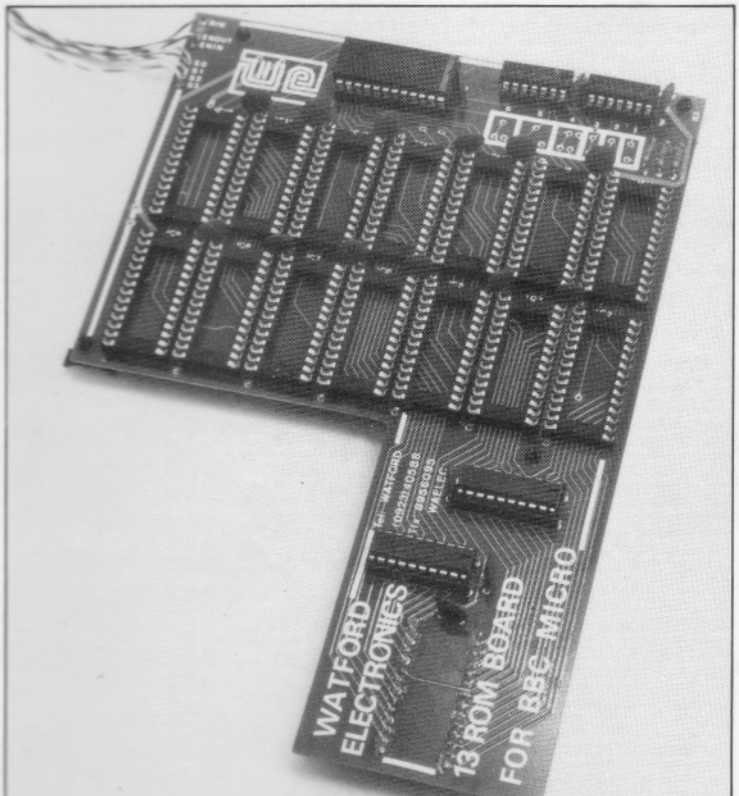
BBC MICRO ROM PAGING SYSTEM



Watford Electronics

THE BBC MICRO ROM PAGING SYSTEM

INTRODUCTION

It is well known that ROMs may be plugged, into the BBC micro to add new languages, filing systems and other special applications programs, The protocols through which these ROMs are called by the operating system are complex and difficult to discover for yourself. The ROM paging system is very complex, a high level of competence in machine code programming will be required to write a ROM. In the course of writing the Watford Electronics DFS, we have discovered the meaning of the most important entries. Whilst we believe the information contained to be accurate we cannot be sure of any of the meanings we have attached to the various calls so it is up to you to ensure that any ROM call you use actually performs as required.

This document only applies to OS 1.0 and higher. It concerns the paged ROM sockets. There are four of these in the standard BBC, on the right hand side under the keyboard. These sockets are arranged in the memory map as 16k units from &8000 to &C000. Any one ROM may be selected at any one time by the operating system which is mapped from &C000 to &FFFF.

The OS supports 16 sockets and the Watford Electronics 13 ROM board allows all 16 to be utilised. The sockets are generally used to contain languages such as BASIC or FORTH and filing systems such as the DFS. If you are intending to write a ROM of some sort for the BBC, it is strongly recommended that you invest in a system of RAM in the &8000 area as offered by the Watford Electronics 13 ROM board.

This allows a 'ROM' to be written into this extra piece of RAM so that modifications can be made by simply reassembling the code, speeding up debugging considerably. Attempting to write a ROM directly into EPROMs, making a modification, blowing a new EPROM, trying it out etc. is only suitable for very simple projects.

ROM RECOGNITION

The first group of bytes in the ROM must be in a specific format for the operating system to recognise the ROM. When BREAK is pressed the OS scans through all 16 ROM sockets examining the contents for legal ROMs. If a ROM is of the correct format then its ROM type byte is stored in a table from &2A1. The address of this table can be found from the operating system with OSBYTE 170 for compatibility with future operating systems.

The format of the ROM is:

Offset (from &8000)		Size (in bytes)
0	JMP language	3
3	JMP service	3
6	ROM type	1
7	Copyright offset	1
8	Version number	1
9	Title string	as required
	00	1
	Version string	as required
	00	1
	'('	1
	'C'	1
	')'	1
	Copyright string	as required
	00	1
	Tube relocate address	4

WHAT ALL THESE MEAN

JMP language

Should contain JMP XXXX to go to the entry of a language in the ROM. All three bytes should be zero if there is no language entry. Thus the Basic ROM contains 4C 1F 80, meaning JMP &801F.

