Snow fun

John Kennedy tries to inject some early seasonal cheer.

As winter closes in and the season of goodwill, presents, alcohol and TV repeats approaches (and only seven weeks to go before Christmas) I thought I would write a little program to delight all those of you with an interest in silly graphic routines. The machine code I proudly present you with will scatter any display you wish to put on the screen with liberal quantities of snow. What is more, you have a choice of two different types: synthetic snow or the real, settling, white stuff.

Synthetic snow is the sort of snow you see on television. You know the kind I mean; actors come in from a sudden snowstorm covered from head to foot in a white powder which has completely vanished by the next scene. (This is not to be confused with dandruff.)

The real stuff will settle into huge heaps that have to be cleared away in the morning with a quick CLS or MODE statement.

As a special treat, not only have we listed the customary assembly language listing for you to type into the assembler you are hoping Santa will bring, but we have even given you a complete Basic program that will produce a pleasant little display for you.

The Snowstorm program is a perfect example of modularisation and the speed of machine code over Basic. Writing a similar program in Basic would be a waste of time, as the snow would seem to be crawling around like bacteria instead of drifting smoothly

```
10 ' Christmas Machine Code Listing  [F1]
20 '   [F2]
30 MEMORY &7FFF [3A]
40 DEFINT a-z [BD]
50 t=0:a=&8000:FOR b=1 TO 145:READ c$:d=VAL("&"+c$) [A1]
60 POKE a,d:t=t+d:a=a+1:NEXT b [82]
70 READ s:IF t<>s THEN PRINT "Error in data !":END [F1]
80 MODE 1 [EB]
90 INK 3,23:INK 2,26:INK 1,6 [14]
100 FOR l=0 TO 16:READ a$ [48]
110 LOCATE 1,1:PRINT a$ [78]
120 FOR x=0 TO 16 [41]
130 FOR y=398 TO 382 STEP -2 [64]
140 PLOT (l*40)+(x*2),(y-380)*8,TEST(x,y) [E4]
150 DRAWR 0,16 [5D]
160 NEXT:NEXT [0A]
170 NEXT [3B]
180 CALL &8000 [6E]
190 DATA CD,14,80,CD,20,80,CD,2B,80,3E,42,CD,1E,BB,C0,CD [F7]
200 DATA 19,BD,18,F2,21,80,C7,06,50,3E,FF,77,23,10,FC,C9 [1B]
210 DATA 11,93,80,06,32,CD,58,80,10,FB,C9,11,93,80,06,32 [53]
220 DATA CD,36,80,10,FB,C9,1A,6F,13,1A,67,1B,3E,00,77,E5 [41]
230 DATA CD,26,BC,E7,FE,00,20,0B,F1,3E,01,77,7D,12,13,7C [F4]
240 DATA 12,13,C9,E1,3E,FF,18,00,CD,63,80,7D,12,13,3E,C0 [C6]
250 DATA 12,13,C9,F5,C5,D5,ED,4B,91,80,2A,91,80,CB,25,CB [6D]
260 DATA 14,09,44,4D,CB,25,CB,14,55,CB,25,CB,14,CB,25,CB [A6]
270 DATA 14,09,44,4D,62,2E,29,B7,ED,42,22,91,80,D1,C1,F1 [C7]
280 DATA C9,16249 [49]
290 DATA H,a,p,p,y," ",C,h,r,i,s,t,m,a,s,!," " [F2]
```

The Basic program (above) and the assembly listing (below and continued on pages 49 and 50).

```
; How the program works:

;set up the screen.

;set up initial positions of the flakes.

;repeat until ESCAPE pressed;

;            for all flakes;

;                      erase old flake

;                      update position (ie move it down)

;                      check for obstical in the way

;                      draw new flake

;            next flake

;

wait_frame      equ &bd19
```

```
test_key        equ &bb1e

down_a_line     equ &bc26


org &8000                                   ;Start Location of code


call set_screen

call set_table


.loop

        call move_all                       ;Move all the flakes.

        ld a,66:call test_key               ;Check for ESCAPE key & if pressed

        ret nz                              ;return to BASIC.

        call wait_frame                     ;Smooth out the movement.

jr loop



set_screen                                  ;Draw the line at the bottom

        ld hl,&c780                         ;of the screen.

        ld b,80                             ;80 bytes in one screen line.

        ld a,255

        ssl

                ld (hl),a:inc hl            ;poke line into place

        djnz ssl

ret


.set_table                                  ;Set initial positions of

        ld de,table                         ;all fifty snow flakes.

        ld b,50

        stl:

                call set_flake

        djnz stl

ret


.move_all                                   ;Move all the flakes, one at

        ld de,table                         ;a time.

        ld b,50

        mal:

        call move_flake

        djnz mal

ret


.move_flake                                 ;Move a single flake.
```

down the screen.

