DRAG'N'DROP

RISC OS **Pi** and all RISC OS 5 machines

October 2016
Volume 8 Issue 1
£3.50

Floozy

256 colour palette switching

Machine code stars

This issue has been blessed with contributions from the following people:
Robin Wilks (Forth)
Paul Dunnington (Python Primary School)
Christopher Dewhurst (everything else)

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

EDITORIAL

Welcome to Volume 8 of *Drag 'N Drop.* The Pi-Top will be featuring on the magazine covers, it's a laptop which could be described as the first native ARM machine since the Acorn A4.

CJE Micros have been hard at work writing RISC OS support software including desktop apps for brightness control and battery indicator. Of course, we've never had so many choices of hardware on which to run RISC OS so if Pi-Top isn't your cup of tea then the ARMX6 or something called the "LaPi AtrixRO" might!

For those attending the London Show and want the *Drag 'N Drop* back issues stick we have a special birthday 'memory cube' (limited stock) Hope to see you there!

Chris.

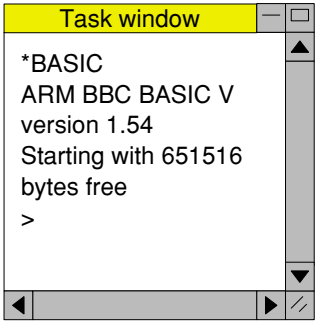
Christopher Dewhurst

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.

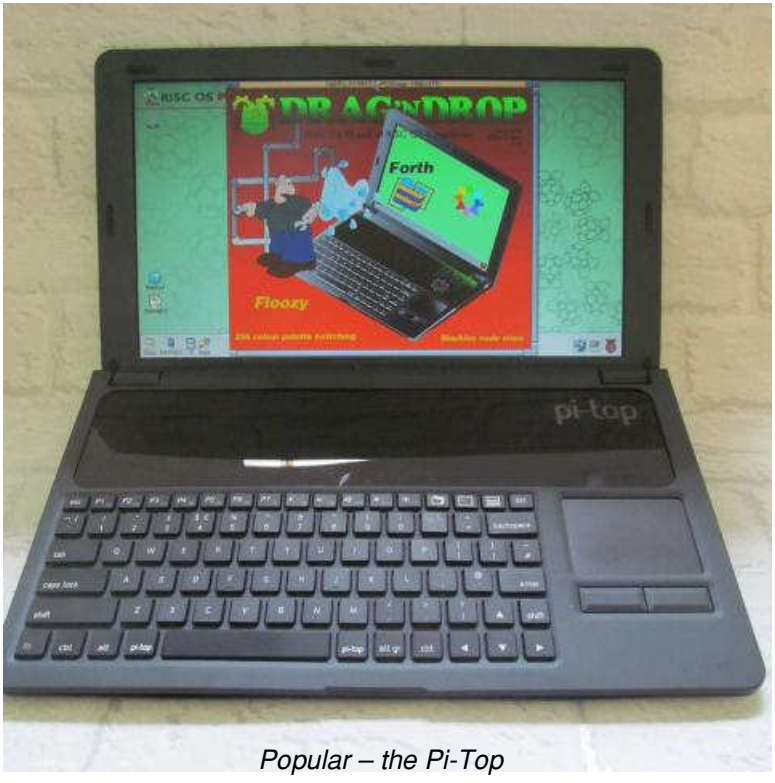
...open an Applcation Directory?

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

News and App Updates

RISC OS London Show

The RISC OS premier show takes place on 29th October at the St Giles Hotel in Feltham, London. The show runs from 11am to 5pm and costs £5 to get in on the door. Come along for an eclectic mix of Acorn past and RISC OS present including an upgrade to the popular Pi-Top computer. Get the latest *Drag N Drop* and lots of other special offers from our stand including a limited edition memory box (flash USB and chocolates) to celebrate the start of volume 8.



Popular – the Pi-Top

RISCBook Go!

R-Comp Interactive have unveiled the latest option for running RISC OS under emulation on a Windows machine, with the bonus that most legacy RISC OS software works! Prices for the RISCBook Go! start at £349 and more details are to be found at http://www.riscbook.co.uk

Text>Draw 1.32

This application aids inputting of text containing different styles (subscript, italic etc.) in Draw. The latest version can be downloaded for free from http://www.chris-johnson.org.uk/software/tdraw.html.

TranJPEG 1.33

A desktop utility for rotating and transforming JPEG photos(using jpegtran) which works on the Raspberry Pi. Download from http://www.chris-johnson.org.uk/software/tranjpeg.html.

AMPlayer 1.41

AMPlayer is a relocatable module which plays audio MPEG files such as MP3s. Desktop apps like DigitalCD use the module and the latest version of AMPlayer can be downloaded for free from http://www.riscos.info/index.php/AMPlayer.

Impact 3.51

A minor upgrade to fix a fault in the version of the relational database released at the RISC OS Wakefield Show is free to users already on version 3.47 or later. See http://sinenomine.co.uk/software/impact/.

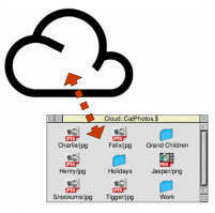
Fireworkz 2.20

A major update to the commercial version of the spreadsheet application is to be released at the London show. A promising new feature is support for a global clipboard so you can copy data from and to other applications and even between icons.

Both RISC OS and a Windows version of Fireworkz come on one CD-Rom costing £39 (upgrades from earlier versions at lower cost) with a Getting Started HTML Guide.

