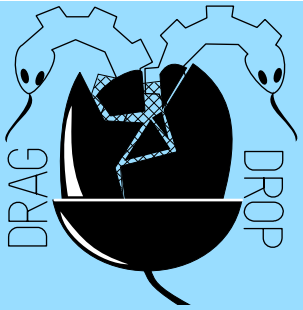

DRAG'NDROP
RISC OS Pi and all RISC OS 5 machines
30th issue
February 2017
Volume 8 Issue 2
£3.50
Mop Tops
Amcog's latest game
Feature
RISC OS sound system
Game
Sid Slug
Wimp apps
Module auto 32 bitter, ConvText
...Type 'em all in!



This issue has been blessed with contributions from the following people:
Anthony Bartram (RISC OS Sound System)
Paul Dunnington (Python Primary School)
Christopher Dewhurst (everything else)

Contact Information
Editor: Christopher Dewhurst
Email: editor@dragdrop.co.uk
www.dragdrop.co.uk

EDITORIAL

I hope you had a good Christmas and start to 2017, unlike your editor who was down with the flu!

Welcome to this 30th edition of *Drag 'N Drop*, packed with stuff to learn about your computer – including the start of an exciting new series on the RISC OS sound system, resurrecting old modules and (if your head is hurting by the end of that) a type-in arcade game.

Along with your favourite magazine there are some new products in the pipeline from *Drag 'N Drop*, the first of which is being released at this year's South West show so see you there!

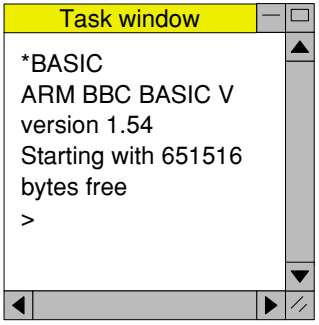
Chris.
Christopher Dewhurst

How do I...?

...get the BBC Basic prompt?

To get the BBC Basic prompt press F12 and type *BASIC and press Return. You can change the screen mode with MODE n where n is a number e.g. MODE 7 or MODE 0. Type AUTO for automatic line numbering. Press Escape to stop and type *SAVE "myprog"* followed by Return to store *myprog* on hard disc.

To return to the desktop type *QUIT. Programs listed in *Drag 'N Drop* are assumed to work on all machines with RISC OS 5 e.g. Raspberry Pi, unless otherwise stated.



You can also program and run Basic programs from the desktop. Double-clicking on the filer

icon runs it, holding down Shift and double clicking loads it into your text editor.

...open a Task window?

Menu click over the Raspberry icon on the right side of the iconbar and select click on Task window. Or press Ctrl + F12.

You may need to reserve more memory for the task in which case adjust-click on the Raspberry icon and under Application tasks click and drag the Next slide bar out to the right.

You can also type programs in a task window, hold down Ctrl and press F12. You can't use the cursor editing facility or change MODE, however.

...select the currently selected directory?

Articles may tell you to set the CSD (currently selected directory). Just click menu over filer window and choose Set directory ^W or you can use the !EasyCSD application presented in *Drag N Drop* 6i1.

...open an Applcation Directory?

Application directories begin with a ! called 'pling'. Hold down shift and double click select to open the directory.

News and App Updates

South West show

The Webbington Hotel overlooking the Somerset levels is again host to another exciting RISC OS show in the South West of England on 25th February 2017. Doors open 10.30am, tickets just £5.00. With so much happening in the RISC OS world it's something not to be missed. Get your copy of *Drag 'N Drop* for £3.00, or why not buy the all 30 issues on a USB memory stick.

20th Century Fonts

We're excited to announce the release of a super new CD-Rom, a collection of PD fonts for RISC OS to blow all other collections away. Launch price £13.00 (including printed manual if you buy at the show). Whilst PD fonts aren't 'perfect' (you would have to spend many hundreds of pounds for 'professional' typfaces) they are good enough for everyday purposes. Supplied in Type 1 format too for use on other platforms.

RISC OS 2016 awards

Get your mouse pointer out and get along to http://www.riscosawards.co.uk/vote2016.html. Vote for in what, in your opinion, were the best (and worst) RISC OS products during 2016. Be sure to elect your favourite magazine as the "Best publication or offline resource" please!

Risc Digital Sound Processor

Making sounds has always been tricky on RISC OS because it involves writing voice generators. However this is all set to change with Tony Bartram's RDSP module. It extends the SOUND statement and even adds BBC Micro-style ENVELOPEs meaning a huge variety of audio effects can be achieved with simple Basic commands. An alpha (test) version can be downloaded from www.amcog-games.co.uk/downloads/rdsp-alpha1.zip

Mop Tops

We've reviewed Amcog's latest game in this issue but as it was going to press we heard that its author, Tony Bartram, has improved the animation and added more levels. Price just £9.99 from !PlingStore or on DVD at shows.

GCC 4.7.4

The free C and C++ compiler for RISC OS has been updated to run on the Raspberry Pi 3 and can be downloaded from www.riscos.info/index.php/GCC.

