of a million cycles in every second. Not quite one million – but don't forget the BASIC instructions in between!

Techno Info

The sacks at CDU HQ are bulging at the seams, here's a few examples

Dear CDU,
I am now hooked on CDU after buying the last couple of mags. It's great value, informative and the disks are fantastic, along with the programs and the routines. But I have a problem! I have been trying without success to write a routine to put on the front of my own programs and games to make the coloured flashing border when my games and routines are loading. Please could you help by putting a routine on the disk or by printing a listing to type in.
David Lomax, Merseyside.

Dear David,
Thank you for the kind comments about CDU. The solution lies in the Test-STOP vector at locations 808 and 809 ($0328/$0329). What you need to do is write a short machine code routine to change the border colour or simply increment it to cycle through all the colours. The Test-STOP vector then has to be changed to point to your routine. It usually points to $F6ED, the values in 808 and 809 being 237 and 246. If you do program in machine code then remember that your routine must end with a JMP $F6ED instruction and not an RTS. However, in case you do not program in machine code I was going to provide the necessary listing but I have gone one setup further and so on this issue's disk you will find a program filed as "PROB1". What it does is create a file, which is saved as "LOADER". This is a one block machine code program and once you have supplied a name for a program that you want it to load, you will be able to type LOAD "LOADER",8,1. The autobooting program will then almost immediately initialise the flashing border routine and then continue to load and then RUN your program, after having first cleared the screen. So, load up and RUN the file PROB1 and then follow the on-screen instructions. As with everything there is one minor point about which you should know. Any BASIC program that is loaded by the loader MUST start with a CLR command if you want it to run correctly. Simply add it to the start of the first line or create a new one. It really isn't too much to ask, is it?! I hope you find the routine useful.

Dear CDU,
I have purchased some books but I cannot find anything on scrolling a map which I could use in my program. I enclose an idea for a character editor and scrolling map that I would like to use, leaving the bottom four lines of the screen free for messages and so on. If you know where I could purchase the above or how about giving the program in your magazine? I hope you can assist.
Mr H. Field, Kent.

Dear Mr. Field,
You do not mention whether you plan on this being one of the super-smooth sort of scrolling maps or just a rough scrolling one. I shall therefore take the easier of the two options – the rough scroll. This does not rely on shifting the screen and "raster splits" and is therefore less complex. On this issue's disk you will find a machine code program filed a s"PROB2". Once loaded (with the ,8,1 suffix) you will have three "commands" at your disposal. They will allow you to shift the map about, read in the character and colour at a specified position and finally, display a specified part of the overall map, which you mentioned would be eighty characters horizontally by forty characters vertically. You did not state whether your map would have different colours but I have allowed for this just in case. All of the information for the map is stored under the interpreter ROM, thereby disrupting no BASIC memory, with the character information (in POKE codes) starting at 40960 ($A000) and the corresponding colour data at 45056 ($B000) onwards. The rou-

tines are activated with BASIC SYS-calls. The first being SYS49152,n where 'n' represents the direction in which the shift will occur. It takes the values zero, one, two or three, thereby shifting the visible area up, down, left and right respectively. The routine has its own checks to see whether the boundaries have been reached. The second routine simply returns the character number and colour of a specified location in the larger map and takes the form SYS49155,x,y. 'x' is in the range 0-79 and 'y' in the range 0-39. The number and colour can then be read by PEEking locations 251 and 252. The last routine is called with SYS49158,x,y and will display the map with 'x' and 'y' being the top left corner's co-ordinates. Your original specification that four lines should remain clear at the bottom has been kept. Also on the disk is a demonstration of the routines in action, which is filed as "PROB2.1". I have compressed it so that it occupies less disk space but once it has 'decompressed' itself and is running, simply press the RUN/STOP key. Then you can have a look at the BASIC program and you should have no problems using the routines in your editor program. Incidently the 'x' and 'y' co-ordinates of the last part of the map displayed are stored at 49161 and 49162 respectively and represent the top left hand corer. To store new sections of the map use the formula ADDRESS=BASE+Y*80+X where BASE is either 40960 or 45056 and X and Y are the co-ordinates in the necessary ranges. To save the map you will have to switch out the interpreter ROM. If this poses any problems you can change the locations of the character and colour information by POKE49270, A: POKE 49339, A: POKE 49289, B: POKE 49357, B where A is the character info start divided by 256 and B is the start of the colour information divided by 256. The present values are 160 and 176. Or if you don't want to do that, then wait for next month's Techno Tip, when I plan to present a short machine code save routine that caters for memory under the interpreter ROM. For now, though, I hope you find the routines a help.

Dear CDU,
Having read the article on the Power Cartridge in January's CDU, I sent off for one. When it arrived it went straight into the 64 – the only problem was that I found it to be incompatible with the older type of Commodore 64. I came to this conclusion after trying it on several other computers belonging to friends. The problem is that the computer locks up when you try to reset it, thereby making the cartridge useless. I have telephoned BDL twice although nobody seems to know why it is not compatible with the older 64. I would like to know if anybody else has had the same problem and whether it can be rectified.
Mr A.Booth, Bristol.

Dear Mr. Booth,
We have not had any other queries about the compatibility of the cartridge. It could be that yours has a faulty chip, unless of course all Power Cartridges suffer from this problem and I would therefore like to hear from anybody else who has spotted a similar fault and shall report back in a future issue if that is the case. I cannot provide any information on how to rectify the problem and I can therefore only suggest that you send your cartridge back to BDL or simply buy a different type of cartridge – perhaps the Expert or one of the Super Snap-Shot or Action Replay series, to name but a few. Sorry that I cannot be of any more help.

Dear CDU,
I am inquiring into a few aspects of the GEOS package. My version is quite old now, being V1.3, and the problems could have been corrected. Firstly, is there a printer driver available for the Citizen 120D. Currently I am using the MPS-801 driver. Secondly, is it possible to print the pages of the documents separately. Using the MPS-801 driver the whole document is printed all at once. Thirdly, how difficult is it to write GEOS programs and finally, could you recommend any good compilers. I have the Laser Compiler from Ocean IQ but it rarely works, and when it does, the code it produces is as slow as BASIC. I had heard that Petspeed was the best but I thought it went out of production many years ago, until, that is, I saw an advert for it!
Adam Trickett, Leeds.