CloudFS

Elesar (the people responsible for development of the Titanium computer) have released CloudFS which is a RISC OS filing system allowing connection to storage on a remote server. Files can be seen by your smartphone, PC etc. as well. CloudFS costs £28.80 and more details are at http://shop.elesar.co.uk/index.php?route=product/product&product_id=63

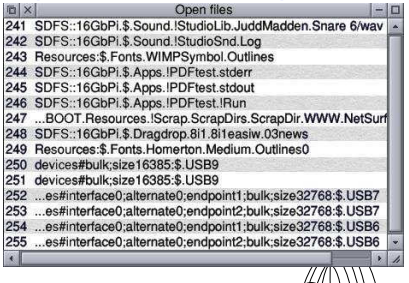


RISC OS FR

David Feugey has been busy supporting french-speaking RISC OS users at https://www.riscos.fr/, English version at https://www.riscos.fr/english.html. The website now hosts a *pret-a-manger* version of RPCEmu for newcomers to get started more easily in the RISC OS world. Also RISC OS FR now hosts subdomains, the first one being https://www.henrikbp.riscos.fr/

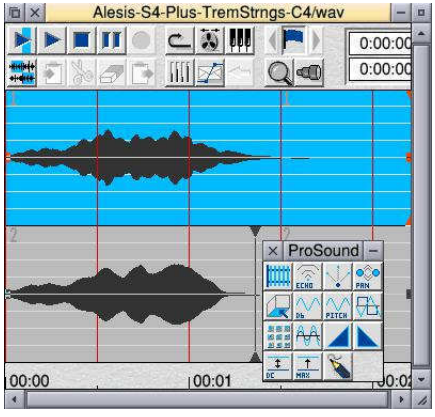
Closer 1.01

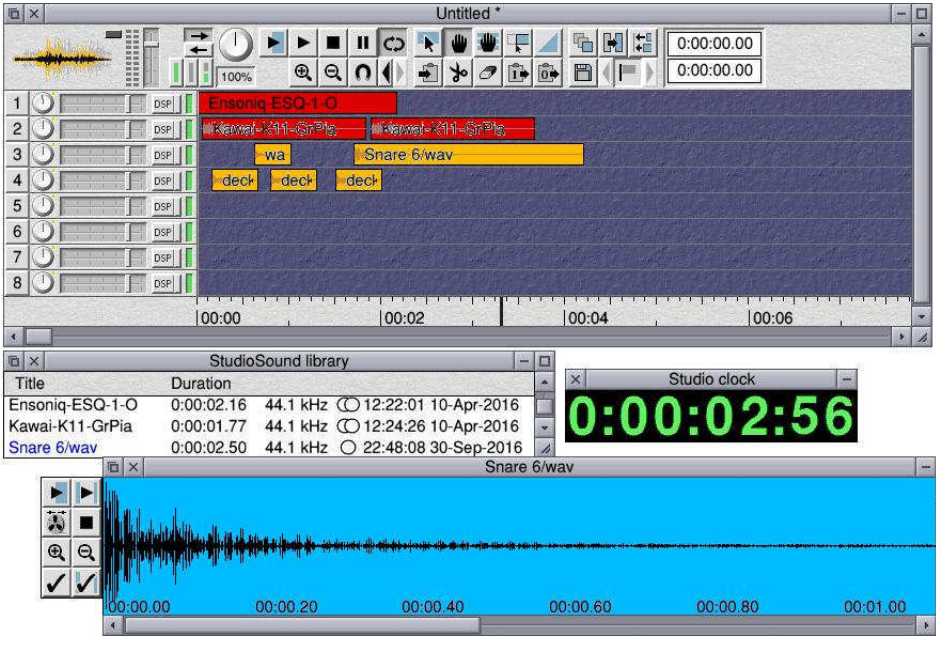
A simple but useful desktop app showing details of currently open files and the opportunity to close them. Version 1.01 can be downloaded from https://www.henrikbp.riscos.fr/



StudioSound 2.05

Fully Raspberry Pi-compatible, StudioSound lets you compose abstract electronic music soundscapes by dragging and dropping WAVe samples onto the main window with options to control track volume, pan, digital sound processing (DSP), looping etc. The latest version is FREE and can be downloaded from https://www.henrikbp.riscos.fr/. Note you may also require the THHeap module from http://www.filebase.org.uk/software/programming/982



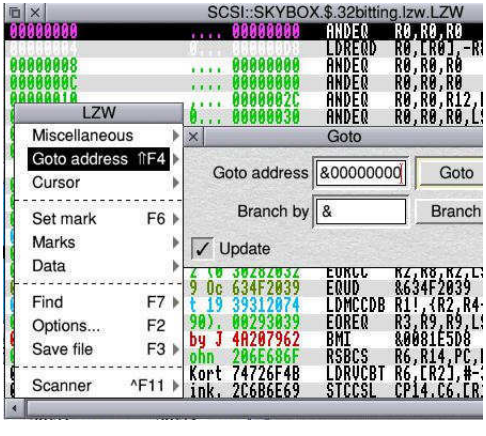


ProSound 2.01

This is a sophisticated sound sample player and editor. A variety of effects can be applied to samples such as WAV files – flange, fade in/out, echo etc. Edited samples can then be used to compose your piece in StudioSound. We have found it ProSound works and feels a lot better than PlayIt and it's also FREE from https://www.henrikbp.riscos.fr/

DisAssem 3.26

A desktop utility to disassemble ARM machine code files and modules and available free from https://www.henrikbp.riscos.fr/



Files of the World

5 MIDI Files

In this instalment we look at Midi Files a development of the Musical Instrument Digital Interface (MIDI) standard devised in the 1980s.

We'll first take a quick look at an application which plays Midi files, then the Midi file structure itself, and finally armed with the information learnt we'll write our own BASIC program to display and play Midi files – very simple, it has to be said, but with much scope for enhancement.

The Midi player I use on my Raspberry Pi is ReMIDI and its home page can be found here: http://www.amplitude.demon.nl/remidi.html.