JMP service

Should contain JMP XXXX to go to the service entry processor in your ROM. Service calls are used to pass commands to the ROM. Put three zeros if not needed, however, only the Basic interpreter is allowed to have no service entry.

ROM type

Identifies the type of ROM, either service or language. The relevant bits are:

S L T K 0 0 1 0

Set S to identify a service entry
Set L to identify a language entry
Set T to indicate that a valid tube relocation address exists
Set K if the ROM supports soft key expansions for the Electron
Bit 1 must be set, though existing operating systems do not use it!

It is common for what is apparently a language ROM to be a service ROM as well.

Copyright offset

Points to the '00' immediately prior to the copyright string (C)...., relative to the start of the ROM i.e. &8000 + offset = CHR$(0).

Version number

This is just an 8 bit number held internally for your own use. It doesn't have anything to do with the Version string.

Title string

Identifies the ROM. The operating system prints this out when the ROM is entered through its language entry.

Version string

The version number of the ROM as an ASCII string. This is completely optional as it is never output by the OS.

Copyright string

It is vital that this string starts null,'(C)'. This is what the OS checks for when detecting ROMs. Look at a ROM to see how it should be done. Also ensure that the copyright pointer points to the null (00).

Tube relocate address

Address to which the code will be moved when sent down the tube to the second processor. This does not relocate the code, it simply shifts it to the address given. Thus you will usually have a relocate address of &8000, otherwise your ROM will have to have been specially assembled to run at a different location such as &B000.

This first block is followed by the rest of the code itself. The two entry points should come into the main code and interpret the various entries possible. Examination of the header part of an existing ROM may prove explanatory, do not however that Basic is different to any standard ROM.

AVAILABLE MEMORY SPACES

The memory map of the BBC is allocated as follows:

00-8F Language workspace

A0-A7 NMI memory, may be used only when NMI is under your control.
A8-AF Used by filing systems when processing * commands only.
B0-BF Filing system temporary workspace, may be destroyed between calls and should not be used for IRQ.
C0-CF Filing system workspace which will remain between calls until the workspace area is taken from you by a service call.

D00-D5F NMI routine should be copied to here on entry. NMI enters &D00. If there is insufficient room for your NMI routine then write directly to the ROM latch at FE30 to enter your ROM and process the NMI, then re-write the ROM latch with the contents of &F4 to restore the original ROM in the space.

E00-... The main workspace area shared by all service ROMs in the machine. This area is grabbed by ROMs as required by a service call. (Service calls are explained later). This is in a fixed place (from &E00) and its length is set by a call to all the ROMs at Break. Any ROM can take control of this area which is used for such things as file buffers and general workspace.

Private. The private workspace area can be found by referencing the table &DFO,rom number for the base page of your private workspace. This area is always yours and its size is set by a service call. This area of memory is used to store information that must not be corrupted when the shared work space area has taken by another ROM, such as which files are currently open.

OSHWM-..The remaining memory above OSHWM is available to language ROMs. These also have various parts of the first block available, detailed in the User Guide.

LANGUAGES

A ROM may either be a language ROM-or a service ROM--- Languages are programs like Basic or wordprocessors that call up other ROMs to handle hardware 'such as disc drives. Any program that is not dependant on another language should be written as a language. Many service ROMs contain parts that are strictly a language. The Watford Electronics DFS for instance treats the Disc Sector Editor as a language, though the rest of the commands are still service calls. A service ROM handles the machine's hardware -devices directly, something that languages cannot do for tube compatibility.

A language is entered through *FX142. The syntax is *FX142,rom number. The rom number being the number of the paged ROM socket from 0 to 15. Thus the user may type *FX142,13 to enter the language in ROM number 13.

The normal way for a ROM to be entered is to type *FRED, where FRED is the word you have decided to recognise to enter your ROM with. The command *FRED must be recognised by the service entry in your ROM which will then use OSBYTE 142 to enter the ROM as a language properly, using the ROM number stored in &F4, also in the X register on entry. Thus your language ROM must have a service entry to be recognised by a * command. This service entry then calls up the language. This rather contorted entry method is necessary to function correctly down the tube, so it is worth using correctly. All ROMs must in fact have a service entry except BASIC. This is because the command *BASIC scans through the ROMs until it finds the first ROM with no service entry and then enters that as a language.
When your ROM is entered as a language you will be entered at the language entry point, having had your title string output. The accumulator contains an action code.

The normal entry is with A=1, in which case the language should simply start running.