Wakefield

Acorn & RISC OS Computer Show

Saturday 22nd April 2017

Show organised by Wakefield RISC-OS Computer Club, Formed 1983

The North's premier RISC OS show

Now in its 22nd year

The Cedar Court Hotel
Denby Dale Road, Calder Grove, Wakefield
West Yorkshire WF4 3QZ
(Adjacent to J39 of the M1)

Getting there by train

- Connections from across the UK to **Wakefield Westgate** station.
- Local buses twice an hour to the venue

Getting there by car

- Easy access from **M1**, **M62** and major routes
- Free on-site parking

- **Big Names and Small Developers**
- **Raspberry Pi**
- **Show Theatre**
- **On-site Catering**
- **Charity Stall** raising funds for Wakefield Hospice

Date	**Saturday, 22nd April 2017**
Opening time	**10.30am** to **4.30pm**
Ticket price	**£5** on the door Entry for children aged 12 or under (accompanied by an adult) is **FREE**
Access	Disabled access to the show venue is via a lift
Enquiries	Wakefield Show 2017, c/o 3 Riverdale Avenue, Stanley, Wakefield, WF3 4LF
Email	show2017@wakefieldshow.org.uk

For further information, visit

www.wakefieldshow.org.uk

RISC OS Sound System

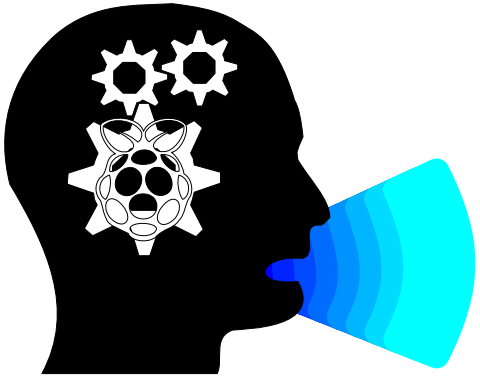
When RISC OS first appeared in the 1980s it provided eight channels for playing back 8-bit samples (or *waveforms*) in stereo.

With the advent of the RISC PC in the 1990s this was upgraded to 16 bit audio. However, the hardware interface has a only single direct memory address handler which provides a *callback* to populate the sound memory.

(If you're not sure what a *callback* is, think of passing parameters, or arguments, in procedures in BBC Basic. A *callback* is an argument except the argument itself is another piece of code and not just a data value.)

Therefore, a module has been provided within RISC OS so that the output to the 16-bit sound handler can be shared. Hence, this module is called Shared Sound.

A code example to use Shared Sound is provided below. It enables a 16-bit WAV file to be played through Shared Sound as well as generating simple BBC Micro-style tones.



This forms a very simple synthesiser written in a small amount of assembly code.

Once you have typed in the listing and checked for errors, save it. Before running it, observe the followign points:

- ensure you have set the 'next' Wimp slot large enough to hold the DIMensioned Basic memory block, at least one megabyte. (Do this by dragging the slider in the Task display. Otherwise you will get a "No room for this DIM" error.)
- set the directory to the same location as the audio sample to be loaded (it must be a WAV file, if you haven't got one then there are plenty available for free on the internet)
- run the example by double clicking it and *not* executing it from the task window.

The program has a number of simplifications for clarity including:

- No support for systems that will not play audio at 44khz. Note, though, that modern RISC OS systems support 44khz audio including the Raspberry Pi, ARMX6 as well as RPCEmu so generally you need not worry about that.
- No error handler for shared sound.
- No support for other shared sound resources running at the same time as this example is designed to run as a single

The 20th Century was a golden era in typeface development with classic and eye-catching fonts being designed and used in the world of printed media.
With the advent of digital publishing many of these have since passed into the public domain, collected for this album of over 700 beautiful fonts in RISC OS (Acorn outline) format and also supplied in 'Type 1' for conversion to other platforms e.g. Windows.

Just another collection?

OTHER FONT COLLECTIONS

- Claims 10,000s of fonts but in reality they're just tedious variations on a few fonts.
- Windows only
- Websites with downloadable fonts are transitory or become inaccessible as time goes by
- Contain lots of 'grunge' and 'distressed' fonts.

20TH CENTURY FONTS

- Over 700 unique and beautiful fonts, high quality even for PD standard.
- RISC OS format and Type 1 (PFA) for conversion to other platforms.
- Absolutely no 'grunge' or 'distressed' fonts in this collection!

20th CENTURY FONTS

- Several classic typefaces not available on the internet specially digitised in-house at Drag 'N Drop! • Printable catalogue (PDF), gazetteer, hints and tips on using fonts on RISC OS.

Special launch price £13.00

RISC OS South West Show

ConvText

ConvText is a simple but highly-customisable application which runs on the desktop to help with text file conversion.

It may be used to strip line feeds, alter line feed/carriage returns (such as when importing text from a PC) and anything else you wish by adding to its library of routines.

There are three listings, The first generates the application directory (!ConvText), sprites, Boot and Run files. The second is the main program which should be saved inside the application directory, !ConvText.!RunImage.