Dear Adam,
There is no printer driver available specifically for the Citizen 120D. However, if you have it connected through the serial port then you should use the MPS-1200 printer driver although you may need to change the setting of the line feed DIP switch. With regard to your second query, I would suggest that you fork out and purchase the geoWriters Workshop from FSSL (address later). This is version 2.1 and is a great improvement on V1.3. It allows much more control over your document, including a better printer option that allows you to select the start and end pages for printing. You can set these the same, thereby printing just the one sheet. I have no experience of attempting to program a GEOS application although there are two commercially available products that assist. They are Becker BASIC 64 and geoProgrammer 64, both available through FSSL although quite expensive – near enough forty and fifty pounds respectively. Becker BASIC adds over 270 commands and allows you to write your own GEOS applications in a form of BASIC. GeoProgrammer is a complete assembly language development package for GEOS. I would recommend that you write to FSSL, requesting their latest catalogue which provides further information on both these products. The address is FSSL, Masons Ryde, Defford Road, Pershore, Worcestershire, WR10 1AZ. On to the compilers. I personally own the Blitz compiler from SuperSoft although Petspeed is a far superior package and I would recommend that you go about purchasing that package. You could use the C-Zap compiler published in CDU quite a while back (March/April 1988) or if you are adventurous you could buy the book 'Compiler Design and Implementation on the 64 and 128", again from FSSL. It has 280 pages of information on writing your own and includes a working compiler. It costs £12.95 and there is an optional program disk containing the programs listed in the book, costing £7.95. Pleasant purchasing!!

Dear CDU,
I have been reading your magazine since issue one and I find it very good. The reason for my writing is to make contact with other C64 users. I live in Belgium and it is very difficult to get programs and so on for the C64. I would like you to publish my letter so that any English users could contact me. My address is as follows: Van den Driessche Jos, Lakborslei 78, 2100 Deurne/ Antwerp, Belgium. Many thanks.
Jos Van den Driessche, Belgium.

Dear Jos,
It is always nice to receive letters from countries other than England. There really isn't much else for me to say on this issue other than I hope that many CDU readers will take up your invitation.

Dear CDU,
I have recently upgraded from a C64 to a 128 with Oceanic disk drive and an MPS-801 printer coupled to the old faithful portable colour television. As you can appreciate the 80 column mode of the 128 is unavailable whilst using a standard television. I have recently been offered a monochrome monitor with the specifications provided and my query is, can this monitor be used with a Commodore 128 in the 80 column mode and if so what cables and connections are required?
Mr R. Grundy, Doncaster.

Dear Mr. Grundy,
It certainly is annoying that standard televisions can't support these 80 column modes. But from the specifications you have provided I can solve the problem. On the back of the 128, immediately to the left of the user port is a nine pin socket resembling a joystick port. This is labelled with RGBI which stands for Red/Green/Blue/Intensity. That in itself doesn't mean much but if you connect your monitor to that then you should be able to obtain your 80 column mode. However there is one drawback – you won't have 40 columns any more! What you really need to do is have your monitor and television set side by side and then use each one as required. Or on the other hand fork out a large sum of money and buy a dual monitor with a switch. I am afraid that I have just upgraded as well and that's what I shall have to do do. But less of my problems – this is the readers' page! I hope I have been of some assistance.

Dear CDU,
Thank you for printing my letter in the February issue of CDU regarding the problem I had with my STAR LC10 Colour Printer. Although I much appreciated your comments, I had already been in contact with FSSL and acquired the package that you recommended. You may also be interested to know that GEOS V2.0 will also produce colour on the Commodore 64 and STAR LC10C setup so long as DIP switch number one is in the off position. After eight letters and four telephone calls to various people the only help that got me printing was given by yourselves and Mr Tim Harris at FSSL. I have included a few printouts for you to judge the results of the STAR LC10C. Thanks again and please keep up the high standard of CDU.
Mr C. Best, Nottingham.

Dear Mr Best,
Thank you for replaying to CDU with the good news. I am sorry that it was a little too late but it is always great when things work out. Thanks also for the information about GEOS. I am sure that a lot of people will be rushing to their computers to try that one!

Erratum

I thought I would take this opportunity to inform you of a couple of small errors in my 'Cheats, Pokes and all that!' feature in the March 1990 issue – not my fault, I hasten to add. I gave a little program to give you an "auto-pilot" mode on the game 'Quad'. Unfortunately line 80 contained two mistakes. The number 249 should be inserted between the 41 and 141 and the number 77 near the end should in fact be a 72. The complete line should therefore read:
80 DATA208, 41, 249, 141, 16, 20, 173, 169, 72, 96
Your ball may disappear occasionally but it will return – that's due to my forgetting the original numbers that I put in and doing the unpardonable thing of erasing my word-processor backup and throwing away the thousands of jotter pads on which I scribbled my findings!

Tip of the Month

This month's dose of help comes in the form of a very short machine code routine, well one hundred bytes to be precise. It will allow you to display the director of a disk from within a BASIC program or your own machine code one. To use it you must first store the device number into location two in memory. Usually this device number is eight. The code resides at 49152 but using a machine code monitor you can relocate it to practically anywhere in memory without making any changes to the actual code. To do this most monitors require the format "T C000 C064 xxxx" where xxxx represents the address to which you want the code to be transferred, in hexadecimal of course!

To operate the routine from BASIC, enter POKE2,d: SYS49152 (where 'd' is the device number) or from machine code give the instructions LDA #d, STA $02, JSR $C000. If the desired device is present then the directory will be listed to the screen in the usual fashion and your BASICA or machine code program will resume as soon as it is complete.

An experienced machine language programmer should have no trouble in alterating the code so that it will dump to printer or store the directory contents for later use. The jump to $FFD2 outputs a character so simply change it to store that character somewhere in the memory. To load the routine from this issue's disk type LOAD"TECHNO TIP",8,1 and then, unless it is loaded from within a program, type NEW.

That wraps it up for this month but if you have any programming wisdom to share with the rest of our readers then please write to Techno Info, CDU, Argus House, Boundary Way, Hemel Hempstead, Herts, HP2 7ST. That is also the address to which you should send any queries or details of programming problems that you are experiencing.