If A=2 or A=3 then a soft key expansion has been requested by an Electron, this will only happen if bit 4 of the ROM type byte is set. For A=3 return X=Key no., Y=length of new key definition. For A=2 pass out the bytes for the key expansion one at a time in Y.

If A=0 on entry then the system is convinced that there are no languages fitted in the micro, this call should only be intercepted by the Tube ROM.

LANGUAGES AND THE TUBE

It is essential that languages are written so that they can run across the tube. At BREAK you may get thrown across the tube and be expected to run on the other side. The whole principle of writing a language is to use the OSBYTE, OSWORD etc. calls rather than access the I/O processor directly. Languages such as BASIC can access the main memory directly for storing the program. but cannot change mode by directly poking to the 6845 for instance.

Thus commands such as the sector editor of the Watford Electronics DFS are treated s languages by the system and so will be sent across the tube. Thus the sector editor loads and saves each sector through OSWORD &7F. Any code to access some part of the I/O machine's memory must not simply consist of a JSR to another part of your ROM, it must call the operating system with an unallocated °SWORD or OSBYTE number of your choice, which will then be passed out to all the ROMs in turn when you will grab it back. execute it, and then return through the OS to your original code.. This may seem confusing. but it really consists of writing your primitive command levels to be accessed through service entries such as OSWORD and then writing the language to use OSWORD etc.

The tube system will only access ROMs through the service entry in the I/O processor, thus a service ROM such as the DFS can access the hardware directly as you can be sure that you are running in the I/O processor, and not executing the copy of this code that may running your language in the tube machine.

Just to confuse things even more the tube system does not check processor type, so your-language may be given to a 16032, you cannot be held responsible for the results so don't worry about it.

THINGS LANGUAGES MUST DO

You must execute CLI (clear interrupt disable bit), or the OS will not work.

You must also set the BRK vector at &202 to handle errors as required. This may not be necessary on very simple ROMs but errors can be generated by the filing system when doing very simple operations such as printing a character.

The memory available for your use is from &400 to &800, from 00 to &8F, together with the main memory from OSHWM to the bottom of the display. Get both of these figures from the operating system for tube compatibility, these will give appropriate values for the tube processor. When accessing memory you may use these main memory areas directly as languages are intended to use the tube processor memory for storing programs etc. However do not use any I/O processor or OS specific locations.

SERVICE ROMS

The most common service ROMs are filing systems. These ROMs can only take * commands, OSBYTE and OSWORD calls etc. The service ROM will not be used on the wrong side of the tube and so can directly access I/O processor memory. It is quite normal for a ROM to be both a service ROM and a language ROM. In this case the hardware access is done by the service section.

when a service is handed to a ROM, A contains the call number, X the ROM number and Y a parameter if needed.

A service call can be initiated with *FX143, call number, parameter. However usually service calls are generated by the OS. If a ROM generates a service call it must use *FX143 to enter the OS and scan the ROMs. When entered, &F4 always contains the ROM socket number that you are plugged in to; this is also in the X register.

Call Number	Description
00	NOP
01	Work space area required
02	Private space required
03	Auto-boot
04	Command not recognised by OS
05	Interrupt not recognised by OS
06	BRK has occurred
07	unknown OSBYTE
08	Unknown OSWORD
09	HELP information
OA	Claim main work space
OB	NMI no longer being used
OC	Request to use NMI
OD	Initialise *ROM
OE	Return one byte in tape format for *ROM
OF	Claim indirection vectors
10	About to close SPOOL and EXEC files
11	Character set about to explode
12	Intialise filing system
FE	Tube Post initialisation
FF	Main Tube initialisation

These various calls are issued either at BREAK or during operation. When offered a call you may either use it or ignore it. You must restore registers to their entry condition before returning. If you have used the call then set A=&00 so that the rest of the ROMs are sent NOP so that they ignore the call.

00

NOP

The call hs already been grabbed by another ROM which has set A=0. This call is to be ignored.

01

Set work space area

This is given at BREAK to determine the total amount of work space required. This is the space from &E00 up to the private space. This area of memory is shared with the other ROMs which can claim it from you at any time. The Y register enters containing the current top of the fixed area (main workspace from &E00 to the bottom of Private space). If this is high enough for your needs then leave it alone, or increment as required. Do NOT decrement this value, some other ROM may need more workspace than you. This memory is used by the DFS for storing the catalogue and for the file buffers etc. It remains your memory space which will not be corrupted by other ROMs until it is claimed by someone else.