The third listing is in fact a series of mini Basic programs - the library routines - and they should be saved as !ConvText.Lib.Routine where Routine is the name e.g. ForceLower.

To use, double click and it will install itself on the icon bar. Select click over the icon to bring up details of the currently installed conversion routine.

To convert a text file just drag to !ConvText's icon. The converted file is saved with the same name.

To install a new library routine, menu click over the iconbar icon and select Show Library. A filer window opens on the library. Drag one of them to !ConvText's iconbar icon to install.

Installing new routines

Each Basic file in the Lib directory is a FuNction defintion - DEFFNchange(in%). *in%* is the 8-bit alue of the character at the current position in the file. An operation is performed on *in%,* for example

```
out%=in%
IF out%=ASC":" THEN out%=10
=out%
```

checks to see if it's a colon and substituting a line feed.

If one character is subtituted for two, three or even four bytes, then the *out%* should be the multi-byte value, for example & 0A0D to output a line feed and carriage return in place of a line feed.

A special case is made for characters to be ignored (removed) where out% should be set to -1 (&FFFFFFFF in hex).

ConvText Makefiles

```
  10REM ConvText Makefiles
  20REM (c) Drag 'N Drop 2017
  30
  40app$="!ConvText"
  50OSCLI"CDIR "+app$
  60PROCcreatesprites
  70PROCcreatefiles
  80END
  90
 100DEF PROCcreatesprites
 110DIM H% 500:!H%=500:H%!8=&10
```

Module Auto 32-Bitter

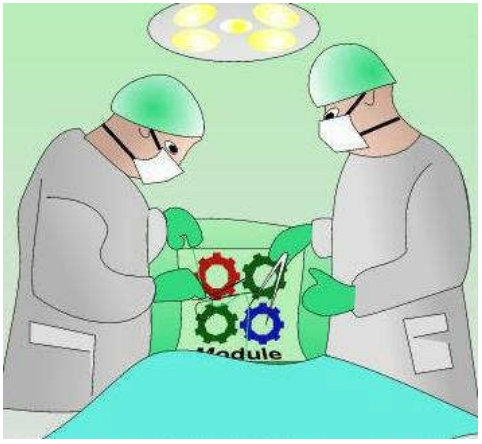
In the last instalment of the Module Surgery series (way back in the Winter 2016 edition of *Drag 'N Drop*) we looked at updating 26-bit modules to run on the Pi by editing the code in StrongEd.

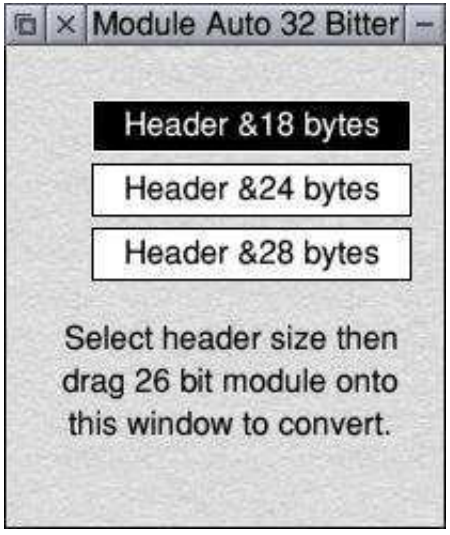
To take the tedium out of doing this and reduce the possibility of errors we present a small application to do it for you, !Auto32.

Please note that it's not guaranteed to magically transform your ancient Archimedes module from B.C. 6000 into a fully functioning modern piece of software but many 26-bit modules can be fixed using !Auto32.

There are two listings, the first creates the application directory, Boot and Run files and application sprites.

The second listing is the program which does all the hard work and should be saved as !RunImage inside the !Auto32 directory.





To use the application, double-click on !Auto32 and a window willappear. We'll discuss the three options beginning "Header..." in a minute but simply drag your module onto the window. A confirmation will be issued saying the module will be saved in the same location with '32' appended to the filename.

To quit the application simply click on its close icon.

How Auto32 works

As you will recall from last time, in earlier (26-bit) versions of RISC OS you could get away with a module header of only 28 (&18 in hex) bytes if the module didn't supply any extra SWI calls.

On RISC OS 5 (32-bit) however, a larger header size of &34 is compulsory. This is illustrated in figure 1, repeated from last time but with the addition of red lines showing header sizes in 26-bit modules.

All 26-bit modules tested with Auto32 have header sizes of &18

Review

Product: Mop Tops
Category: Gaming
Price: £9.99 (PlingStore), also available on DVD at shows
Supplier: Amcog games
www.amcog-games.co.uk

After a severe shortage of new games for RISC OS we already have four in less than two years from Amcog.

Having enjoyed Tony Bartram's *Overlord*, *Legends of Magic* and *Xeroid* I looked forward to Mop Tops. It's a Lemmings-style game where you must help the bemused-looking creatures (the Mops) through a landscape by giving them helpful objects like ladders to climb or tools like drills to get through walls.

I ran into a stumbling block before I could get Mop Tops to run. The game checks for version 1.40 of the AmPlay module but 1.39 is supplied on the DVD. A quick search of the Internet revealed that version 1.42 can be downloaded from www.riscos.info/index.php/AMPlayer.