First we must decide on the number of flakes and how they will be moved. I have chosen to animate fifty flakes, for no particular reason. Instead of using an X and Y co-ordinate to describe their position on the screen, we will use their Screen Address. As you will no doubt be aware from previous discussions, the screen is simply an area of memory that also happens to appear on a visual display unit. By writing directly to this memory we influence what appears on the screen. Writing directly also speeds things up considerably and allows fifty snow-flakes to move with no discernable effort whatever.

A table of one hundred bytes is reserved to store all the screen memory address we will use; one of the first things the program must do is initialise this table, by making up random screen addresses. The very first thing the program does, however, is draw a line at the bottom of the screen to stop the snow drifting away and corrupting memory left, right and centre.

We have two main problems to overcome. The first is how to move the snow down the screen. How do we calculate the new screen address? The second problem is how to find out whether the snow has hit the ground or an obstacle.

The answer to the first problem has thoughtfully been provided in a routine which already exists in the computer's operating system. All we have to do is load the HL register pair with the old screen address, call the magic routine and voilà! The new screen address is magically calculated for us.

The second problem is of a slightly more taxing nature. Obviously to decide if the flake has hit something we will have to examine the screen memory at its address. Unfortunately, this presents us with a further small problem. Examining the memory by attempting a LD A,(HL) instruction will not work. This is because it just so happens that a ROM is sitting right slap-bang over our screen memory! Tragedy! How can we read the screen RAM when all we can discover is the contents of an irrelevant ROM? Again the operation system comes to our rescue by providing a special routine

```
           ld a,(de):ld l,a:inc de           ;Get the screen address of the
           ld a,(de):ld h,a:dec de           ;snow flake,
           ld a,0:ld (hl),a                  ;and blank it out.

           push hl
           call down_a_line                  ;Move the screen address down.

           rst 4                             ;See text!
           cp 0:jr nz,hit                    ;Check for an obstical.
           pop af

           ld a,l:ld (hl),a                  ;Draw a new flake, and store
           ld a,l:ld (de),a:inc de           ;its new screen address.
           ld a,h:ld (de),a:inc de
   ret

   .hit
          pop hl:ld a,255                    ;Draw a splatted flake
          ;ld (hl),a                         ;See text!
   jr set_flake

   .set_flake                                ;Create a random start for the
          call random                        ;snow flake.
          ld a,l                             ;This is LSB of screen address.
          ld (de),a:inc de
          ld a,&c0                           ;This is MSB of screen address.
          ld (de),a:inc de
   ret

  .random                                    ;The Magic random number generator.

  push af:push bc:push de:ld bc,(seed):ld hl,(seed)
  sla l:rl h:add hl,bc:ld b,h:ld c,l:sla l:rl h
  ld d,l:sla l:rl h:sla l:rl h:add hl,bc:ld b,h
  ld c,l:ld h,d:ld l,&29:or a:sbc hl,bc:ld (seed),hl
  pop de:pop bc:pop af:ret:seed dw 0

  .table                                     ;Somewhere to store all the screen
  ds 100                                     ;addresses of the flakes.

  list:end
```

to do the job for us. This time the routine is a bit special and is not of the common or garden CALL & address variety. Instead it is what we in-the-know refer to as an RST or RESTART instruction. RESTART instructions are special 'definable op-codes'. To those of us who prefer speaking in English, this means that when the system engineer was designing the way in which the Amstrad CPC was going to work, he decided what would happen if one of these special RESTART instructions was executed.

These are, of course, completely different to all the other instructions that the Z80 uses; they are all totally pre-programmed and can on no account ever be changed, not even by me.

There are several of these RESTARTS, imaginately numbered from 0 to 7. The codes that you would type into your assembler are therefore RST 0 to RST 7. (Note! Some assemblers are a bit picky about this and may try to number the RESTARTS 0,8,16,24,32,40,48,56.) Anyway, the RST instruction that we are going to use is number 4, and this does a LD A,(HL) with all the ROMs switched off.

We may return to RESTART number 6, which has no special function and which is available for us to experiment with.

So we have now solved the problem of looking to see if the snowflake has hit anything. If it has not, then we can continue as before; if, however, something has got in the way, we jump out of the loop and into a routine called HIT, in order to decide what to do next. One instruction here is commented out, and it is this one that determines whether you are using real or synthetic snow. When the line is in place, it will draw a heap of snow on whatever has been hit; otherwise the flake will completely vanish. The choice is yours – unless you do not have an assembler, of course, in which case you are stuck with the fake snow supplied in the Basic listing.

Try typing in the Basic program and loading in some drawings of your own. Using snow on some digitised pictures will produce some extremely strange results!

See you next month! ●