Super Snapshot V5

The cartridge gets an uplift. Is it worth forking out for the extras? We investigate on your behalf

By S.Wickham

The controversy that surrounds Cartridge Utilities will be with us for many years to come. Apart from the obvious legal questions there are a number of moral and ethical ones appertaining to the use of such utilities. Whether you are a great believer in their role or whether you think they should be banned is a matter of personal preference. One thing cannot be denied though, and that is that they give the dedicated programmer some very useful ammunition in their arsenal of programming aids.

Of all the cartridges on the market, the one that does not seem to get much publicity, yet in my opinion is one of the best, is Super Snapshot from LMS TECHNOLOGIES of Canada. This little piece of wizardry gets an uplift in the form of SUPER SNAPSHOT V5.

Choices galore!

Any utility must, if it is to succeed, offer the user as many options as is possible. Bearing this in mind, SUPER SNAPSHOT V5 cannot be faulted. Indeed, if the number of options available is the main criteria of a utilities importance, then this cartridge is streets ahead of anything else around. Just look at this list of possible options:

Disk Copier(s)
File Copier
Parameter Copier
DOS Support
Boot Sector Support
Turbo DOS
Screen Copy (With Sprites)
Games Monitor
Machine Code Monitor
Track and Sector Editor
Drive Monitor
Video RAM Monitor
REU Monitor
Sprite Monitor
Sound Sample Monitor
Character Set Monitor
File Reader
Extra Basic Keywords
1571 Support
BBS Support
Cartridge RAM Expansion

Even the most sceptical amongst us has to agree that this is one heck of a list. It is fair to say that some of these facilities are accessible from the supporting system disk. However, unlike the other products available, you do not have to program SUPER SNAPSHOT V5 before you can use it.

Select but a few

I always wish that I could show every feature of a utility like this one. Unfortunately, space never allows my indulgence, therefore I have selected

just a few of what in my opinion are the more important aspects of the cartridge.

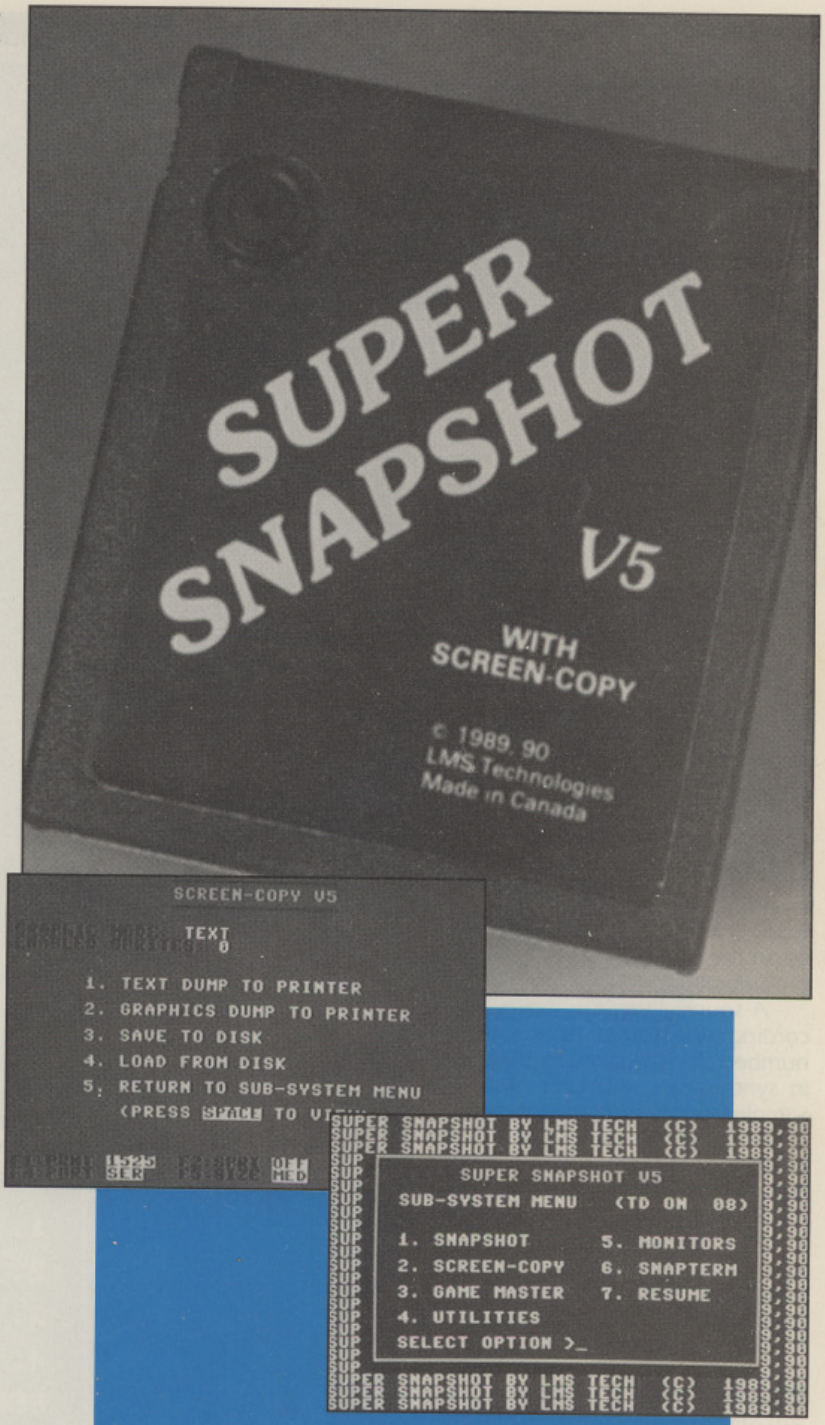
GetitMonitored

The nucleus of any good cartridge has got to be its ability to monitor what is happening inside the computers memory. Not only to monitor it, but to alter and amend it as you so desire. SUPER SNAPSHOT M/C Monitor, next a Monitor for Sprites, Monitor for Characters, Monitor for Sound. The Drives internal memory can be Monitored, as can the REU (Ram Expansion Unit) and Video RAM.

Those of you that have read my reviews before will know that my favourite utility of any cartridge has always been the M/C Monitor. The built in transparent monitor on this cartridge is excellent. Teaching machine code is not the intention of this review, therefore I will not attempt it. Suffice to say that if you examine the following table you will be impressed by the commands available to you.

