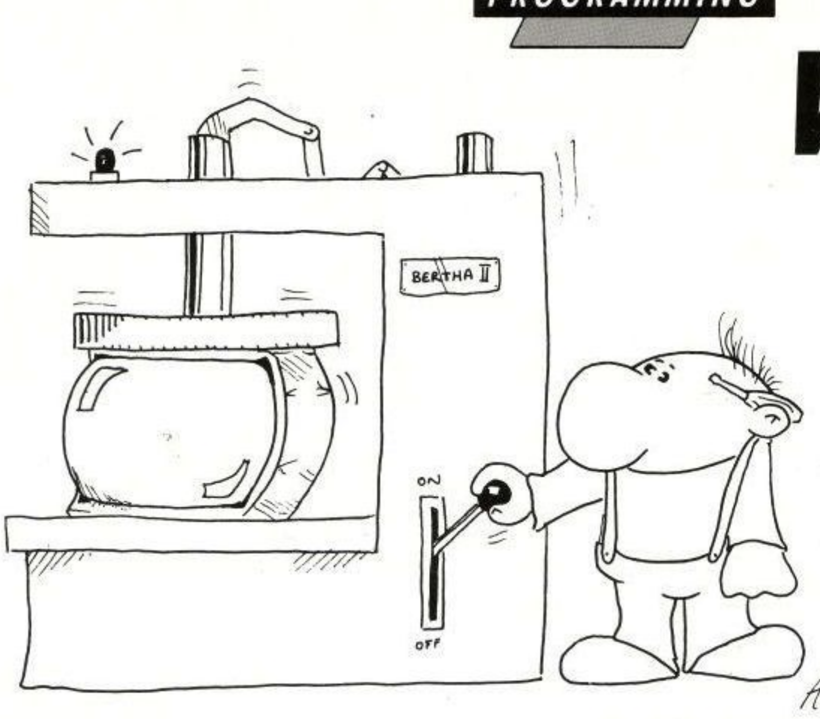


When it comes to the crunch

Auntie John has designs on data compression

FOR some reason the display of a computer, or more accurately the screen memory, always gets more attention from machine code programmers than any other area. Far be it for me to fly in the face of public opinion. This month we are going to look at a routine that will compress screens.

What this means is that the 16k normally needed for storing the pixels is going to be reduced, on average, to about 9k, allowing screen designs to take up less space on disc and tape, and so speeding loading and saving.

The amount of memory saved depends a great deal on the content of the design. This will become apparent when I explain the compression technique to be used.

Due to the Bermuda Twilight Triangle Zone, which as you know is centred over the tiny village of Brentwood in Amstradshire, this issue of ACU is dated March, although it is barely February as you are reading and, as I write, Scott and Charlene aren't even engaged yet.

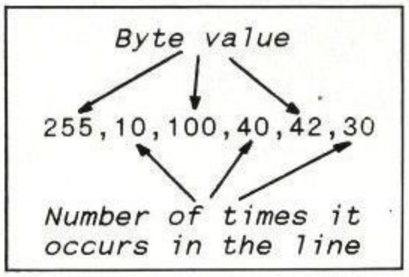
An imaginative explanation for this time difference was once put forward by Lance Davis, our letters editor, who suggested that by dating the magazine in this way we could all get a month off for holidays in July or June. Or was it May?

Anyway, as with everything else you read in *ACU*, this should be taken not only with a pinch of salt, but with a dash of pepper, a dollop of brown sauce and perhaps a little tomato ketchup or Soy sauce as well. But I digress – I only wanted to warn you that St. Valentine's Day is coming up soon. I want lots of cards, preferably from the female readership.

Back to data compression. The technique we are going to use is called *run-length* coding. Instead of storing each byte of the display, we store the value of the byte and how often it occurs. For example, if the first line of the screen contained 10 bytes of value 255, 40 bytes of value 100 and 30 bytes of value 42 we would store the bytes as shown in Figure I.

You can see that if the screen contains simple blocks of colour with little variation, the memory saving will be very large. On the other hand if the screen is very complicated – a digitised picture for example – the saving will not be so great.

Other compression techniques are used for these more complicated pictures, and perhaps we'll look at some in a later life.



