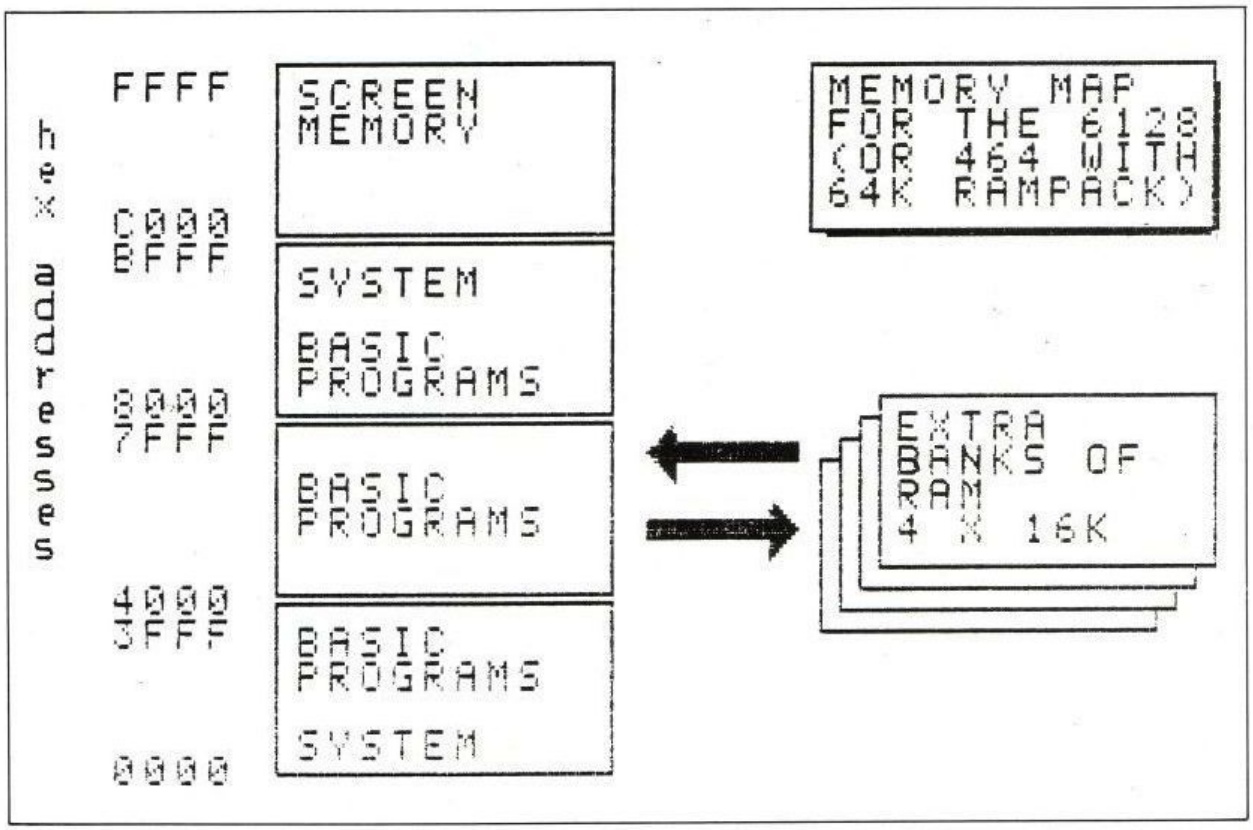
Welcome to Auntie John's Summer Column

This month we look at the extra memory CPC6128 owners have. Most of the time the memory is unused by the Basic and machine code programmer and it rests on the sidelines doing nothing. That is a pity because computer memory is expensive and potentially very useful. Although it is many times cheaper than it was 10 or so years ago, world shortages recently have pushed up its price again. The moral is if you have it, use it.

Deep inside your beloved grey plastic box is a tiny chip of silicon encased it a little plastic packet and called a Zilog Z-80 microprocessor. The Z-80 is now getting rather old, though they have been saying that for years, and is what is referred to as an 8-bit microprocessor. The eight bits does not mean that inside it are eight pieces but that the information is moved round inside the computer in the form of packets of data eight bits wide.

Remembering that a bit is an ON or OFF signal, you can imagine that eight separate wires are needed to carry information from one part of the computer to another. To store all that information a 16-bit address system is used. If you know binary arithmetic you will realise that 16 bits will give 16,384 * 16,384 separate addresses and so the Z-80 can access 64K of memory directly, where 1K equals 1,024 bytes and one byte equals eight bits.

Experienced programmers will laugh at this point, since everybody knows about bits and bytes. They might, however, pause for a second when they realise that the CPC6128 manages to break a few of the rules they hold so dear. The Z-80 was de-signed to be able To use 64K



Summer is here at last, school is almost out and John Kennedy has slipped into something casual to examine the extra memory on CPC6128s.

of memory and in a CPC6128 there is more than 170K of memory.