Control is just by means of clicking the mouse on a supply of building blocks at the bottom of the screen and dragging them to position in the scene, for example wall sections to divert the Mops' path, or keys to give them to unlock doors.

The graphics are excellent with photo-quality backdrops scrolling independently of the scenery in the foreground. Synthesized music provides an audio accompaniment in an otherwise silent gameplay.

Clicking the forward or back arrows let you quickly scroll left and right through the landscape.

I found the clicking and dragging a little unresponsive but helpfully a red square appears on the screen where the block will end up when you have released the mouse button.

When you have guided all the Mops to the exit sign you move to the next level and are given a passcode so you can skip straight through the next time you play the game.

Mop Tops is another enjoyable outing from Amcog so go out and buy yourself a copy.

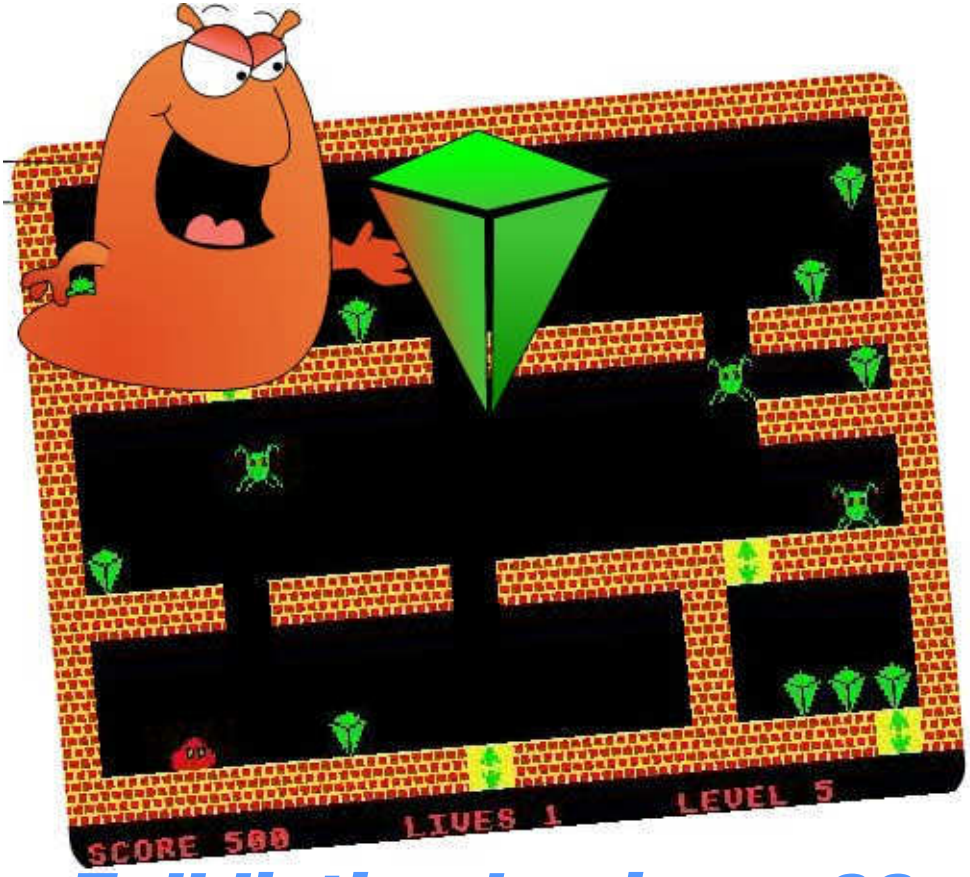
Sid Slug

Sid Slug is in a spot of bother, can you help him collect all the diamonds and escape from the underground maze whilst avoiding the aliens, heaps of rubble and other pitfalls?

Use Z and X for left and right. If you are on a lift you can press / and @ to go up and down. Additionally, Shift and Return will 'stretch' Sid in the appropriate direction enabling you to short cut distances.

When you have collected all the diamonds make your way to the exit gate, assuming of course you haven't blocked off your way back. You will then proceed to the next level.

The game ends when you have completed all levels or lost all three lives.