Download **rel061p.zip (2477k) complete with samples**. Copy this out into a suitable location on your hard disc. Then download **lyonix compatible version without samples** and copy it over the top. Double click to put ReMIDI on the icon bar.

Next you need some Midi files, there are thousands of free ones on the internet. http://www.mfiles.co.uk/ is a good place to start and is compatible with Netsurf.

Midi files will usually be recognised and filetyped MIDI by Netsurf. If they aren't then use the filer to set the filetype to &FD4.

Then just drag the file onto ReMIDI's iconbar icon and it will play.

Midi files are sequences of musical events and can be thought of as like instructions on which note(s) to play on the piano keyboard and when.

The actual piano sound sample is stored on your computer (within ReMIDI in fact). Note this is different from WAV files which we looked at last time. Waveform data isn't included in Midi files, just directions on when and how to play them.

We're going to look at a simple tune, *Twinkle Twinkle Little Star* as an example.

Figure 1 is an annotated dump of the first 256 bytes of *Twinkle*. The left hand column is the offset in bytes from the start of the file, the middle column the bytes in hex, the right hand column the Ascii representation – which is what you would see in a memory editor. Lines across the top of bytes in the middle column group particular bytes in order.

By now we're used to files having a *header* of some sort. This is the first few bytes of a file identifying what the file is. Midi files are no exception.

The header kicks off with a word (four bytes) which spell MThd (the bytes 4D 54 68 64 in hex).

The next word is the length of the header in bytes (not including the MThd) which has always been 6 but subject to change. Why is the 6 the fourth byte and not the first? I'll come to that later.

Forth on the Pi

Forth is a computer language created by Charles Moore in the 1960s. "It behooves new programmers to sample all languages available. Forth is the only one that's fun," he said.

In the beginning Forth wasn't so much a computer language as a complete programming system with operating system, editor, assembler plus a compiler for finished code and an interpreter for program development.

In Forth there are just a few primitive instructions called *words* (forming the nucleus of the system) with all other words being defined in terms of these primitives, or each other.

So Forth is 'extensible' - new words are treated the same as those in the nucleus. To run a new word its name only has to be typed (or loaded) and parameters are passed between words via the *stack.*

Nowadays Forth is usually simulated and is available for many different OSs including RISC OS on the Pi.

!ARMForth is available from my website http://www.rforth.uk and was originally written in 1992 by Rob Turner. Since the source code is written in BBC Basic and assembly code (using Basic's built-in assembler) it was relatively easy to update it to run on modern RISC OS machines.

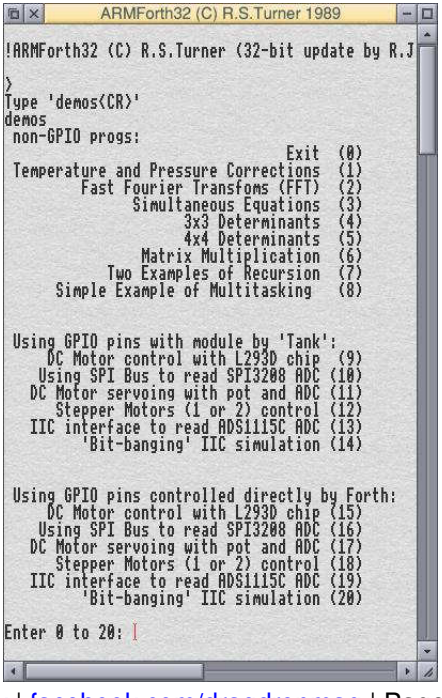
I have called the updated version ARMForth32 to distinguish it from the original software which would run on the Acorn Archimedes only.

Download **ForthPi/zip** from the Downloads section.Then extract **!ARMforth32** to a suitable location on your hard disc e.g. in **16GbPi.$.Programming**.

Double-click on the  !Armforth32 icon in the normal way and a blue and yellow icon will appear on the icon bar.

Clicking select on this icon will launch the Forth application specified in the file **!ARMforth32.Examples.!Startup**. In this case **!Startup** is a Forth program giving a menu of example programs. Type **demos** followed by Return for a list of example programs, then a number between 1 and 20 to run example programs.

Incidentally I have not used a Pi3 so don't know if there are any problems with ARMForth32 it.



M/Code Scrolling Stars

Here we feature several machine code routines which scroll a field of stars on the screen, ideal for backgrounds in arcade games.

After typing one or more listings of your choice save it in case you've made a typing slip in the assembly code which could hang the machine.

The top left hand corner of the screen has an address in Ram which varies from screen mode to screen mode and machine to machine.

Some calls to the operating system are made to ascertain this value, the extent of the screen area, and also the number of pixels across the screen. A line by line commentary on how the program works is given below.