M/L Monitor Commands

A	Assemble Code
BR	Set Break point
C	Compare Memory
D	Disassemble Memory (Sadly lacking on my Dolphin DOS)
F	Fill Memory
G	Go (to and execute)
H	Hunt through memory (Hex, Dec or ASCII)
I	Interpret Memory
IO	Display I/O registers
L	Load File
M	Display Memory
O	Output (Screen, Drive or Printer)
R	Display Registers
S	Save File
SP	Disable Sprite Collision
SPB	Disable Sprite to Background Collision
SPS	Disable Sprite to Sprite Collision
T	Transfer Memory
X	Exit the Monitor (The way you entered monitor)
XM	Exit to Sub-Menu System
:	Modify Memory
;	Modify Registers
,	Modify Disassemble
#	Hex to Decimal Conversion
#+	Decimal to Hex Conversion



+	Enable Decimal Entry
$	Disk Directory
*****	I/O Modify
@	Read Error Channel
@#n	Set Device Default
***Rn**	Sets Bank in REU
***V**	Accesses the C128 Video RAM

...ally the same as the above. All one needs to do to access it is to put a *n (where 'n' signifies device number) in front of the command. The drive monitor is obviously very useful for transferring the contents of the buffers into the computers memory, where you can examine, modify and then replace

them back into the drives memory.

Graphically Speaking

Sprite designing, like Character designing has always been a laboriously long job, even for those of us that think we are ok at it. No matter how proficient you are, there are no real quick methods. There are, however, ways of making this task a little easier. One of these is of course to 'pinch' someone elses ideas. (Don't forget, you cannot pinch the design and incorporate them in your own commercially available programs). The Sprite and Character Monitors come to your aid. With these facilities you can examine, modify, add to and generally play around with any Sprite or Character you like. The on-screen representation of the creations you are working on, is clear and full of the necessary information.

Sounds Great!

What surely must be a first from LMS Technologies is the Sample Monitor. I have to admit that I haven't come across one before. I also have to admit, that if there is one field of computer useage I fall down in, it's Sound and Music. I know absolutely nothing at all on the subject. So what exactly is the Sample Monitor.

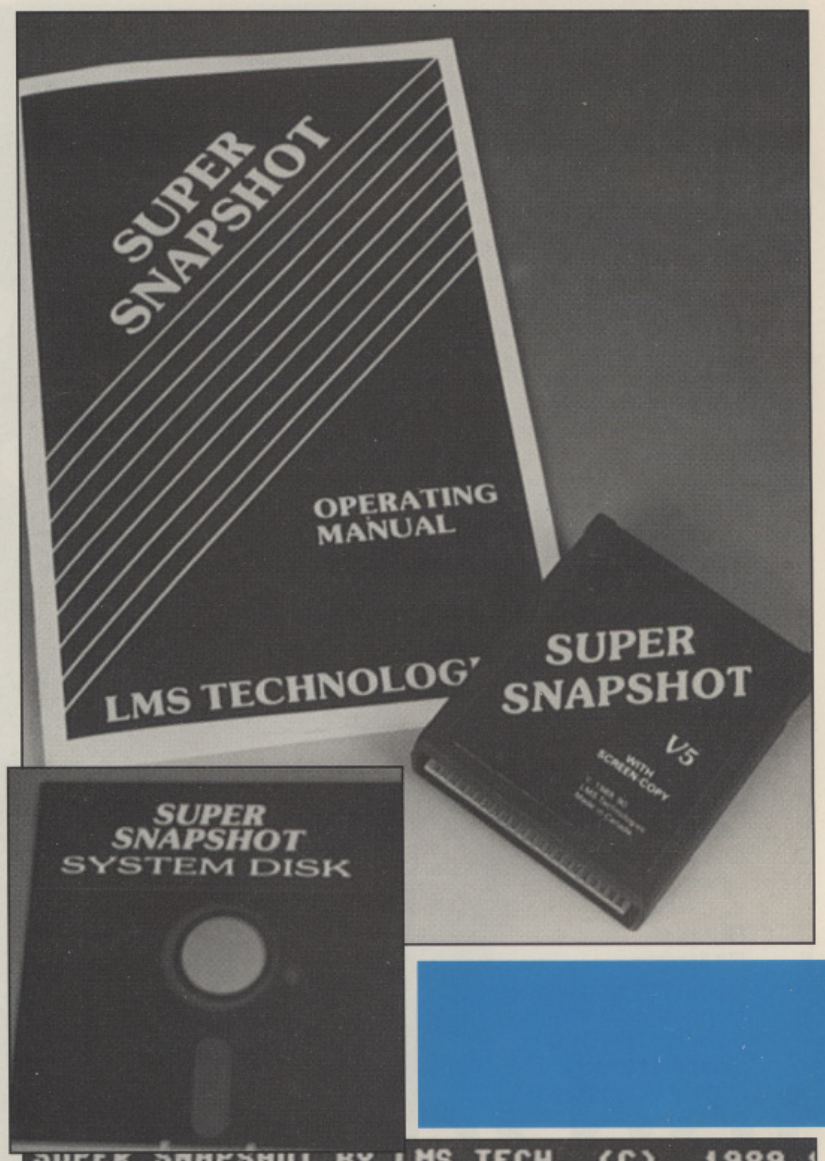
A sound sample is a way of recording any sound as a series of numbers. It is the same method used in synthesizers and CD's. With the Sample Monitor you can capture these sounds and by using the PLAYER module on the system disk, you can incorporate them into your own programs.