Full listing begins p.22

Sid Slug listing

```
  10REM Sid Slug
  20REM (c) Drag 'N Drop 2017
  30MODE13
  40DIM AL(4,3),scale 12
  50ON ERROR PROCerror:END
  60PROCsprites:TINT2,0
  70OFF
  80REPEAT
  90LE=1:sco=0:lives=1
 100REPEAT CLS
 110RESTORE (3890+LE*160-160)
 120dias=0:rubble=0:ALI=1:CLS:FOR
B=0 TO 14
 130READ A$
 140FOR A=0 TO 19:B$=MID$(A$,A+1,
1)
 150IF B$="A" THEN PROCsprite("AL
IEN",A*64,956-B*64):AL(ALI,1)=A*64
:AL(ALI,2)=956-B*64:AL(ALI,3)=1:AL
I=ALI+1
 160IF B$="R" GCOL G%?4:MOVE A*64
+28,1016-B*64:DRAW A*64+32,1016-B*
64
 170IF B$="0" PROCsprite("WALL",A
*64,956-B*64)
 180IF B$="L" PROCsprite("LIFT",A
*64,956-B*64)
 190IF B$="D" PROCsprite("DIAMOND
",A*64,956-B*64):dias=dias+1
 200NEXT:NEXT:ALI=ALI-1
 210A=64:B=124:MOV=0:MT=TIME:MO=T
IME:ALMT=TIME:PROCscore
 220PROCsprite("SLUG",A,B)
 230REPEAT
 240IF INKEY-67 AND TIME>MO+8 PRO
Cright
 250IF INKEY-98 AND TIME>MO+8 PRO
Cleft
 260IF INKEY-80 PROCup
 270IF INKEY-105 PROCdown
 280IF INKEY-74 AND rubble=0 PROC
stretchright
 290IF INKEY-1 AND rubble=0 PROCs
tretchleft
 300IF dias=0 PROCsprite("EXIT",6
4,124):dias=-1
 310IF dias=-1 AND ((B=124 AND (A
=64 OR A=96)) OR (A=64 AND B=188))
 THEN sco=sco+100:LE=LE+1:MOV=5
 320IF POINT(A+32,B-4)=0 AND POIN
T(A,B-4)=0 AND POINT(A+62,B-4)=0 A
ND MOV=0 THEN PROCfall
 330IF FNpoint(A+32,B-4,2) AND FN
point(A+30,B-4,2) AND MOV=0 THEN s
co=sco+10:dias=dias-1:PROCscore:PR
OCfall
 340IF TIME>ALMT+10 AND ALI>0 THE
N PROCmovealien
 350IF MOV=3 THEN lives=lives-1:P
ROCdie
 360IF MOV=2 AND TIME>MT+5 THEN P
ROCleft2
 370IF MOV=1 AND TIME>MT+5 THEN P
ROCright2
 380UNTIL MOV=3 OR MOV=5
 390UNTIL lives=0:*FX15,0
 400A=GET
 410UNTIL 0
 420DEFPROCright
 430IF MOV<>0 THEN ENDPROC
 440IF FNpoint(A+92,B+60,4) THEN
rubble=1
 450IF FNpoint(A+92,B,1) OR FNpoi
nt(A+92,B+60,1) OR FNpoint(A+92,B,
3) OR FNpoint(A+92,B+60,3) OR FNpo
int(A+92,B,5) OR FNpoint(A+92,B+60
,5) THEN ENDPROC
 460IF FNpoint(A+94,B,2) AND FNpo
int(A+94,B+60,2) THEN GCOL 0:RECTA
NGLE FILL A+64,B,62,60:sco=sco+10:
dias=dias-1:PROCscore
 470exit=0:IF rubble=2 THEN
 480PROCsprite("RUBBLE",A,B)
 490A=A+64
 500PROCsprite("SLUG",A,B)
 510rubble=0:exit=1
 520ENDIF
 530IF exit=1 THEN ENDPROC
 540IF rubble=1 THEN rubble=2
 550GCOL 0:RECTANGLE FILL A,B,60,
60
 560PROCsprite("SLUG2",A,B)
 570MOV=1:MT=TIME
 580ENDPROC
 590DEFPROCright2
 600MOV=0
 610GCOL 0:RECTANGLE FILL A,B,60,
60
 620A=A+32
 630PROCsprite("SLUG",A,B)
 640MO=TIME
 650ENDPROC
 660DEFPROCleft
 670IF MOV<>0 THEN ENDPROC
 680IF FNpoint(A-32,B+60,4) THEN
rubble=1
 690IF FNpoint(A-32,B,1) OR FNpoi
nt(A-32,B+60,1) OR FNpoint(A-32,B,
3) OR FNpoint(A-32,B+60,3) OR FNpo
int(A-32,B,5) OR FNpoint(A-32,B+60
,5) THEN ENDPROC
 700IF FNpoint(A-36,B,2) AND FNpo
int(A-36,B+60,2) THEN GCOL 0,0:REC
TANGLE FILL A-64,B,62,62:sco=sco+1
0:dias=dias-1:PROCscore
 710exit=0:IF rubble=2 THEN
 720PROCsprite("RUBBLE",A,B):A=A-
64
 730PROCsprite("SLUG",A,B):rubble
=0:exit=1
 740ENDIF
 750IF exit=1 THEN ENDPROC
 760IF rubble=1 THEN rubble=2
 770GCOL 0:RECTANGLE FILL A,B,60,
```

Armcode for Beebsters

In the October 2016 edition of *Drag 'N Drop* we started looking at Arm assembly language from point of view of someone used to programming in 6502 code on the BBC Micro.

We made the following observations:

- 6502 instructions can be one, two or three bytes big whereas Arm instructions are always four bytes (one word) big.
- There are three eight-bit registers in 6502 and 16, 32-bit registers in Armcode called R0...R15.
- 6502 uses regular brackets, only curly or square brackets are used in Armcode.
- There's no RTS equivalent in Armcode so a subroutine has to push the registers onto the stack with STMFD SP!,{LR} and pulll off at the end with LDMFD SP!,{PC} when it's finished.
- On the BBC Micro you could poke bits of assembly code in odd corners of memory by setting P% directly but on RISC OS it is always advisable to DIMension a block of memory and set P% to its address (DIM code% 100:P%= code : [... and not P%=&900 : [...)