A hypothetical line of data or a line of hypothetical data

Thus we have two programs to write: One to compress our screen design, another to decompress and redisplay it. Such routines running in Basic could take several minutes; in Z80 machine code, however, the timing is a matter of seconds.

Simply red

Isn't it typical. No sooner have I poked fun at Chris the Hippie for having a Spectrum than he goes out and buys an Atari 520ST. It appears that all the trendy new computers are that funny off-white colour. You know, white with a hint of cooking chocolate. Therefore to obtain maximum street credibility with your trusty gun-metal CPC, I recommend removing the outer casing and spraying it with white car paint. Remember to wave bye-bye to your warranty as you turn the screws.

If you want to be even trendier, you can use a different colour. I predict that bright red micros are just around the corner. Why not be the first on the block with a pillar-box CPC?

For the more fashion conscious, stylish sticky-backed plastic is available in an assortment of colours. Care should be taken when applying it – one key looks very like another when coated in a layer of bright green plastic. And although such an arrangement will liven up your letters, your machine code programs will suffer.

The individual style of the programmer is also considered very important in some circles: Hawaiian beach shorts are out, but dark glasses and peaked caps are very popular. If the glasses are very dark, difficulty will be experienced in reading the screen and will give the same results as the sticky-backed plastic.

Anyway, Listing I is an assembly language program suitable for producing run-length data. It makes several assumptions. The first is that the screen memory starts at hex address &C000, meaning that the screen has not been scrolled. The second assumption is that the compressed screen data is to be placed at address &4000 onwards. For this reason, always set HIMEM to below &4000 before calling the routine – *MEMORY &3FFF* will do the trick.

The length of the compression data is stored at hex addresses &8060 and &8061. After the routine has been called, the length can be calculated as follows:

```
210 lngth=PEEK(&8060)+PEEK(&8061)*256
```

The *compression ratio* – a measure of how successful the compression was – can be found with two lines of Basic. The value 16384 is how many bytes a non-compressed screen takes up:

```
220 ratio=100-(length/16384)*100
230 PRINT "Compression:";ratio;"%"
```

A ratio of less than 20 per cent means that your screen display is just too darn complicated and it is probably not worth your while compressing it. If the ratio is greater than 20 per cent the data is worth saving. The following line will achieve this:

```
SAVE "squashed",b,&4000,length
```

Of course, having all this wonderful data saved to disc or tape is quite useless without a routine to reverse the compression. Listing II takes data starting at hex address &4000 and reconstitutes the picture. If the data is not at &4000 some very pretty and very useless screen designs will result.

Compile time

As a very special Easter gift to those of you *still* without assemblers – and this is not as may be suggested simply a way of padding out my article to get more money to pay off my student overdraft – I am including Listing III, a Basic hex loader.

How's the juggling coming along? I hope you've been practising and can now keep the balls up in the air for at least 15 minutes. If so, you are ready to make your first social debut – overnight you can become the most popular person in the neighbourhood.

Pick a good night. If you intend to be outside, make sure it is not too windy; if you are indoors, make sure the ceiling is high enough. The best way to start is to act naturally. Casually reach into your pocket during a conversation and produce your three objects. Then, still talking as though nothing untoward is going to happen, begin your act.

If all goes well, you can finish your short display, place the objects back in your pocket and make your farewells. The look of open-mouthed awe on the faces of those around you will be amazing.

Warning: If you should happen to drop an object or, worse still, throw it somewhere by mistake, on no account attempt to retrieve it. Believe me, it's safer where it is. Simply place the remaining objects in your pocket, glance at your watch and say, "Gosh, look at the time – I'm late for my therapy".

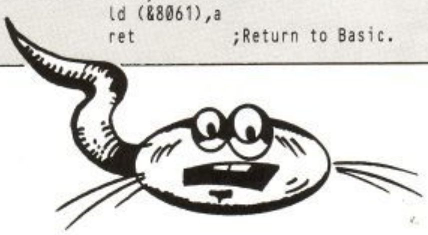
Ahem. So what use can be made of the routines we've discussed? Well, if you are writing your own art package and want to cut the time the user will spend saving and loading pictures – compression is one solution. If you are writing a graphics adventure program and want to store as many pictures in ram or on disc as possible – again, compression will be quite useful.