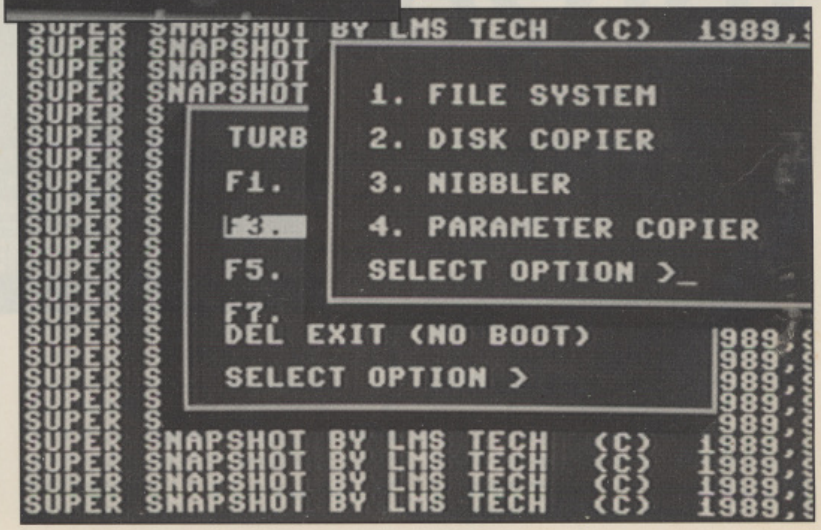
The instructions in the manual make the job of capturing a sample, then saving it out for later use in your own programs, relatively easy.

Picture This

One of the nicer facilities offered by Super Snapshot V5, is the ability to freeze a screen and save it out to disk as a picture file. Various formats are catered for here including Koala, Run Hires, Doodle, Blazing Paddles.

Another good feature is that you can also save the sprites. Once you have the screen you desire in mem-





ory, you simply press the button on the cartridge and you enter a submenu. The screen type is displayed which includes one of five types:

Standard bit mapped. Standard Character. Multi colour bit mapped. Multi colour text or just text.

A large variety of printers are catered for in the dumps, including a few of the more popular colour printers. As an exercise into the possibilities this feature offers, I tried the following.

I loaded one of my games into memory. I saved out the screen in question, including the sprites. I then ran the saved picture through a convertor program, which saved out an Amiga IFF file. The sprites I then loaded into Dpaint III and converted them to brushes. From here I completely redesigned the original screen and repositioned the sprites. Finally, I resaved the changed screen and converted it back to a C64 picture file. I then obtained a colour print out of my modified screen. All in all, a very satisfactory and rewarding aspect of this cartridge.

Round up Time

A lot of you will be disappointed that I haven't mentioned the Copiers, Nibblers, Parameter Utilities and Backup programs available. The question of the morality and ethics of these options is one which will always be strongly debated upon. Suffice to say that if you do want to make PERSONAL backups, then the facilities offered by Super Snapshot V5 are excellent.

In conclusion I will say this. If you are thinking of buying a Cartridge to update your collection, or if you want to buy one for the first time, then Super Snapshot V5 offers excellent value for money. I would go so far as to say that if a C128 switching facility had been incorporated, similar to the Warp25, then you would never need to remove this cartridge from the back of your machine.

Supplier: F.S.S.L Ltd
Price: £39.95

Fortran

Basic and 6510 are not the only languages out there. Discover the joys and pains of an alternative language

By Andy Partridge

Fortran is not a programming language I would go out and buy, but this version is quite good. It just falls down seriously in a few places.

The first thing you must do when creating a Fortran program is enter a source file (Your instructions for the computer), and funnily enough, this is where the first problem is. There is no dedicated editor for entering the source in this package. The instruction manual suggests you use a word-processor that outputs sequential files, or failing that you can enter the source using the BASIC editor (I.E. 10... 20... when you turn on the computer) and prefix all Fortran lines with ':' Also, after entering saving a file using the Basic editor, you have to load a program to convert it into a SEQ file. Why no dedicated editor? It would have been a lot easier! What if you come up with an error after all that loading/saving? You have to go all the way back to the beginning. Not Good!

The next step is to compile your program. This is where the main job of the Fortran package comes in. <<'Compiling a program' means that your Source file is converted into machine/code that the 64 understands, and thus allows the program to be executed.>>.

The compiler is quite fast, and you have the option to send the source to the printer as it is being compiled. A pause and abort mode is also supported. For users with two drives, you can take the source off one disk and compile it onto another. Still, even with all these options, should an error come up it's back to the beginning you go!

Compiled modules can be linked together to form larger modules. This is done in the linker, where options again exist to make hardcopys, use multiple drives and full error readouts.

Once the code is compiled and saved, it can be loaded and run. A basic line with a SYS XXXX to start the code is include, so no other mucking around has to be done.

```
      PROGRAM  TEST
*     Example program
      CHARACTER NAME (20)
      INTEGER FLAGS (10), A,I,Z
      DATA NAME/' JONES   '/, FLAGS (7)/ 4,5,6,7/
      OPEN 5,4
      WRITE (0,4) 'PLEASE ENTER YOUR NAME:
      READ (3,*) NAME
      DO 10 J = 10,1,-1
      WRITE (5,50) NAME
10    CONTINUE
      A=I * ((Z + A) -Z) / 14
50    FORMAT( 'THE NAME IS ' , A10)
      IF ( I.EQ 10) THEN
            WRITE (5,*) I
      ELSE
            WRITE (5,*) FLAGS(I)
      END IF
      CLOSE 5
      STOP
      END
```