STARS

```
   10REM *** M/C Scrolling Stars ***
   20REM (c) Drag 'N Drop Oct 2016
   30MODE 13
   40ON ERROR REPORT:PRINT" at ";ERL:END
   50OFF
   60PRINT TAB(9,15)"SCROLLING STARS DEMO"
   70ns%=40
   80PROCmc
   90CALLinit
  100FOR i%=0 TO ns%*4-4 STEP 4
  110i%!stars%=RND(!scrsiz%)+!scrtop%
  120NEXT
  130CALL plot
  140REPEAT
  150CALL scroll
  160a=INKEY(5)
  170UNTIL 0
  180END
  190
  200DEF PROCmc
  210DIM code% 1000
  220FOR pass=0 TO 2 STEP 2
  230P%=code%
  240[ OPT pass
  250.plot
  260STMFD sp!,{lr}
  270ADR r0,stars%  \star table addr in r0
  280MOV r1,#ns%-1  \no of stars minus 1 in r1
  290.loop
  300MOV r2,r1,lsl#2  \r2 = r1*4
  310LDR r3,[r0,r2]   \get star's screen address
  320LDRB r4,[r3]     \load byte from screen ram
  330EOR r4,r4,#RND(255)  \EOR it with 255
  340STRB r4,[r3]     \store back in screen ram
  350SUBS r1,r1,#1   \decrease star pointer
  360BPL loop       \repeat until done all 50
  370LDMFD sp!,{pc}  \return to Basic
  380
  390.scroll
  400STMFD sp!,{lr} \store copy of Basic return addr
  410BL plot  \erase stars
  420LDR r5,scrend%  \screen ram end in r5
  430LDR r6,scrsiz%  \screen ram size in r6
  440ADR r0,stars%   \star table addr in r0
  450MOV r1,#ns%-1 \no of stars minus 1 in r1, star pointer
  460LDR r4,xpix%  \no of pixels across screen in r4
  470.loop2
  480MOV r2,r1,lsl#2  \r2 = r1*4
  490LDR r3,[r0,r2]  \get star's screen addr
  500ADD r3,r3,r4    \add no pixels across screen
  510CMP r3,r5       \off bottom of screen?
  520SUBGE r3,r3,r6  \subtract screen ram size if so
  530STR r3,[r0,r2]  \store updated address
  540SUBS r1,r1,#1  \decrease star pointer
  550BPL loop2   \until done all 50
  560BL plot     \replot stars
  570LDMFD sp!,{pc} \restore Basic
```

Floozy

The local water board has claimed to invent a revolutionary new pipe to replace existing ones but unfortunately the Tetraflurine Carbonadium making up the pipes has disintegrated and you have been employed to put all the pipes back before the houses are flooded!

Move the cursor around the screen with the Z, X, / and @ keys and press Return to lay a pipe section from the display in the top left corner.

If you fail to connect the pipe before the water reaches you then you lose the game!

The game features animated water, six types of pipe section and and a high score table.

Type in the Basic listing and save it before running, there is a short machine code section and if there are bugs it could crash your machine.

Save it on your hard disc and double click from the desktop to run.

Full listing begins p.19

Procedures

cycle Cycle the colours, calls machine code routine to scan screen to change pixels of value D% to E%.

dead% Did not manage to complete pipeline so display failure message

delay Delay routine

completed Game complete routine

game Main game proc

hiscore Player has achieved hall of fame, invite to enter name

init Sets up the sprites.

mc Assemble machine code

scores Display high score and instructions

space Print message and await press of space bar

sprite() Plot sprite number A% at (X%,Y%)

text() Centre and Print multi-coloured text

Variables

dead% is TRUE if flooze seeps out of unconnected pipe section

dir Current direction water is flowing

fin% only TRUE when pipeline is complete

fx,fy Flooze co-ordinates

fzi,fzc Flooze counters

fza Flooze colours (Basic GCOL values)

fzb Flooze colours (screen pixel colour values)

grid() Map of playing area

hi$() High score names

hi%() High scores

nextt Next pipe section to be laid

score% current score

sheet% current screen number

tile Current pipe section being laid

xpos,ypos player position

```
3970D.!KKKKKKKKK$!!!!!!!!!!!!!!!!
3980D.!````KK```$!!!!!!!!!!!!!!!!
3990D.!``6``K``6$$!!!!!!!!!!!!!!!!
4000D.!``6``6````$$$$$$$$$$$$$$$!
4010D.!``6``6``6``K``6``6````6$!
4020D.!````6K``6``6`````6``66KK!
4030D.!6666KK66K666K6666K66KKKK!
4040D.!KKKKKKKKKKKKKKKKKKKKKKKK!
4050D.!K`!!!KKKKKKKKKKKKK!!!!K!
4060D.!K`66!KKKKKKKKKKKKK!66`K!
4070D.!K`66!KKKKKKKKKKKKK!66`K!
4080D.!K```!KKKKKKKKKKKKK````K!
4090D.!KKKKKKKKKKKKKKKKKKKKKKK!
4100D.!!!!!!!!!!!!!!!!!!!!!!!!!
4110
4120D.!!!!!!!!!!!!!!!!!!!!!!!!!
4130D.!KKKKKKKK$$!!!!$$KKKKKKKK!
4140D.!K````KKKK$!!!!$KKKK!```K!
4150D.!K`66!KKKK$!!!!$KKKK!66`K!
4160D.!K`66!KKKK$!!!!$KKKK!66`K!
4170D.!K!!!!KKKK$!!!!$KKKK!!!`K!
4180D.!````KK```$!!!!$``K`````K!
4190D.!``6``K``6$!!!!$6`6``666K!
4200D.!``6``6```$!!!!$``6``6``K!
4210D.!$`6``6``6$!!!!$6`6``6``K!
4220D.!$$$$$$$$$$!!!!$K`6````6K!
4230D.!!!!!!!!!!!!!!$K66K6666K!
4240D.!!!!!!!!!!!!!!$KKKKKKKKK!
4250D.!!!!!!!!!!!!!!$``K````KK!
4260D.!!!!!!!!!!!!!!$$6`6``66`6!
4270D.!$$$$$$$$$$$$$$6K`6``KK`6!
4280D.!$`6``6``6``K``6``6````6K!
4290D.!````6K``6``6`````6``66KK!
4300D.!6666KK66K666K6666K66KKKK!
4310D.!KKKKKKKKKKKKKKKKKKKKKKKK!
4320D.!K`!!!KKKKKKKKKKKKK!!!!K!
4330D.!K`66!KKKKKKKKKKKKK!66`K!
4340D.!K`66!KKKKKKKKKKKKK!66`K!
4350D.!K```!KKKKKKKKKKKKK````K!
4360D.!KKKKKKKKKKKKKKKKKKKKKKK!
4370D.!!!!!!!!!!!!!!!!!!!!!!!!!
```

Python Primary School

In classtime in the Summer 2016 edition of *Drag 'N Drop* we looked at a Python program **Temp/py** with routines to controll a temperature sensor connected to the Pi's GPIO (General Purpose Input/Output port) and examples of how to access RISC OS's SWI calls.