Although Arm proclaims to have 16 registers in reality only 13 are available because registers 13 to 15 have special uses. R13 is the stack pointer, R14 is the link register and R15 is the program counter.

They can all be referred to by their alternative notations: SP for R13, LR for R14 and PC for R15. Which is to say, the instruction STMFD SP!,{LR} can be written STMFD R13!,{R14} and LDMFD SP!,{PC} as LDFMD R13,{R14}.

In fact, you can miss out the 'R' because the assembler understands LDMFD 13,{14} perfectly but it's not good practice since it can be confusing.

In 6502 code we are used to operating on memory locations directy. For example, the code fragment below could be used to toggle eight characters of text at memory location *data* from upper to lower case:

```
LDA #32
LDX #0
.loop
ORA data,X
INX
CPX #8:BNE loop
RTS
.data EQUD0:EQUD0
```

In Arm code all you can do is to load data from memory and store data to it, you cannot alter it directly. So we would have to write:

```
MOV R0,#32
MOV R1,#0
ADR R2,data
.loop
LDRB R3,[R2,R1]
ORR R3,R3,R0
STRB R3,[R2,R1]
ADD R1,R1,#1
CMP R1,#8
BNE loop
MOV PC,LR
.data DCD 0:DCD 0
```

DCD by the way is exactly the same as EQUD (and DCB is

DIY Pointers

If you have grown bored of the standard blue pointer on your computer screen then this article is for you.

RISC OS allows you to redefine the arrow shape to anything you wish, provided it's no larger than 32 by 32 pixels and consists of three colours.

The listing below lets you choose between a hand, wand, cigarette (not that we're encouraging *Drag 'N Drop* readers in that nasty habit), sausage and trowel. You can add plenty more of your own.

The program sets up a user sprite area and READs in the sprites from the DATA lines from 520 onwards. Each sprite is specified, firstly, with the screen Mode it was defined in. This can be any four-colour mode such Mode 1, Mode 8 or Mode 19. Next come the name and dimensions in pixels.

The meaning of the Ascii characters in the DATA lines is as follows: full stop for background,1, 2 or 3 for the three colours.

The last line for each sprite is the palette (see later). A menu is presented inviting the you to choose the desired pointer. The new pointer remains in effect until you reset your machine (or re-run the program).

The name of the standard pointer shape in the Wimp sprite pool is known as *ptr_default*. The chosen sprite in the user sprite area is renamed to *ptr_default,* the sprite area saved out as Sprites, then a *IconSprites command is issued to merge new pointer shapes to the sprite pool. Thus the new *ptr_default* replaces the old *ptr_default*.

Setting the colours for the pointer is a little unconventional. Instead of taking the colours from a palette in the sprite file, they have to be redefined with a couple of SYS calls.

SYS "Wimp_ReadPalette",, block% fetches 20 words of data in to a parameter block at *block%*. That's 16 words for the 16 Wimp colours, plus border colour and three pointer colours.

One word of memory describes the colour in &BBGGRR00 format. Pointer colour 1 is at block%+68, pointer colour 2 at block%+72 and pointer colour 3 at block%+76.

The default colours are &F0F000 (cyan) and &00F000 (blue) respectively. (The standard arrow shape only has two colours, the third colour isn't used).

The new colours are poked into the parameter block and **SYS "Wimp_SetPalette",,block%** is issued to effect the change.

To save space in the program, the palette is written in BBGGRR format (no & prefix) and line 240 uses EVALuates the hex number

Python Primary School

This term we make a start at Wimp programming with Python.

In preparation for this I want to go through **bitset()** and **padstring()** so we know what we are up against. They are used in the Bitset program listed below which demonstrates the problems we have to overcome.

Bitset/py listing

```
 1# block.bitset() test
 2# Drag 'n Drop magazine
 3# by Paul Dunnington
 4
 5import swi
 6b = swi.block(6, [0,0,0,0,0])
 7print swi.integers(b.start,6)
 8
 9b.bitset(2,7,0)
10b.bitset(3,2,0)
11b.bitset(4,7,0)
12b.bitset(5,1,0)
13print swi.integers(b.start,6)
14print
15b.padstring(chr(7),'a',3,4)
16b.padstring(chr(2),'A',4,5)
17b.padstring('',chr(7),5,6)
18b.padstring('',chr(1),6,7)
19print swi.integers(b.start,6)
20print
21print swi.tuples(b.start,4,6)
22print
23b.padstring('',chr(3),0,1)
24b.padstring('',chr(1),1,2)
25b.padstring('',chr(12),2,3)
26b.padstring('',chr(0),3,4)
27print swi.tuples(b.start,4,6)
28print
29b.padstring('',chr(4),2,6)
30print swi.tuples(b.start,4,6)
31#b.bitset(4,0xff030012,0)
32b.bitset(5,-1,0)
33print swi.integers(b.start,6)
34print
35b[5] = -16580590
36#b[5] = 0xff030012
37print swi.integers(b.start,6)
38
39#os.system("ScreenSave SDFS::RISCOSpi.$.Screens.
Python")
```