The CPC range was designed to make use of a technique called bank switching, by which the necessary portion of memory is switched on only when it is needed and so the 64K maximum rule is never violated. The 170K memory exists but the Z-80 is allowed to look only at certain parts of it at a time.

The design of the CPC allows yet more memory to be provided by ROMs and EPROMs. They overlap the bottom and top of the possible memory map and contain the operating system, Basic and, if disc drives are present, the code needed to use them. Other programs such as word processors, assemblers and even other languages can also be stored in ROM and switched in when needed.

This sounds complicated but all this is completely transparent to the user, to whom it appears that the programs in ROM are like any other program, except that they take up no RAM and they load instantly. The switching of ROMs, EPROMs and RAMs is done from another of the chips in the CPC, called the gate array.

The firmware routine situated at address &BD05 will allow us to choose which 16K RAM bank goes where – figure 1. This machine code does not exist on a 464 or 664, so even if extra memory packs are pluged into those machines the CALL &BD05 routine will not work. Unless, of course, someone supplies you with the necessary code. Some memory packs code with suitable code.

The extra 64K of memory on the 6128 is split into four banks, each 16K in size. There are several configurations all the memory banks can take but the ones on which we concentrate on are those which switch the memory in and out of the addresses from &4000 to &7FFF. This block is in a good place to shift in and out and will form the basis for the subroutines which follow.

Moving round the banks of RAM can be a nasty business, especially if you are trying to execute a program in one at the time. The computer would be running a program when the program vanishes and the computer becomes paranoid. The hardware also expects the screen display to occupy one of two places and will refuse to agree to any other locations, so take care when experimenting. You will not break anything but you can easily get into a situation where only a total power-down will save the day. It is interesting that the contents of the extra memory are not re-set whenever the ESC — SHIFT — CONTROL keys are pressed, so any picture or other data will not be affected by a minor crash – but do not take it for granted.

So what will we do with all this extra memory? The following machine code will allow you to store and retrieve screen images to and from the banked RAM. You will also be able to look at a miniature version of each stored screen so that you

can see what is happening. The machine code makes use of the RSX system, so you will not have to remember hex addresses. As an act of unheard generosity, I will even give you the hexcode in a Basic program.

Program 1 is the assembly language listing. The material at the start is all the machine code needed to set up the RSX table. Notice how each routine extracts some parameters – the numbers which follow the RSX – and uses them to choose the bank of RAM in question. Each RSX must check that one, and only one, parameter is supplied and that the parameter is in the desired range. If this is not the case the routines return to Basic without doing anything; you might

Listing 1 – The assembly language listing.

```
;Assembly Language Program to Store and Fetch screens
;from the extra memory on a CPC6128.

log_rsx equ &bcd1
next_line equ &bc26
rambank equ &bd5b

        org &8000

        ld bc,commandtable
        ld hl,buffer
        call log_rsx
        ret

.buffer ds 4

.commandtable
        dw nametable
        jp storescreen
        jp fetchscreen
        jp miniscreen

.nametable

        db "STORESCREE","N"+&80
        db "FETCHSCREE","N"+&80
        db "MINISCREE","N"+&80
        db 0

.storescreen

;Store screen display in extra memory.

        cp 1            ;Return to BASIC if there isn't exactly
        ret nz          ;one parameter.


        ld a,(IX)       ;Make sure the parameter is in the
        cp 5            ;range 1-4, and if not
        ret nc          ;return to BASIC.
        cp 0
        ret z

        add 3           ;Adjust the number, and call the
        call rambank    ;firmware.

        ld hl,&c000     ;The start of the screen memory.
        ld de,&4000     ;The start of the Ram Bank.
        ld bc,&4000     ;The length of the screen.
        ldir            ;The magic machine code instrucion LDIR!

        ld a,0
        call rambank    ;Put back the original RAM bank.
        ret             ;and back to BASIC we go..


.fetchscreen

;Copy screen image from extra RAM to the screen.

        cp 1            ;Check for one parameter.
        ret nz

        ld a,(IX)
        cp 5
        ret nc
        cp 0
        ret z           ;Check for range 1 to 4.

        add 3           ;Adjust it and use the firmware
        call rambank    ;to bank in the next memory.

        ld de,&c000     ;Move data FROM this address
        ld hl,&4000     ;TO this address
        ld bc,&4000     ;THIS amount,
        ldir            ;with this instruction.

        ld a,0
        call rambank    ;Put original memory back.
        ret

.miniscreen

;Draw a miniture version of the image held
;in banked memory.

        cp 1
        ret nz                  ;Check for a parameter.

        ld a,(IX)
        cp 5
        ret nc                  ;Return if out of range.

        cp 0                    ;The parameter value of ZERO
        jr z,special            ;is a special case.

.mini
        add 3


        push af
        call rambank            ;Switch  on the RAM bank.
        pop af

        ld hl,&c000             ;Set a different screen position
        cp 4                    ;for each Banked RAM image to be
        jp z,shrink             ;drawn at.

        ld hl,&c028
        cp 5
        jp z,shrink

        ld hl,&e3c0
        cp 6
        jp z,shrink

        ld hl,&e3e8


.shrink
        ld b,100                ;The height of the mini-image.
        ld de,&4000             ;The address stored image.
        .loop1
        push hl                 ;Store the start addresses.
        push de
        ld c,40                 ;The width of the mini-screen.

        .loop2
                ld a,(de)       ;Copy the data from the stored
                ld (hl),a       ;image to the actual screen..

                inc hl
                inc de:inc de   ;Update the screen addresses.

                dec c
                ld a,c
                jr nz,loop2     ;End of 'width' loop

        pop hl                  ;Cunningly read DE into HL
        call next_line          ;and move it down a line
        call next_line          ;or two and
        ex de,hl                ;then swap it back.

        pop hl
        call next_line  ;Move HL to the next line.

        djnz loop1              ;End of 'height' loop

ld a,0                          ;Put the original
call rambank                    ;block of RAM back in place.
ret

special

;Draw all the RAM banks, 1 to 4.

ld a,1:call mini
ld a,2:call mini
ld a,3:call mini
ld a,4:call mini

ret
```