To complement **Temp/py** we'll look at second program in class today named **PyPressure/py** which uses a Freescale I2C

Precision Altimeter to read the air pressure. This chip runs on 3.3 volt and so can be directly connected to the GPIO pins on the Pi.

I bought my MPL3115A2 Altitude/Pressure/Temp Sensor Breakout Board from Hobbytronics. You may need the ROOL wiki for the information on OS_IICOp and also the iic_transfer structure, they are not in the PRMs.

The application note AN4519 for the MPL3115A2 can be found at http://www.freescale.com/ as well as the data sheet at http://cache.nxp.com/files/sensors/doc/data_sheet/MPL3115A2.pdf?fsch=1&sr=10&pageNum=1).

I originally used **swi.integers()** to read the block of data returned by OS_IICOp and another five lines to separate each byte, 22 lines in all.

I then remembered **swi.tuples()** and when I worked out how to use the list of tuples there were only 10 lines of code, a useful saving.

There are four lines of comments at the program start, then line 6 **imports time** as well as **swi**. A polling loop was used to start with but changed to **time.sleep()** to wait for the conversion to finish.

Line 7 creates an eight-byte block for the register reads in the global namespace so the **read_iic()** function as well as the rest of the program can access it. We only read six bytes here but blocks are defined in words, so two words equals eight bytes.

Lines 9 to 12 are the **read_iic()** function definition which takes three parameters:

- **iic** is the IIC address of the sensor
- **register** the address to start reading from
- **number** is how many reads to make.

Line 10 defines another block of one word and is initialised with the **register** variable, so we now have **dat.start** as a pointer to register.

Line 11 defines yet another block of six words, initialised as two **iic_transfer** structures, (address, start register, data length).

The first writes the register to start reading from passing one

Armcode for Beebsters

Having learnt machine code on the BBC Micro there were some aspects of Arm machine code with which I initially struggled.

So I thought I would put together a couple of articles (this issue of *Drag 'N Drop* and the next) to help people reared on 8-bit 6502 assembly transfer their skills to writing 32-bit Armcode.

All versions of BBC Basic have a built-in assembler, something which was unique in Beeb days (contemporary micros just left programmers to "hand assemble" code and poke it into memory.) A typical assembly looks like:

```
DIM code 100
FOR pass=0 TO 2 STEP 2
P%=code
[OPT pass
\...
\assembly code
\...
]NEXT
```

Memory is reserved with the DIM statement and P% is set to the address of *code* where the machine code begins.

The above framework is the same whether you are on 6502 or Armcode.

On the BBC Micro you could assemble code in odd corners of memory by setting P% directly:

```
FOR pass=0 TO 2 STEP 2
P%=&900
[OPT pass
...
```

This is dodgy on RISC OS because there is no guarantee which part of the memory your program resides in.

PAGE is normally get &8F00 so you can sometimes get away with resetting PAGE to a higher value and assembling at &8F00 but isn't advisable.

You should always DIMension a block of memory on RISC OS to assemble code in.

6502 programs generally always end with an RTS to get back to Basic:

```
oswrch=&FFEE
DIM code 20
P%=code
[
LDA #ASC"A":JSR oswrch \code for A
RTS \return to BBC Basic
```

Before Jumping to the SubRoutine the 6502 processor pushes the return address (the address in the Program Counter) onto the stack. When the RTS is encountered the address pulled from the stack and put onto the Program Counter so execution continues where it left off.

At first sight there seems to be no equivalent of RTS in Armcode. How do you return to Basic after the following?

```
MOV R0,#ASC"A" \Ascii code for A
SWI "OS_WriteC" \write it
```

In Armcode the Program Counter is held in Register 15. It's more commonly called "the PC" instead of "Register 15". Before a subroutine, a copy of the PC is put into Register 14 (R14). R14 is also called the link register or LR for short.

256 colour palette switching

One of the few drawbacks of RISC OS 5 compared to earlier versions of the operating system is that the hardware can only display Modes with 256 (or more) colours.

The lower colour depths (2-, 4- or 16-colour Modes) available on RISC OS 2-4 were useful for animation. This was achieved by palette switching using VDU 19 but these Modes aren't available on the Pi.

Whilst VDU19 can still be used in 256 colour modes of the Pi the shade of the colours can only be subtly altered and not switched to another colour to achieve animated effects.

The trick I'll reveal in this article is to get a short machine code routine to scan the screen and change pixels of a specified colour to another.

It is necessary to understand the relationship between colours as Basic sees them and how the display hardware sees them.

There are 64 colours available with the COLOUR and GCOL statements numbered 0 to 63. 64 to 127 are the same as 0 to 63 and 128 to 255 the same as 0-127 but set the text background in the COLOUR statement.

Bits 0-1 specify the amount of red, bits 2-3 the green and 4-5 yellow.

Number 15 (%001111, green plus red) is a bright yellow colour and the following program plots a yellow pixel at the top left hand corner of the screen.

```
  10 REM Program 1
  20 MODE 13
  30 PRINT
  40 GCOL 0,3
  50 PLOT 69,0,1023
  60 DIM T 8
  70 !T=149:T!4=-1
  80 SYS"OS_ReadVduVariables",T,T
  90 R=!T
 100 PRINT POINT(0,1023)
 110 PRINT ?R
```

The pixel is then read back using Basic's POINT statement which gives 15 as you would expect. However the byte stored in the screen ram (peeked in line 110) is different, it's 119.

What happens is the bits in the Basic colour are all repositioned in the screen colour as shown in Figure 1.

