Hot Shots

```
CALL mc,&802F,65535    Emulating DOKE &8000,65535
19 bytes of code used.

                : 2 parameters are being passed (&8000 & 65535)      :
CP 2            : Register A should yield the number of parameters. :FE02
                :                                                    :
RET NZ          : If not 2, we cannot continue and must return!      :C0
                :                                                    :
LD L,(IX+2):  The first parameter,&8000, is available at IX2/3 :DD6E02
LD H,(IX+3):  so get it and put it into HL.                    :DD6603
                :                                                    :
LD A,(IX+0):  The last parameter, 65535, is available at IX0/1 :DD7E00
                : so put its LSB into register A.                    :
LD (HL),A       : And thus into &8000.                               :77
                :                                                    :
LD A,(IX+1):  Now the MSB into register A.                     :DD7E01
INC HL          : Advance HL to point to &8001.                      :23
LD (HL),A       : And put MSB into &8001.                            :77
                :                                                    :
RET             : DOKE complete, so return to BASIC.                 :C9
```

The first task which must be done before any machine code routines can be tagged on to Basic is to find somewhere to put them. High-memory sites are the usual place for m/c bytes and on the Amstrad that means looking at locations near to &BFFF, since the top 16K (C000-FFFF) is occupied by both screen RAM and the Basic ROM. The top of the usable memory, HIMEM as its called, is nowhere near so high as &BFFF, because Basic needs workspace, the Z-80 processor needs a stack, and the tape – or disc – filing system likes to take 4K of RAM from time to time.

Also, if you are in the habit of defining dozens of little space invader characters, you could say goodbye to another 2K. To see how low HIMEM can get on your machine, re-set the computer and run the following program:

```
100 PRINT HEX$(HIMEM)
110 SYMBOL AFTER 0
120 PRINT HEX$(HIMEM)
130 OPENOUT "fred"
140 PRINT HEX$(HIMEM)
150 CLOSEOUT
```

The final HIMEM figure will be somewhere between &8EFB and &A2FB. The short routines used to illustrate this article will be placed between &8000 and &8E00 and HIMEM should be set to &7FFF, so that the routines lying above are protected from Basic. MEMORY &7FFF is the command needed to do that.

Both the RSX and the CALL use the same system to deliver information to the installed machine code routine and to receive back information from it. The system is flexible, efficient and even generous; 64 assorted numerical values and string characters can be inter-changed by any single RSX or CALL. The key to its efficiency lies in the use of a special stack set up by Basic and marked by the Z-80 IX register.

To show how it works here is a routine which loads a 16-bit number into two adjacent memory locations, the double POKE, sometimes referred to as a DOKE. The routine – figure one – is shown as a CALL from Basic. It can be changed into an RSX later.

From that example three rules are apparent. The number of parameters passed is transferred to the A register. The last parameter typed on the Basic line is pointed to by IX+0 and IX+1. Other parameters, if there are any, follow in pairs and in each pair the lower IX i.e., the "even" IX – holds the Least Significant Byte of information.

Values are passed back to Basic via an integer variable, which must be declared before the CALL is made. Double PEEKs are DEEKs – what else could they be? – and if a machine code routine is constructed to perform a DEEK, the value found in memory must be passed back to Basic. Figure two shows the DEEK code, as a simple CALL.

The DEEK code illustrates two more rules. The direction of travel of information is denoted by the @ symbol. Thus, CALL mc,A% sends the value contained in A% to the machine code routine, whereas CALL mc,@A% sends nothing from A% but accepts a machine code generated value into A%. For an @A% transfer, the IX pointers indicate indirectly where to place the computed value, which is assumed to be an unsigned integer.

A string can also be transferred and the format for both sending and receiving is A$="M":CALL mc,@A$. The IX pair involved at the machine code end of the transfer will produce a number which is the address of three bytes elsewhere in memory; it is those which describe the string A$. The first of those bytes is the length of the string and the next two bytes are the address of the string. Once the address of the string is determined, the machine code routine can examnine the string, copy it or alter it.

The RSX system, in its simplest interpretations, is a way of giving names to machine code routines so they do not have to be CALLed by their hexadecimal address. Any name of reasonable length can be used, although it should not be the same as one already used by Basic or by an extension ROM utility. The name is preceeded by a bar symbol – to be found on the shifted-@ key – so that it is recognised by the operating system as an RSX

```
A%=0:CALL &8042,&8000,@A%  Emulating A%=DEEK(&8000)
16 bytes of code used.

                : 2 parameters are being passed (&8000 & A%)         :
CP 2            : Register A should yield the number of parameters. :FE02
                :                                                    :
RET NZ          : If not 2, we cannot continue and must return!      :C0
                :                                                    :
LD L,(IX+2): The first  parameter,&8000, is available at IX2/3 :DD6E02
LD H,(IX+3): so get it and put it into HL.                     :DD6603
                :                                                    :
LD E,(IX+0): The last parameter, A%, has an address which can  :DD5E00*
                : be retrieved from IX0/1.                           :
LD D,(IX+1): DE now holds that address.                        :DD5601*
                : These last 6 bytes are not needed! See text.       :
                :                                                    :
LD A,(HL)       : This is a machine-code PEEK. It's the LSB.         :7E
LD (DE),A       : Put it into the LSB of A%.                         :12
INC DE          : Point at MSB of A%.                                :13
INC HL          : Advance HL for next PEEK.                          :23
LD A,(HL)       : Another machine-code PEEK. MSB this time.          :7E
LD (DE),A       : Put it into MSB of A%.                             :12
                :                                                    :
RET             : DEEK complete, so return to BASIC.                 :C9
```

and not taken as a typing error. Setting-up a single RSX command such as DOKE uses 20 bytes, excluding the code of the DOKE routine. Further RSX commands require an overhead of only a few bytes and they are mostly the name of the routine. Figure three shows the assembly listing for four RSXs.