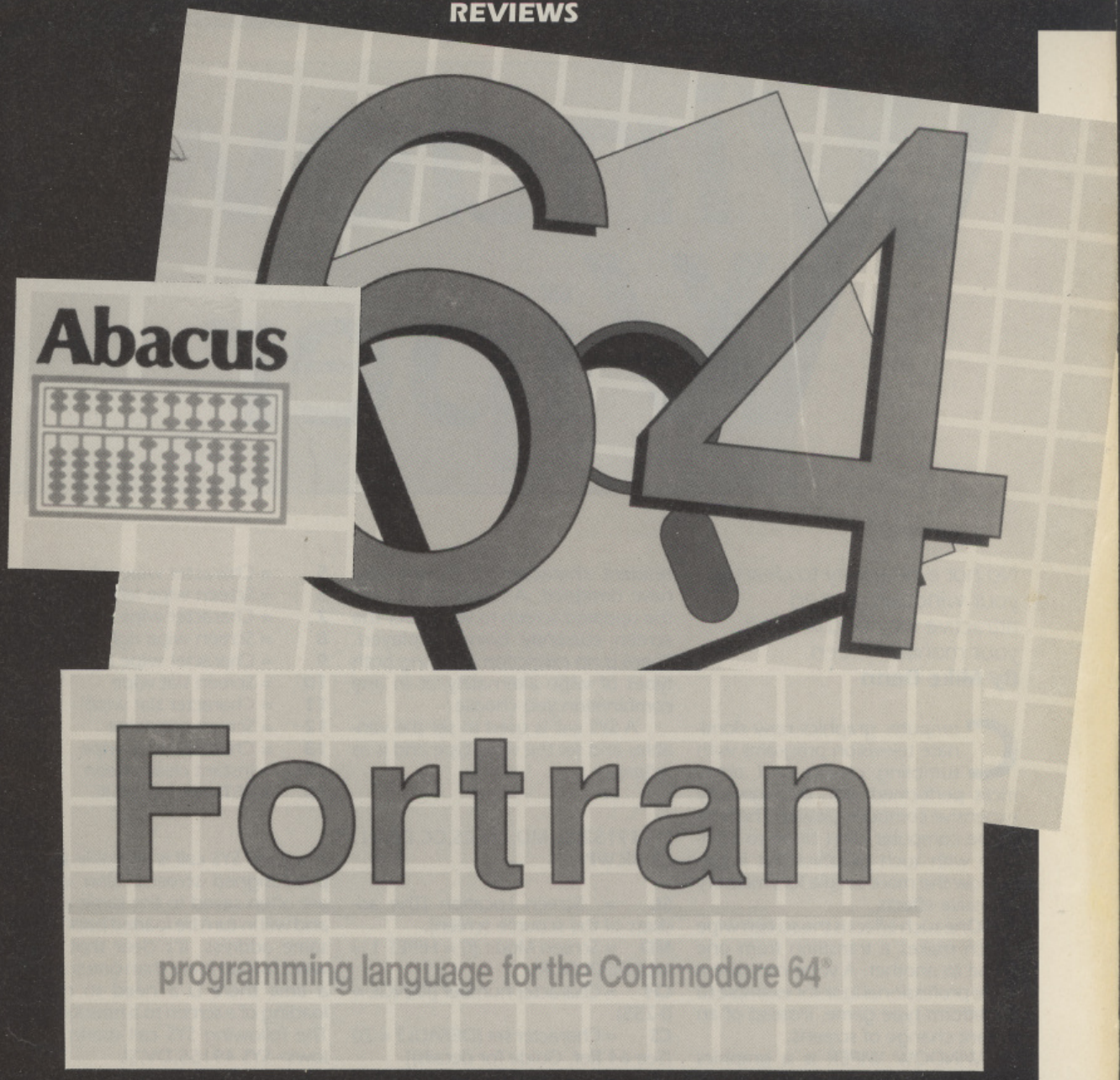
Other options exist on the main menu to translate the Basic program into a SEQ file, and to change colours. A HELP option is there also. This is a good idea, and it explains each of the options available on the main menu.

All in all, this is not really a program I would recommend someone unless they were interested in learning the language, or as part of a school project (I had to learn PASCAL, for example, as part of my GCSE Computing course!) As I said in the beginning, it's a good version, but it isn't very well presented or thought out fully in places.

This version of the language has most of the commands in it. it can handle REAL, INTEGER, LOGICAL and CHARACTER DATA. There are lots of Trigonometric functions (SIN, COS and TAN) and Arithmetic functions (EXP, ALN, LOG, ABS, TABS, POW, FLOAT, IFIX, MOD and AMOD)

Complete list of FORTRAN Commands:

ALL – Transfers program control to the specified subroutine.
CALL EXEC – Allow direct access to user of kernal machine code routines.
CLOSE – Close an Opened communication channel.
COMMENTS – Allows embedded documentation in programs.
COMMON – Allow arrays in subroutines to share same memory.
CONTINUE – Allows block structuring of a program.
DATA – Give values to variables.
DIMENSION – Define size of arrays.
DO – Allow looping on a group of



statements.
ELSE – Conditional statement 'Do this ELSE do this!'
END – Indicate the end of a program unit.
ENDCOMMON – Marks the end of command arrays.
END IF – Marks end of a block IF statement.
FORMAT – Structure I/O information.
FUNCTION – Define beginning of a Sub-routine.
GOTO – Jump to label.
IF – IF so-and-so...
IF THEN – If... then do...
IMPLICIT – Confirm the type associated with the first letter of a variable name (??!?).
OPEN – Open communication channel.
PAUSE – Stops program until a key is pressed of a certain time as passed.
PROGRAM – To name a program
READ – Get data from keyboard or device.
RETURN – Return from a subprogram.
STOP – Stops program and returns to BASIC.
SUBROUTINE – Define beginning of subprogram.
TYPE – Specify type of data in a variable.
WAIT – Halt program for a specified length of time.
WRITE – Write data on screen.

Sample Fortran Program (Written in Basic Editor)

```
10: PROGRAM TEST
20: * THIS PROGRAM WILL WRITE A
30: * LIST OF THE ODD NUMBERS
40: * FROM 1 TO 10 TO A PRINTER
50: INTEGER A, B
60: OPEN 4,4
70: DO 101=1, 10, 2
80: WRITE (4,100) 1
90L 10 CONTINUE
100: CLOSE 4
110: STOP
120: 100 FORMAT (15)
130: END
```

Window Wiper

No not a new way to clean your windows, but an attractive way of changing your monitor screen

By Mike Benn

Computer graphics now dominate television programs with tumbling credits and other tricks performed with outrageously expensive computer power. The 8 bit home computer can't hope to compete with such graphics but it can borrow the more simple less memory intensive effects.

One such effect is the screen wipe which makes a transition from one scene to another. A wipe can give a more professional look for example to a platform type game, instead of an instant change of screens.

WINDOW WIPER is a graphics utility which offers a number of wipes to include in your own programs. They can be games using multiple screens or perhaps some form of video presentation.

The program works with pre-designed screens which can be readily drawn up using a character/sprite designer. Full control over colours, character set, screen and type of wipe are manipulated from within the program. Pre-designed screens can be stored and grabbed from almost anywhere in computer memory (See OUT OF BOUNDS).

The program uses two types of wipe, character wipe and screen wipe. A character wipe uses a single character of your choice it can be part of the resident character set or one you have designed. A screen wipe uses the updated screen to wipe over the screen currently being displayed. There is no restriction on using both types of wipe alternately or in any combination you choose.