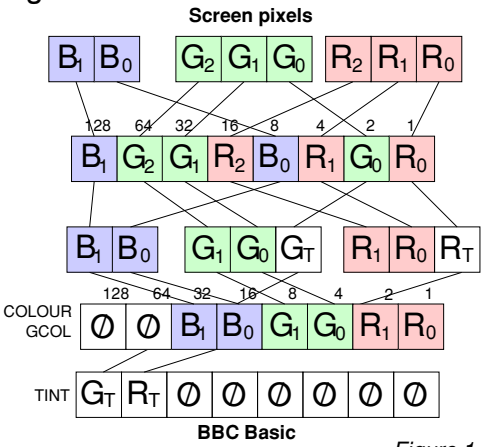


Figure 1

The 'tint' is a two bit value used by Basic to brighten or darken its 64 colours thereby achieving the 64*4=256 colours (although only 64 are available at one time)

As you can see it's a dog's breakfast but a study of our yellow pixel will make things clear. It starts off in Basic as:

There are four possible screen

Fibonacci Wallpaper

This short program shows off the power and speed of BBC Basic on the Raspberry Pi by generating patterned wallpaper on the screen.

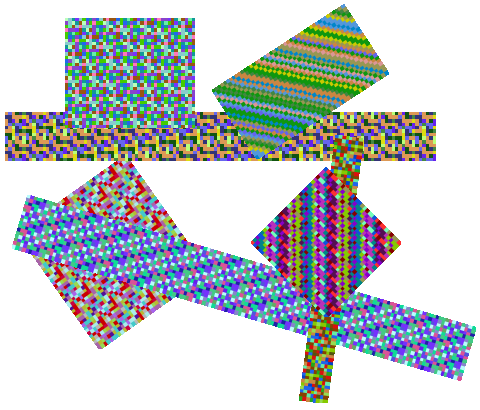
It uses the Fibonacci sequence first documented by the 12th century Italian mathematician. Each number in the sequence is the sum of the previous two. If you reduce the result modulo 10 then the pattern repeats after a while.

This repetition is exploited by the program. The starting numbers are chosen at random between 1 and 9. These are indexes into a *table* of colours. *xw* and *yw* are the dimensions of the rectangle chosen at random but within the limits of the Mode 13 screen, 320 pixels across and 256 down.

The address of the top left hand corner of screen ram is found by a SYStem call and stored in *R*. *S* is the offset from this address.

There are 320 pixels across the Mode 13 screen, the rectangle may be smaller than this however so once a line of pixels of length *xw* is filled 320 is added to *S*.

If you see a design you like you can tap the S key to save it as a sprite to the current directory. It can be loaded into Paint and perhaps incorporated into scenery sprites for games.



FibWall listing

```
   10REM Fibonacci Wallpaper
   20REM By C.R.Dewhurst
   30REM (c) Drag 'N Drop October
2016
   40MODE 13:OFF
   50DIM table 9
   60!table=149:table!4=-1
   70SYS "OS_ReadVduVariables",tab
le,table
   80R=!table
   90N=0
  100REPEAT
  110 CLS
  120 xw=RND(320)
  130 yw=RND(255)
  140 S=0
  150 FOR x=0 TO 9
  160  x?table=RND(255)
  170 NEXT
  180 n1=RND(9)
  190 n2=RND(9)
  200 FOR Y=1 TO yw
  210  FOR X=1 TO xw
  220   S?R=n2?table
  230   temp=n1
  240   n1=n2
  250   n2=(n2+temp)MOD10
  260   S=S+1
  270  NEXT
  280  S=S+(320-xw)
  290 NEXT
  300 A=INKEY(200)
  310 IF A=ASC"S" PROCsave
  320UNTIL 0
  330END
  340:
  350DEF PROCsave
  360VDU24,0;1023-yw*4;xw*4;1023;
  370a$="screensave fib"+LEFT$("00
0",3-LENSTR$N)+STR$N
  380N=N+1
  390OSCLI a$
  400ENDPROC
```

MemAlloc module

This is a relocatable module originally dating from the early days of RISC OS which has been improved and updated to run on the Raspberry Pi.

It allows you to control various aspects of RISC OS – size of the Ram disc, system sprite area, etc. – from within programs by use of nine extra 'star' commands.

The listing assembles and saves the module. To install double click **MemAlloc,** press Ctrl+F12 and type ***Help MemAlloc** to list the commands and ***Help *command*** (where *command* is one of the commands provided) for further information.