It is not difficult to think of routines to add to Amstrad Basic and armed, with the Amsoft *Concise Firmware Guide* which has the addresses and information about all the ROM subroutines of interest to machine code programmers, there is many a winter evening's worth of coding to be done. The trouble is that there are a host of extremely interesting RSXs already developed and tucked into commercial ROMs which plug into the back of the Amstrad. ROM RSXs do not take precious bytes away from Basic and are always available, so they are useful commands to have.

ENDZONE

The two final examples are PAN ,x which moves *the contents* of the whole screen sideways and PAN ,0,y which moves the whole screen sideways – there is a difference. Suggested values for x are 1-80 and for y 1-29. The effects of PAN ,x last while a program is running ,but normality is restored when Basic returns with the READY prompt. PAN ,0,y on the other hand causes more permanent lateral shifts and the screen can be returned to normal only by issuing PAN ,0,16 or simply PAN on its own. Note the space between the RSX name and the first comma, that being the required syntax.

TWIST ,c,r,n looks at a character on the screen at row r and column c and rotates it n times through 90 degrees, which is useful for creating vertical writing for labelling the Y axis of graphs. The machine code does not check the legality of c and r and a co-ordinate outside the limits of a screen mode will have unpredictable effects. It also uses the eight-byte storage area used for CHR$(255), so avoid using that character as a user-defined graphic. The listing of figure four loads the code for all four RSXs and then deletes lines 20 to 80, which ensures that subsequent RUNs do not re-initialise the RSXs and so confuse the computer. The program goes on to demonstrate PAN and TWIST; what you do with your RSXs after that is entirely for you.

```
                        : The INITIALISATION code.
                        : ------------------------
LD BC,COMMANDTABLE      : Where the addresses of all the routines are held.
LD HL,BUFFERSPACE       : A 4-byte area of scratch-RAM.
CALL &BCD1              : Inform the operating system that the RSX exists.
RET                     : RSXs initialised, so return.
                        : 10 bytes of code so far.

BUFFERSPACE             : This is 4 bytes of storage.

                        : The COMMANDTABLE takes this form.
                        : ---------------------------------
NAMETABLE               : 2 bytes, the address of the RSX's names.
JUMP 1                  : 3 bytes in the form of C3 aa aa where aaaa is the
                        : address of the the first RSX routine.
JUMP 2                  : The jump for RSX routine number 2.
JUMP 3                  : And so on...
JUMP 4                  : The final jump in our example.
                        : 14 bytes

                        : The NAMETABLE itself.
                        : ---------------------
DOKE                    : 4 bytes, the last having bit 7 set high.
DEEK                    : 4 bytes.
PAN                     : 3 bytes.
TWIST                   : 5 bytes.
00                      : 1 byte end of table marker.
```

```
10 REM DEEK, DOKE, PAN & TWIST
20 REM AMSTRAD RSXs R.SARGENT 1985
30 REM First load the machine-code
40 MEMORY &7FFF
50 FOR A=&8006 TO &80DD
60 READ D$:POKE A,VAL("&"+D$):NEXT A
70 CALL &8006:REM INITIALISE RSXs
80 DELETE 20-80
90 REM Demonstration of PAN & TWIST
100 MODE 1:BORDER 12
110 FOR X=1 TO 130
120 PRINT "ABCDEFGH";:NEXT
130 FOR N=1 TO 30
140 FOR X=1 TO 40:NEXT
150 |PAN ,N:NEXT
160 FOR N=16 TO 29
170 FOR X=1 TO 40:NEXT
180 |PAN ,0,N:NEXT
190 FOR N=29 TO 16 STEP-1
200 FOR X=1 TO 40:NEXT
210 |PAN ,0,N:NEXT
220 FOR R=1 TO 25
230 FOR C=1 TO 40
240 |TWIST ,C,R,1:NEXT
250 NEXT

900 DATA 01,10,80,21,02,80,CD,D1,BC,C9,1E,80,C3,2F,80,C3
901 DATA 42,80,C3,52,80,C3,7A,80,44,4F,4B,C5,44,45,45,CB
902 DATA 50,41,CE,54,57,49,53,D4,00,FE,02,C0,DD,6E,02,DD
903 DATA 66,03,DD,7E,00,77,DD,7E,01,23,77,C9,FE,02,C0,DD
904 DATA 6E,02,DD,66,03,7E,12,13,23,7E,12,C9,B7,28,21,FE
905 DATA 02,28,04,EB,C3,05,BC,7B,FE,01,D8,FE,1E,D0,B7,C6
906 DATA 1E,F5,01,00,BC,3E,02,ED,79,01,00,BD,F1,ED,79,C9
907 DATA 3E,2E,18,ED,FE,03,C0,3E,FF,CD,A5,BB,D0,22,00,80
908 DATA D5,DD,6E,02,DD,66,04,2D,25,E5,CD,93,BB,CD,2C,BC
909 DATA ED,5B,00,80,CD,56,BC,E1,D1,43,CD,BB,80,10,FB,E5
910 DATA CD,78,BB,D1,E5,EB,24,2C,CD,75,BB,3E,FF,CD,5D,BB
911 DATA E1,CD,75,BB,C9,E5,2A,00,80,F5,C5,0E,08,E5,06,08
912 DATA CB,3E,17,23,10,FA,E1,F5,0D,20,F2,0E,08,09,41,F1
913 DATA 2B,77,10,FB,C1,F1,E1,C9
```