02

Set private space required

This call is also sent at BREAK to set the size of the private memory area needed. This space is used to keep essential information when the main workspace is grabbed by another ROM. The DFS for instance will keep information on open files in this area when needed. If you need private space then STY &DF0,X to store the address of the base of your space in the private area table at &DF0. Then increment Y as needed and return, i.e. add on the number of pages of private memory you require to store essential information. You must now always use the value stored in the table for the start of your private space. The private workspace should never be corrupted by anything as no other ROM is allowed to touch it. However programmers have a habit of running Basic programs all over this area of memory. You must allow for this possibility when using this area
i.e. it should only be used when it is unavoidable. The private memory is after the main workspace.

03

Auto-boot

Each ROM is given the opportunity to start up when BREAK is pressed. Test to see if a key is pressed. If a key you recognise is pressed then start up otherwise return with the register contents as on entry.

If Y=0 on entry then you should do something with a file called !BOOT. The DFS can *LOAD, *RUN and *EXEC this file, another filing system may do something different.

04

Command not recognised by OS

If a command is typed that the OS cannot recognise then that command is passed to all the ROMs in turn until one is found to recognise it. At this point ROMs should only intercept commands that are independent of the filing system currently in use. Thus *DISC and *DUMP should be intercepted by the DFS, but *WIPE should not.

On entry Y contains an offset from a base address stored in (&F2) pointing to the line to be processed. This pointer has already been passed over any asterisks or spaces in the line before the command. You must compare the text with your command table. If you recognise the command then execute it and set A=0 before returning to the OS.

If the command is not recognised by any of the ROMs currently resident then the command is passed to the currently active filing system where the filing system dependant commands are dealt with.

05

Interrupt not recognised by Os

The OS has not recognised an IRQ and so passes it to the ROMs, you must check to see if the interrupt is of interest to you. If so process it before setting A=0 and returning. Note that you should return with an RTS NOT an RTI

06

BRK

A BRK instruction has been executed. This call happens before the indirection at &202 is used. BRK is used for error generation.

07

OSBYTE not recognised by OS

The OSBYTE parameters A, X, Y are stored in &EF,&F0,&F1. Check &EF and process if required. Return with A=0 to stop the call being sent to other ROMs. Store the new X,Y values in &F0, &F1 and restore X and Y.

08

OSWORD not recognised by OS

&EF,&F0,&F1 are used to store A,X,Y for the OSWORD call. Deal with as for unrecognised OSBYTE. Use OSWORD numbers less than 128 as negative OSWORD calls ae sent to the user vector instead. DFS uses &7D,&7E,&7F.

09

HELP information

*HELP has been typed. The rest of the line is pointed to by (&F2),Y as for call number 04. If the rest of the line is blank then output the ROM name and version number together with any subheadings to which the ROM will respond.

If there is text after the command then compare each word with the subheadings you recognise and give the information for the sub headings as required. You should respond to each word in turn.

It .is very important to leave the pointer to the HELP line at its original position or ROMs further down the sockets will get garbage.

0A

Claim main work space

When you wish to claim the use of the work space area whose size was set by call 01 you must execute OSBYTE 143 with X=&0A and Y=&FF. This is sent to the ROM that currently has the workspace which must then take all vital information out of the main work space and put it into their private space. The DFS will write all file buffers to disc and store information on open files in the private area etc.

Having claimed the main workspace you are also free to use the zero page workspace (&B0 to &CF).

You must keep marker bytes in your private work space to show when you are the selected filing system and when you have grabbed the workspace. If this is not done you could be selected as filing system without having the. workspace. This is very likely to crash the machine when the original workspace user is returned to, as this ROM will think that memory is intact. These marker bytes are vulnerable as they are in the area where people tend to load programs when they are running short of memory. If the contents are not sensible then grab the workspace just in case. It is perfectly normal to have the workspace but not be the selected filing system. If another ROM needs the space it will take it from you.

0B

NMI no longer being used

Whoever last .claimed the NMI from you is now giving it back. Y contains a ROM ID, if it is yours then NMI is being returned to you. If NMI is not being given to you then preserve all registers, otherwise set A=0. Do not use the filing system page zero memory during this call.

0C

Request to use NMI

OSBYTE 143 with X=&0C and Y=&FF to tell any ROM that is using NMI to stop generating NMIs. Thus you must turn your NMI hardware off on receiving this call. When the OSBYTE returns to the ROM that generated it Y will contain the ROM number of the last user or &FF if it was not in use. You must store this ID so that when you finish using NMI you can OSBYTE 143 with call number 0B to return NMI to the last user.