MemAlloc listing

```
   10REM MemAlloc
   20REM Updated for 32 bit machin
es
   30REM by Drag 'N Drop, October
2016
   40
   50i1$="If it is not possible to
 obtain this amount of memory, "
   60i2$="If it is not possible to
 do this, "
   70o$="the optional * command wi
ll be executed."+CHR$13
   80
   90DIM Q% 3000
  100FOR I%=0 TO 3 STEP 3
  110P%=0
  120O%=Q%
  130[OPT I%+4
  140.header
  150EQUD 0 ; Start offset
  160EQUD 0 ; Initialisation offse
t
  170EQUD 0 ; Finalisation offset
  180EQUD 0 ; Service call handler
 offset
  190EQUD title ; Title string off
set.
  200EQUD help ; Help string offse
t.
  210EQUD commands ; Help and comm
and keyword table
  220EQUD 0
  230EQUD 0
  240EQUD 0
  250EQUD 0
  260
  270\for 32 bit compatibility
  280EQUD 0
  290EQUD module_flags
  300.module_flags EQUD 1
  310
  320\Title string
  330.title
  340EQUS "MemAlloc"+CHR$0
  350ALIGN
  360
  370\Help string
  380.help
  390EQUS "MemAlloc "+CHR$9+"0.2 "
+MID$(TIME$,5,11)+CHR$0
  400ALIGN
  410
  420\Command table
  430.commands
  440FNassemblecommands
  450EQUD 0 ;end of command table
  460
  470\Code for *SystemSize
  480.code1
  490STMFD sp!,{lr}
  500MOV R2,#&0800
  510BL getnum
  520MOV R0,#0
  530BL doit
  540LDMFD sp!,{pc}
  550
  560.help1
  570EQUS"*SystemSize allows you t
o set the system heap size in Kbyt
es, "
  580EQUS i1$
  590EQUS o$
  600.syntax1
  610EQUS"Syntax: *SystemSize (siz
e) [(*command)]"+CHR$0
  620ALIGN
  630
  640\Code for *RMASize
  650.code2
  660STMFD sp!,{lr}
  670MOV R2,#&1000
  680BL getnum
  690MOV R0,#1
  700BL doit
  710LDMFD sp!,{pc}
  720.help2
```

```
  730EQUS"*RMASize allows you to s
et the RMA size in Kbytes. "
  740EQUS i1$
  750EQUS o$
  760.syntax2
  770EQUS"Syntax: *RMASize <size>
[<*command>]"+CHR$0
  780ALIGN
  790
  800\Code for *ScreenSize
  810.code3
  820STMFD sp!,{lr}
  830MOV R2,#&01E0
  840BL getnum
  850MOV R0,#2
  860BL doit
  870LDMFD sp!,{pc}
  880.help3
  890EQUS"*ScreenSize allows you t
o set the screen memory size in Kb
ytes. "
  900EQUS i1$
  910EQUS o$
  920.syntax3
  930EQUS"Syntax: *ScreenSize <siz
e> [<*command>]"+CHR$0
  940ALIGN
  950
  960\Code for SpriteSize
  970.code4
  980STMFD sp!,{lr}
  990MOV R2,#&1000
 1000BL getnum
 1010MOV R0,#3
 1020BL doit
 1030LDMFD sp!,{pc}
 1040.help4
 1050EQUS"*SpriteSize allows you t
o set the system sprite size in Kb
ytes. "
 1060EQUS i1$
 1070EQUS o$
 1080.syntax4
 1090EQUS"Syntax: *SpriteSize <siz
e> [<*command>]"+CHR$0
 1100ALIGN
 1110
 1120\Code for FontSize
 1130.code5
 1140STMFD sp!,{lr}
 1150MOV R2,#&0400
 1160BL getnum
 1170MOV R0,#4
 1180BL doit
 1190LDMFD sp!,{pc}
 1200.help5
 1210EQUS "*FontSize allows you to
 set the font cache size in Kbytes
. "
 1220EQUS i1$
 1230EQUS o$
 1240.syntax5
 1250EQUS"Syntax: *FontSize <size>
 [<*command>]"+CHR$0
 1260ALIGN
 1270
 1280\Code for RAMFSSize
 1290.code6
 1300STMFD sp!,{lr}
 1310MOV R2,#&1000
 1320BL getnum
 1330MOV R0,#5
 1340BL doit
 1350LDMFD sp!,{pc}
 1360.help6
 1370EQUS "*RAMFSSize allows you t
o set the RAM disc size in Kbytes.
 "
 1380EQUS i2$
 1390EQUS o$
 1400.syntax6
 1410EQUS "Syntax: *RAMFSSize <siz
e> [<*command>]"+CHR$0
 1420ALIGN
 1430
 1440\Code for RMAFree
 1450.code7
 1460STMFD sp!,{lr}
 1470MOV R2,#&1000
 1480BL getnum
 1490STMFD sp!,{R1,R2}
 1500MOV R0,#5
 1510SWI "OS_Module"
 1520LDMFD sp!,{R0,R1}
 1530SUBS R1,R1,R2
 1540BMI exit
 1550BEQ exit
 1560STMFD sp!,{R0,R1}
 1570MOV R0,#1
 1580SWI "OS_ReadDynamicArea"
 1590MOV R0,R1
 1600LDMFD sp!,{R1,R2}
 1610ADD R2,R2,R0
 1620MOV R0,#1
 1630BL doit
 1640.exit LDMFD sp!,{pc}
 1650.help7
 1660EQUS "*RMAFree allows you to
set the free space in the RMA in K
bytes. "
 1670EQUS i2$
 1680EQUS o$
 1690.syntax7
 1700EQUS "Syntax: *RMAFree <free
space> [<*command>]"+CHR$0
 1710ALIGN
 1720
 1730\Code for SpriteFree
 1740.code8
 1750STMFD sp!,{lr}
 1760MOV R2,#&1000
 1770BL getnum
 1780STMFD sp!,{R1,R2}
 1790MOV R0,#8
 1800SWI "OS_SpriteOp"
 1810SUB R2,R2,R5
```

```
1820LDMFD sp!,{R0,R1}
1830SUBS R1,R1,R2
1840BMI exit2
1850BEQ exit2
1860STMFD sp!,{R0,R1}
1870MOV R0,#3
1880SWI "OS_ReadDynamicArea"
1890MOV R0,R1
1900LDMFD sp!,{R1,R2}
1910ADD R2,R2,R0
1920MOV R0,#3
1930BL doit
1940.exit2 LDMFD sp!,{pc}
1950.help8
1960EQUS "*SpriteFree allows you
to set the free space in the syste
m sprite area in Kbytes. "
1970EQUS i2$
1980EQUS o$
1990.syntax8
2000EQUS "Syntax: *SpriteFree (fr
ee space) [(*command)]"+CHR$0
2010ALIGN
2020
2030\Code for FontFree
2040.code9
2050STMFD sp!,{lr}
2060MOV R2,#&400
2070BL getnum
2080STMFD sp!,{R1,R2}
2090MOV R0,#0
2100SWI "Font_CacheAddr"
2110SUB R2,R2,R3
2120LDMFD sp!,{R0,R1}
2130SUBS R1,R1,R2
2140BMI exit3
2150BEQ exit3
2160STMFD sp!,{R0,R1}
2170MOV R0,#4
2180SWI "OS_ReadDynamicArea"
2190MOV R0,R1
2200LDMFD sp!,{R1,R2}
2210ADD R2,R2,R0
2220MOV R0,#4
2230BL doit
2240.exit3 LDMFD sp!,{pc}
2250.help9
2260EQUS "*FontFree allows you to
 set the free space in the font ca
che in Kbytes. "
2270EQUS i2$
2280EQUS o$
2290.syntax9
2300EQUS "Syntax: *FontFree (free
 space) [(*command)]"+CHR$0
2310ALIGN
2320
2330.getnum
2340STMFD sp!,{lr}
2350MOV R1,R0
2360MOV R0,#&20000000
2370SWI "OS_ReadUnsigned"
2380MOV R2,R2,LSL #10
2390.loop LDRB R0,[R1,#0]
2400CMP R0,#0
2410CMPNE R0,#10
2420CMPNE R0,#13
2430MOVEQ R1,#0
2440LDMEQFD sp!,{pc}
2450CMP R0,#32
2460ADDNE R1,R1,#1
2470BNE loop
2480.loop2 LDRB R0,[R1,#0]
2490CMP R0,#0
2500CMPNE R0,#10
2510CMPNE R0,#13
2520MOVEQ R1,#0
2530LDMEQFD sp!,{pc}
2540CMP R0,#32
2550ADDEQ R1,R1,#1
2560BEQ loop2
2570LDMFD sp!,{pc}
2580
2590.doit
2600STMFD sp!,{R0-R2,lr}
2610SWI "OS_ReadDynamicArea"
2620LDMFD sp!,{R0,R3}
2630LDMFD sp!,{R2,lr}
2640RSB R1,R1,R2
2650STMFD sp!,{R3,lr}
2660SWI "XOS_ChangeDynamicArea"
2670LDMFD sp!,{R0,lr}
2680MOVVC pc,lr
2690STMFD sp!,{lr}
2700TEQ R0,#0
2710SWINE "OS_CLI"
2720LDMFD sp!,{pc}
2730
2740]
2750NEXT
2760
2770a$="*SAVE MemAlloc "+STR$~Q%+
" +"+STR$~P%
2780PRINT a$
2790OSCLI a$
2800*settype memalloc module
2810END
2820
2830DEF FNassemblecommands
2840RESTORE
2850FOR com%=1 TO 9
2860READ command$
2870[OPT I%+4
2880EQUS command$+CHR$0
2890ALIGN
2900EQUD EVAL("code"+STR$com%)
2910EQUD &FF0001
2920EQUD EVAL("syntax"+STR$com%)
2930EQUD EVAL("help"+STR$com%)
2940]
2950NEXT
2960=0
2970
2980DATA SystemSize,RMASize,Scree
nSize,SpriteSize,FontSize,RAMFSSiz
e,RMAFree,SpriteFree,FontFree
```