Listing 2 - The Basic HexCode loader program.

```
      10 ' Machine Code Hex Loaded
      20 ' SAVE program before running
      30 '
      40 MEMORY &7FFF
      50 s=0
      60 FOR a=&8000 TO &80E7
      70 READ b$:b=VAL("&"+b$)
      80 POKE a,b
      90 s=s+b
      100 NEXT a
      110 IF s<>&6889 THEN PRINT "Error in data."

120 CALL &8000
130 PRINT "RSXs installed."
140 NEW
150 DATA 01,0E,80,21,0A,80,CD,D1,BC,C9,8C,9F,0E,80,19,80
160 DATA C3,3A,80,C3,5C,80,C3,7E,80,53,54,4F,52,45,53,43
170 DATA 52,45,45,CE,46,45,54,43,48,53,43,52,45,45,CE,4D
180 DATA 49,4E,49,53,43,52,45,45,CE,00,FE,01,C0,DD,7E,00
190 DATA FE,05,D0,FE,00,C8,C6,03,CD,5B,BD,21,00,C0,11,00
200 DATA 40,01,00,40,ED,B0,3E,00,CD,5B,BD,C9,FE,01,C0,DD
210 DATA 7E,00,FE,05,D0,FE,00,C8,C6,03,CD,5B,BD,11,00,C0
220 DATA 21,00,40,01,00,40,ED,B0,3E,00,CD,5B,BD,C9,FE,01
230 DATA C0,DD,7E,00,FE,05,D0,FE,00,28,48,C6,03,F5,CD,5B
240 DATA BD,F1,21,00,C0,FE,04,CA,AD,80,21,28,C0,FE,05,CA
250 DATA AD,80,21,C0,E3,FE,06,CA,AD,80,21,E8,E3,06,64,11
260 DATA 00,40,E5,D5,0E,28,1A,77,23,13,13,0D,79,20,F7,E1
270 DATA CD,26,BC,CD,26,BC,EB,E1,CD,26,BC,10,E5,3E,00,CD
280 DATA 5B,BD,C9,3E,01,CD,8B,80,3E,02,CD,8B,80,3E,03,CD
290 DATA 8B,80,3E,04,CD,8B,80,C9
```

like to add a routine to print an error message.

Program 2 in the Basic program will create the RSXs and then nuke itself so save it before running. When all is working you will have three RSXs logged on to your system - STORESCREEN, FETCHSCREEN and MINISCREEN. They all need the number of a RAM bank to use a number from 1 to 4. MINISCREEN is special because it also can take the number from 1 to 4. MINISCREEN is special because it also can take the number 0 after it. Try it and see what it does.

Program 3 is a short demonstration you can run after getting the RSXs to work. You must save a screen to disc or tape with which the program can load or, alternatively, you could write a short routine using DRAW and PLOT to put squiggles on the screen.

So the extra memory is put to good use. If you want a good exercise in programming, try using the RSXs to supply you with some pull-down menus. Pull-down menus are menus which appear on the screen, covering anything which was there previously. Whenever a choice is made from the menu they disappear and anything underneath is re-drawn. You can use STORESCREEN and FETCHSCREEN to protect the contents of the screen under the menus.

Listing 3 - A short Basic demonstration program.

```
10 ' Example Basic Program
20 'RSX's MUST already be defined!
30 MODE 1
40 LOAD "screen",&C000:'A screen display of your own design.
41 REM Preceed name with ! if using a tape (spit) system
50 FOR a=1 TO 5
60 FOR b=1 TO 4
70 |STORESCREEN,b
80 NEXT b
90 |MINISCREEN,0
100 'try changing 0 to 1 above
110 NEXT a
```