The compression routines will even work on data other than screen designs – any section of computer memory can be squeezed in the same way. It depends on the complexity of the data how successful the compression will be.

See you next time, and remember, if you teach the cat to ride a motorbike you must be prepared for some very scared looking mice around the house.

```
        org &8000     ;Suggested start of code.

        ld hl,&c000   ;Start of screen memory.
        ld de,&4000   ;Where data is to be stored.
        ld a,(hl)     ;Get first value.
loop1   ld (de),a     ;Store value.
        inc de
        ld b,a        ;B = current value and
        ld c,1        ;C = occurrence (or length).
loop2   inc hl
        ld a,h
        or l
        cp 0
        jp z,exit     ;Check for end of screen.
        ld a,(hl)
        cp b          ;Is value the same as last time?
        jp nz,quit    ;If not the same type, goto "quit"
        inc c         ;otherwise, length=length+1.
        ld a,c
        cp 1
        jp nz,loop2   ;If length<=256 then continue.

        ld a,0        ;C is now 257,
        ld (de),a     ;so store 256 as length
        inc de        ;and continue as though
        ld b,a        ;a new value has been found.
        jp loop1      ;NB 0 = 256 mod 255

quit    ld b,a        ;Remember new value.
        ld a,c        ;Store length.
        ld (de),a
        inc de
        ld a,b        ;Recall new value and
        jp loop1      ;go back to finding length.

exit    ld a,c
        ld (de),a     ;Store very last length.

        ex de,hl      ;Now calculate total length
        ld bc,&4000   ;of data by subtracting &4000
        scf           ;from last storage address,
        ccf           ;and store in address &8060
        sbc hl,bc     ;and &8061.
        ld a,l
        ld (&8060),a
        ld a,h
        ld (&8061),a
        ret           ;Return to Basic.
```

Listing I: The compression routine



```
      org &8045     ;Suggested start of code.

      ld hl,&4000   ;Start of data.
      ld de,&c000   ;Start of screen memory.

loop3 ld a,(hl)     ;Get value
      inc hl        ;and
      ld c,a        ;place in C.
      ld a,(hl)     ;Get number of times it occurs
      inc hl        ;and
      ld b,a        ;put it in B.
loop4 ld a,c
      ld (de),a     ;Poke value to screen.
      inc de
      ld a,d        ;Check to see if the end
      or e          ;of the screen has been
      cp 0          ;reached.
      ret z         ;Yes it has, return to Basic.
      djnz loop4    ;No it hasn't, continue.

      jp loop3      ;That's the end of that
                    ;value, let's try another.
      end
```

Listing II: Decompressing a screen

```
10 ' Screen compress.
20 ' Basic loader and demo.
30 '
40 MEMORY &3FFF
50 FOR a=&8000 TO &805D
60 READ a$:b=VAL("&"+a$)
70 POKE a,b
80 NEXT
90 MODE 1
100 FOR t=1 TO 50
110 x=INT(RND*600)
120 y=INT(RND*360)
130 i=INT(RND*4)
140 ORIGIN 0,0,x,x+40,y,y+40:CLG i
150 NEXT
160 CALL &8000
170 CLS:PRINT "Screen compressed."
180 PRINT:PRINT "Press spacebar."
190 WHILE INKEY(47):WEND
200 CALL &8045
210 lngth=PEEK(&8060)+PEEK(&8061)*256
220 ratio=100-(length/16384)*100
230 PRINT "Compression:";ratio;"%"
240 DATA 21,00,c0,11,00,40,7e,12,13
250 DATA 47,0e,01,23,7c,b5,fe,00,ca
260 DATA 30,80,7e,b8,c2,28,80,0c,79
270 DATA fe,01,c2,0c,80,3e,00,12,13
280 DATA 47,c3,07,80,47,79,12,13,78
290 DATA c3,07,80,79,12,eb,01,00,40
300 DATA 37,3f,ed,42,7d,32,60,80,7c
310 DATA 32,61,80,c9,20,20,21,00,40
320 DATA 11,00,c0,7e,23,4f,7e,23,47
330 DATA 79,12,13,7a,b3,fe,00,c8,10
340 DATA f6,c3,4b,80
```

Listing III: For poor people