Autofocus

Usually on RISC OS you have to click somewhere in a desktop window for it to gain the input focus. This short machine code utility automatically focuses the input on the window under the pointer.

The listing assembles and saves a file called AF. Double click AF to install. To quit, right click on the Raspberry icon, click menu over Autofocus in the Application tasks section and choose Quit.

One easy improvement you can make is to package up the AF file inside an application with an application sprite, !Boot and !Run file.

AutofocSrc listing

```
   10REM Autofocus
   20REM Automatically focuses inp
ut
   30REM on window under pointer.
   40REM Drag 'N Drop October 2016
   50
   60DIM code 200
   70FOR pass=0 TO 3 STEP 3
   80P%=0
   90O%=code
  100[OPT pass+4
  110
  120MOV R0,#200
  130LDR R1,task
  140ADR R2,name
  150SWI "Wimp_Initialise"
  160
  170.loop
  180MOV R0,#0
  190ADR R1,block
  200SWI "Wimp_Poll"
  210TEQ R0,#17
  220TEQNE R0,#18
  230BNE skip
  240LDR R0,[R1,#16]
  250TEQ R0,#0
  260BEQ exit
  270.skip
  280SWI "Wimp_GetPointerInfo"
  290LDR R0,[R1,#12]
  300LDR R2,temp
  310TEQ R0,R2
  320BEQ loop
  330STR R0,temp
  340MOV R8,R1
  350MVN R1,#0
  360MOV R2,#0
  370MOV R3,#0
  380MVN R4,#0
  390MVN R5,#0
  400SWI "Wimp_SetCaretPosition"
  410MOV R1,R8
  420B loop
  430
  440.exit
  450SWI "Wimp_CloseDown"
  460MOVS PC,R14
  470.temp EQUD 0
  480
  490.task EQUS "TASK"
  500.name EQUS "Autofocus"+CHR$0:
ALIGN
  510.block EQUS STRING$(32,CHR$0)
  520
  530]
  540NEXT
  550a$="*SAVE AF "+STR$~code+" +"
+STR$~P%
  560PRINT a$
  570OSCLI a$
  580*settype af absolute
```

Line 60 reserve 200 bytes of memory to assemble code into.

Lines 70-100 assemble code at O% but as if to run at P% by using OPT with bit 3 set.

Lines 120-150 is the equivalent of SYS "Wimp_Initialise",200, "TASK","Autofocus".

Lines 180-200 call Wimp_Poll and test for reason codes 17,18 (message).

Lines 280-410 obtain position of pointer and mouse button state, offset 12 is window handle under pointer. If same as last time round loop skip. Otherwise set up parameter block and call SWI to focus the input. ■