A SYS call is used to set the variables and set the wipe type and is as follows:-

SYS49152,SL,MD,CH,CS,CC,B0,B1,B2,BR,WP

SL = Screen Location (The address of the scource screen).
MD = Screen Mode (0 = HIRES 1 = MULTICOLOUR).
CH = Character number (RANGE = 0-255).
CS = Character set (DEFAULT = 20 (See 64 Ref. Guide for details))
CC = Character colour (HIRES RANGE = 0-15/MULTI RANGE = 8-15).
B0 = Background colour (RANGE = 0-15).
B1 = Background colour 1 (RANGE = 0-15).
B2 = Background colour 2 (RANGE = 0-15).
BR = Border colour (RANGE = 0-15).
WP = Wipe type (see Table).

Wipe Table

0 = Instant screen fill
1 = Instant character fill
2 = Screen wipe down
3 = Character wipe down
4 = Screen wipe up
5 = Character wipe up
6 = Screen wipe left to right
7 = Character wipe left to right
8 = Screen wipe right to left
9 = Character wipe right to left
10 = Screen slat wipe
11 = Character slat wipe
12 = Screen rain wipe
13 = Character rain wipe
14 = Instant clear screen
15 = RETURN TO BASIC

A second SYS call is available to load pre-designed screens. User screens are often saved to the same address and will in turn be loaded back to that same address. It's clear that if you want to load more than one screen at a time there is a need to redirect loading of a screen to a new location. The following SYS call works as follows:- SYS 49155,DV,SL
SL = SCREEN LOCATION to which you want the screen to be loaded.
DV = DEVICE number (8 = DISK/1 = TAPE).

Out of Bounds

The following areas should be avoided at all cost. The program will fail or worse.

000-2048 ($000-$0800)
49152-51456 ($C000-$C9000)
53248-57343 ($D000-$DFFF)

To see WINDOW WIPER in action select WIPER from the menu. This BASIC program will load all the other parts so just sit back and watch.

Commodore DISK USER

Lineage: 53p per word. (+ VAT)
Semi display: £11.50 plus VAT per single column centimetre minimum 2cm. Ring for information on series bookings/discounts.
All advertisements in this section must be prepaid.
Advertisements are accepted subject to the terms and conditions printed on the advertisement rate card (available on request).
Make cheques payable to ASP Ltd.

Send your requirements to:
CLASSIFIED DEPARTMENT
ASP LTD. ARGUS HOUSE,
BOUNDARY WAY, HEMEL HEMPSTEAD HP2 7ST.

0442 66551

DISKS

WESTONING LTD

3½ DISKS	5¼ DISKS	AMIGA
DS/DD 46p	DS/DD 23p	BATMAN PACK £355
DS/HD £1.15	DS/HD 57p	FLIGHT FANTASY PK £355
		CLASS OF 90's £510
PRICE EACH INC VAT AND LABELS	PRICE EACH INC VAT, SLEEVES AND LABELS	512K RAM £56
		WITH CLOCK £62

LOCKABLE BOXES		LOCKABLE BOXES	
CAP	CAP	CAP	CAP
40 - £4	80 - £5.75	50 - £5.25	100 - £6.50

NEC EXTERNAL
3½ DRIVE ...£60
1084D MON £239

PRICES INCLUDE VAT *NO QUIBBLE GUARANTEE*
POST ADD £3 *NEXT DAY ADD £7*

12 SANDERSON ROAD, WESTONING,
BEDFORD MK45 5JY
Tele. (0836) 775060 or (0525) 718668 Fax (0525) 715789

SOFTWARE

PUBLIC DOMAIN SOFTWARE for the C64/128.
We have 150 disks full of all types of Programs
All disks are £2.95 each
NOW FOR THE +4
from £3.75 per disk.
Send SAE or phone for free catalogue (state model)
Kingsway Computer Services
72 Glencoe Road, Sheffield
S2 2SR Tel: (0742) 750623

COMPLETE THE COUPON BELOW OR CALL 0442 66551 TO ADVERTISE

REPAIRS

M.C.E. SERVICES
33, Albert Street, Mansfield, Notts.
NG18 1EA Tel: 0623 653512

FIXED PRICE COMMODORE REPAIRS

C64	£24.95
C16	£19.95
PLUS4	£24.95
AMIGA A500	£39.95
1541	£29.95
1571	£31.95
C2N etc	£12.95

All prices inc. VAT, Labour, Parts etc.
NO MORE TO PAY.
Please ring for details.

COMMODORE SPARES & REPAIRS

REPAIRS
C64, C+4 .....£25 inc.
C128, 1541 .....£35 inc.

SPARES

906114	£9.30	901227	£11.95
6510	£9.95	8501	£11.45
6526	£10.95	C64 Power Pack	£22.00
6581	£12.95	C64 Cassette Unit	£22.00
6569	£19.95	C128 Power Pack	£69.95
		Inc Vat & PP	

OASIS COMPUTER SERVICES,
Dept 4D, 14 Ridgeway Road,
Salisbury, Wiltshire, SP1 3BU.
Tel (0722) 335061

COMMODORE REPAIRS, SPARES & LEADS
C64, +4, 16 - £24.00 C128 - £32.00
AMIGA 500 -£34.00
Three month warranty. For price list send 1st class stamp (State model)
ACE REPAIRS, Outways FM, Pelynt, Looe, Cornwall, PL13 2NW. Tel. (0503) 20282

TO ADVERTISE IN CDU CALL TONY FLANAGAN ON 0442 66551

SERVICES

COMMODORE SUPPLIES

C64 Power Unit	£21.99
C128 Power Unit	£39.99
C2N/C3N Datasette	£24.99
100 x 3½" Discs	£29.99
C64 Magnum Lightgun	£34.99
C64/128 Serial Cable	£5.99

Prices include VAT & P&P Chq/PO to:
OMNIDALE SUPPLIES (CD),
23 Curzon Street, Derby, DE1 2E6
Tel 0332 291219

Commodore DISK USER

Lineage: **53p** per word (+ VAT)
Semi Display: **£11.50** (+ VAT) per single column centimetre. Min size 2cm.
Series discounts available on request. All classified advertisements must be pre-paid.
There are no reimbursements for cancellations.
SEND TO CDU ADVERTISEMENT DEPARTMENT, ARGUS HOUSE, BOUNDARY WAY, HEMEL HEMPSTEAD, HP2 7ST

I enclose my Cheque/Postal Order for £ ..... for ..... insertions, made payable to Argus Specialist Publications.
(Delete as necessary)

PLEASE DEBIT MY ACCESS/BARCLAYCARD NO

☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐ **EXP. DATE** .....