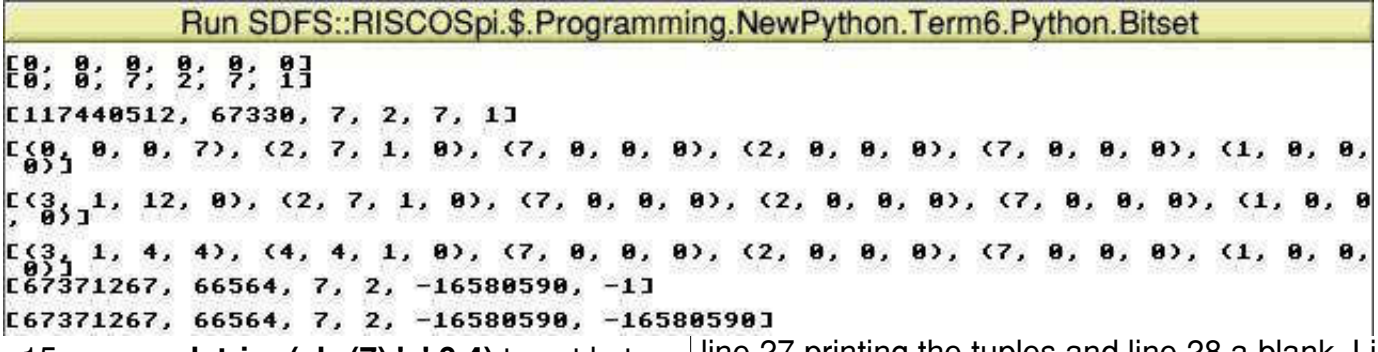
Note: Python programs unlike Basic do not have line numbers. They are shown in the listings for reference only. If you are using StrongEd turn the automatic line numbering on BaseMode > Choices > LineNos > Physical. Edit does not have this facility but tapping f5 will tell you the line number the caret is on.

Line 5 **import**s **swi** then line 6 defines a **block** of 6 words, initialising it with the list of five integers and padding out with zeros. Line 7 **print**s out the six integers for confirmation. Line 9 uses **bitset(i,x,y)** to set the bits in b[i] to x after clearing

them with y. Thus **b.bitset(2,7,0)** reads the word from block *b* at index *i*, here 2, then bitwise ANDs with y, here 0, which clears the whole word to 0, and then EORs with x, here 7, so setting the word at b[2] to 7.

In effect, when y = 0, we get b[i] = x. Note that b[2]=7 is *not* equivalent to Basic's b?2=7, Python sets the whole word as in b!8=7 so b[2]=7 is the way unless we want to set, or clear, individual bits in a word.

Lines 10-12 set the next three words before line 13 prints six integers as the result. Line 14 outputs a blank line. The first line of the screen shot shows six zeros even though we gave a list of five. In fact we only need a list of one zero to clear the block, **b=swi.block(6,[0])**. The second line shows words 2 to 5 set to 7,2,7 and 1.



Line 15 uses **padstring(chr(7),'a',3,4)** to set byte three of the block, the most significant byte of the first word of the block, to 7. It's putting the string, here only a single character **chr(7)** into the block starting at byte three, up to but not including byte four, padding with **a** if needed.

The **a** isn't needed here but we have to include the pad character or we get an error. Line 16 does the same for byte four, inserting a 2 and not padding with A.

Line 17 is a neater way to achieve our goal, inserting a null string, and padding with **chr(7)** from byte five up to, but not including, byte 6. Line 18 does the same for byte seven, inserting a 1.

Line 19 prints the list of integers, line 20 a blank, line 21 prints a list of six tuples, each containing four bytes, and line 22 prints another blank.

Line 4 of the screenshot shows the 6 integers but the bytes we just set are not obvious. Line 6 and 7 of the screen shows all the bytes in tuples of 4 and we can see the bytes are where we put them.

Lines 23 to 26 of the program set the first four bytes to 3, 1, 12 and 0 in the same way, with line 27 printing the tuples and line 28 a blank. Line 29 then uses the same method to put a 4 into bytes 2 through 5, (from index 2 to, but not including, 6), with line 30 printing the result. Line 31 uses **bitset()** to insert a hex number with the top bit set into the 4th word of the block and line 32 inserts -1 into

Tracing Outlines

A need which often arises in graphics design on computers, particularly in the application of fonts and typography, is the prodution an outline version of a shape originally constructed with thick lines.

To illustrate what I mean, supposing you have drawn the shape in Figure 1 in Draw – a trivial example consisting of one curve and a line thickness of 4 points.