£ .....**FOR** .....**INSERTIONS**

Name .....**Postcode** .....

Daytime Tel No. ..... **Signature** .....**Date**.....

☐ **FOR SALE** ☐ **SOFTWARE** ☐ **SPECIAL OFFERS** ☐ **REPAIRS** ☐ **HARDWARE** ☐ **DISKS**

...it's dynamite!

POWER CARTRIDGE

FOR YOUR COMMODORE 64/128

"...unbelievable value for money" ZZAPP! Dec 89

- POWER TOOLKIT
- POWER MONITOR
- TAPE & DISK TURBO
- PRINTERTOOL
- POWER RESET
- TOTAL BACKUP

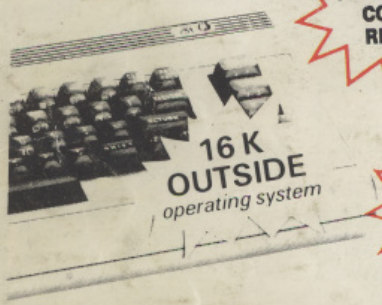
..."Money well spent" YC/CDU Jan 90

42 Pg Manual - "Damned Good Handbook" CCI Jan 90

SO MUCH FOR SO LITTLE

AVAILABLE FROM ALL GOOD COMPUTER RETAILERS

TRIED AND TESTED - OVER 100,000 SOLD IN EUROPE



...."highly recommended for C64 users" CCI Jan 90

YOU WILL WONDER HOW YOU EVER MANAGED WITHOUT IT

POWER TOOLKIT

A powerful BASIC-Toolkit (Additional helpful commands) that considerably simplifies programming and debugging.

AUTO	HARDCAT	RENUMBER
AUDIO	HARDCOPY	REPEAT
COLOR	HEX$	SAFE
DEEK	INFO	TRACE
DELETE	KEY	UNNEW
DOKE	PAUSE	QUIT
DUMP	PLIST	MONITOR
FIND	ILOAD	BLOAD

RENUMBER : *Also modifies all the GOTO's GOSUB's etc. Allows part of a program to be renumbered or displaced.*
PSET : *Set up of printer type.*
HARDCAT : *Prints out Directory.*

The toolkit commands can be used in your programs.

DISK TOOL

Using POWER CARTRIDGE you can load up to 6 times faster from disk.
The Disk commands can be used in your own programs.

BLOAD	DVERIFY	DIR
DSAVE	MERGE	DEVICE
DISK		

MERGE : *Two BASIC programs can be merged into one.*
DISK : *With DISK you can send commands directly to your disk.*

TAPE TOOL

Using POWER CARTRIDGE you can work up to 10 times faster with your data recorder. The Tape commands can be used in your own programs.

LOAD	SAVE	VERIFY
MERGE	AUDIO	

POWERMON

A powerful machine language monitor that is readily available and leaves all of your Commodore memory available for programming.
Also works in BASIC-ROM, KERNAL and I/O areas.

A ASSEMBLE	I INTERPRET	S SAVE
C COMPARE	J JUMP	T TRANSFER
D DIS-ASSEMBLE	L LOAD	V VERIFY
	M MEMORY	W WALK
F FILL	P PRINT	X EXIT
G GO	R REGISTER	$ DIRECTORY DOS Commands
H HUNT		

PRINTERTOOL

The POWER CARTRIDGE contains a very effective Printer-Interface, that self detects if a printer is connected to the Serial Bus or User-Port.
It will print all Commodore characters on Epson and compatible printers.
The printer-interface has a variety of set-up possibilities. It can produce HARDCOPY of screens not only on Serial printers (MPS801, 802, 803 etc) but also on Centronic printers (EPSON, STAR, CITIZEN, PANASONIC, etc).
The HARDCOPY function automatically distinguishes between HIRES and LORES. Multi-colour graphics are converted into shades of grey. The PSET functions allow you to decide on Large/Small and Normal/Inverse printing.
The printer PSET functions are:

PSET 0 - *Self detection Serial/Centronics.*
PSET 1 - *EPSON mode only.*
PSET 2 - *SMITH-CORONA mode only.*
PSET 3 - *Turns the printing 90 degrees!!*
PSET 4 - *HARDCOPY setting for MPS802/1526.*

PSET B - *Bit-image mode.*
PSET C - *Setting Lower/Upper case and sending Control Codes.*
PSET T - *All characters are printed in an unmodified state.*
PSET U - *Runs a Serial printer and leaves the User-port available.*
PSET Sx - *Sets the Secondary address for HARDCOPY with Serial Bus.*
PSET L1 - *Adds a line-feed, CHR$ (10), after every line.*
PSET L0 - *Switches PSET L1 off.*

POWER RESET

On the back of the POWER CARTRIDGE there is a Reset Button. Pressing this button makes a SPECIAL MENU appear on the screen.
This function will work with many programmes.

CONTINUE - *Allows you to return to your program.*
BASIC - *Return to BASIC.*
RESET - *Normal RESET.*
TOTAL BACKUP DISK - *Saves the contents of the memory onto a Disk. The program can be reloaded later with BLOAD followed by CONTINUE.*
RESET ALL TOTAL BACKUP TAPE - *RESET of any program. As BACKUP DISK but to TAPE.*
HARDCOPY - *At any moment, prints out a Hardcopy of the screen. Using CONTINUE afterwards you can return to the program.*
MONITOR - *Takes you into the Machine language Monitor.*

BDL

Bitcon Devices Ltd

88 BEWICK ROAD
GATESHEAD
TYNE AND WEAR
NE8 1RS
ENGLAND

Tel: 091 490 1975 and 490 1919 Fax 091 490 1918
To order: Access/Visa welcome - Cheques or P/O payable to BDL
Price: £16.99 incl. VAT.
UK orders add £1.20 post/pack total - £18.19 incl. VAT.
Europe orders add £2.50. Overseas add £3.50
Scandinavian Mail Order and Trade enquiries to: Bhiab Elektronik, Box 216, Norrtalje 76123, SWEDEN. Tel: + 46 176 18425 Fax: 176 18401
TRADE AND EXPORT ENQUIRIES WELCOME