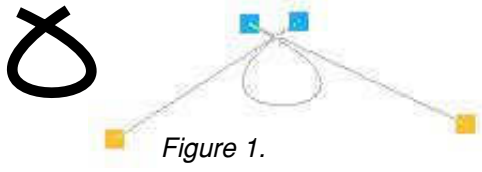


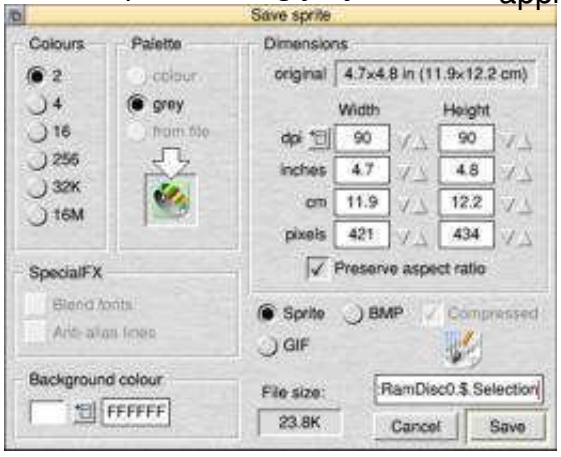
Figure 1.

Now, if you wanted to enlarge or reduce the shape you can use Draw's magnify option (Menu > Transform > Magnify). Scaling of line width will be taken care of provided your original line width wasn't set to 'thin')

This wouldn't work though if the shape was part of an outline font. What has to happen is that the outline of the shape has to be traced to result in a filled shape shown in figure 2.

Figure 2.

Fortunately this doesn't have to be done painstaikingly by hand.



Essentially you take a snapshot of the shape in bitmap format, trace the bitmap, and import the result back into Draw.

A couple of excellent (and free) tools are available in RISC OS to do this.

They are DrawToSprite from http://www.sinenomine.co.uk/software/index.html and Trace from http://www.davidpilling.com/wiki/index.php/Trace.

There seems to be only one download option for Trace, the source code, but inside the download you will find the !Trace application which is all you need.

It's best to blow your shape up before tracing to achieve best results. I enlarged the example shape by a factor of 10 before importing it into DrawToSprite.

Drag the draw file onto DrawToSpr's iconbar icon. A window appears with lots of options. You don't need to adjust any of them for our purposes, the number of colours can remain at 2, a grey

Module Saver

Relocatable modules are machine code programs which when loaded act as an extension to the operating system.

Whilst RISC OS has a 'star' command to load a module (*RMLoad) there isn't. strangely, a complementary one to save a module.

The short program presented here fills that omission by providing a *RMSave command. The assembly code creates a small utility file called RMSave in the currently selected directory.

Press Ctrl+F12 to open a task window and type ***modules** to see the list of modules installed on your machine.

Type ***RMSave <module>** where <module> is the name of the module whose code you wish to save, for example ***RMSave WaveSynth** will save a copy of the WaveSynth module in the currently selected directory.

You may wish to move RMSave to your machine's library, which is located at !Boot.Library so that it can be used independently of the currently selected directory.

RMSaveSrc listing

```
  10REM RMSaveSrc source code
  20REM saves out module code
  30REM Drag N Drop January 2017
  40
  50DIM code &100
  60FOR pass=0 TO 3 STEP 3
  70P%=code
  80[OPT pass
  90STMDB sp!,{r0-r7,lr}  ;preser
ve registers r0-r7 and link
 100MOV r6,r1  ;copy r1 to r6
 110LDRB r0,[r6,#0]  ;get Ascii c
ode
 120CMP r0,#32  ;is Ascii code le
ss than 32?
 130ADRLT r0,syntaxerror  ;yes, p
ut address of error in r0
 140BLT error  ;and jump to error
 code to advise user of correct sy
ntax.
 150
 160MOV r0,#18
 170MOV r6,r1
 180SWI "XOS_Module"  ;Perform SY
S "OS_Module",18,<module name>
 190BVS error  ;O/Flow flag set o
n return, jump to error routine
 200
 210MOV r0,#10  ;OS_File 10
 220MOV r1,r6   ;Pointer to filen
ame from r6
 230LDR r2,filetype  ;File type s
tored at 'filetype'
 240
 250LDR r5,[r3,#-4]  ;length of m
odule stored in word previous to m
odule position
 260MOV r4,r3  ;start of module c
ode
 270ADD r5,r4,r5  ;add length to
get end of module code
 280MOV r3,#0  ;r3 not used in th
is call, set to zero
 290SWI "XOS_File"
 300BVS error  ;O/Flow flag set s
o report error
 310LDMIA sp!,{r0-r7,pc}  ;all ok
 so return to Basic.
 320
 330;  ***error routine***
 340.error STR r0,[sp]
 350LDMIA sp!,{r0-r7,lr}
 360MSR CPSR_f,#1<<28 ;set O/Flow
 flag in status register
 370MOV pc,lr  ;and return to Bas
ic.
 380
 390.filetype EQUD &FFA  ;file ty
pe 'Module'
 400
 410.syntaxerror
 420EQUD &DC
 430EQUS "Syntax: *RMSave <module
>"+CHR$0
 440ALIGN
 450]
 460NEXT
 480SYS "OS_File",10,"RMSave",&FF
C,,code,P%
```