Having claimed NMI you can use the NMI zero page &A0 to &A7. The NMI service routine should be put at &D00. You may assume that only your own hardware will be generating NMI as all other ROMs should have turned any other sources off, but it would be a good idea to check that the NMI is coming from your own hardware to make the system more bomb proof.

Do not destroy the filing system zero page memory whilst using call 0C.

0D

Initialise *ROM

When this call is recognised you must check the contents of &F6 to see if it contains your logical ROM number - this is unrelated to your physical ROM number. If it is, then set the bytes &F6,&F7 to point to the data Area of the ROM ready for the data to be read.

OE

Return one byte for *ROM

Using the pointer at &F6 return a single byte to the OS in Y. You should simply return the bytes one by one for all the programs in the ROM. The OS will handle file names etc.

OF

Claim indirection vectors

When claiming the vector space, to point it to your own code, first OSBYTE 143 with X=&0F. This call is intended to allow ROMs other than the filing system to note the fact that the current filing system has been disconnected. It allows you to become aware of the need to re-check the present filing system. Do not use the zero page filing system memory while this call is in use

10

About to close *SPOOL and *EXEC files

This call is a warning to ROMs that these files are about to be closed, usually because the filing system is being closed down. If you cannot allow them to be closed then set A=0, this will force them to be left open through the change.

11

Character set about to explode/implode

Probably mostly of use to a language. This call allows anything in the language main memory to be moved out of the way before the character set explodes all over the program. The character set may also be imploding, in which case you have more memory available. Y contains the new setting for OSHWM. This call is generated when the user types *FX20,X. X sets the number of pages of memory devoted to soft character definitions.

12

Initialise filing system

If Y equals the filing system number for your ROM then initialise your self and set A=0.

FE

Tube post intialisation

This call is always emitted once at BREAK and at no other time. This can be very useful for putting out messages at BREAK time. This is intended to allow Tube machines to finish their initialisation after OSHWM has been set. The main use being to allow the Tube machine to explode the character set.

FF

Tube hardware present

This is the tube pre-initialisation call. This is only given if the tube hardware is fitted.

This occurs before OSHWM etc. are set and so is used to allow the Tube processor to be reset and the tube hardware to be initialised before languages etc. are sent across. It is also used so that the tube start-up message can be output instead of the BBC micro. message.

*ROM

The *ROM mechanism exists to allow programs to be loaded from ROMs as if they were being loaded off tape. The system is to return bytes one by one in the format of a standard tape. The OS will test the file names given to it etc. until it finds the required file.

When using this system it is possible to use a condensed format for the block headers. All headers other than the first and last may be replaced by a single 'hash' character ('E'). This replaces the synchronisation byte.

For protection of *ROM software, use the 'file attributes' as described in the User Guide and DFS manual. These work with the cassette system so you can set your programs in the ROM to be not readable by you or others, but still allowing them to be executed. To be of use you must have at least one file that can be *RUN as this is the only type of file that can be protected. The user can of course save the ROM contents directly and can copy the ROM, store the data internally in a strange format and make the accessing code as obscure as possible to make it harder to read the bytes from the ROM in the correct order manually.

FILING SYSTEMS

A filing system ROM uses the above mechanisms to obtain workspace, the use of NMI etc. There is a second system for accessing filing system commands such as *RUN through the file control system.
At Break you must set the size of the main workspace and private workspace areas when

service calls 1 and 2 are received:

You should then be offered service call-3 at which point you should either start up or pass the call on. You should process !BOOT if required.

The filing system may be started up by three different routes:
At Break, the usual method
By a * command such as *DISC from service call 4 Service call &12, (check Y to see if it is your filing system number)

When starting up you must claim the main workspace memory by calling OSBYTE 143 with X=&0A, Y=&FF if the use of the memory is required.

EXTENDED VECTOR SPACE

You may require one of the vectors at &200 to point into your own ROM. This would allow all print output to be directed to the ROM for instance. DFS directs OSBPUT, OSFILE etc. to gain control of the machine when a filing operation occurs. The obvious thing to do is to point the indirection vector directly at the routine in your ROM. Unfortunately this does not work as it is necessary for the correct ROM to be selected in the sideways ROM space. Thus each required vector must be pointed to a special routine in the OS which accesses a new table of vectors, this table containing the physical ROM number as well as the address.
To gain control of any routine in this way is rather complicated. You must first inform the last user of the extended vector space that you are grabbing the vectors. This allows the last

user to switch itself off properly. This is achieved through a call to OSFCM, the filing system control routine. This is entered with JMP(&21E). You must execute this with A=6 to inform the last user of the vectors that they are being claimed. Failure to do this will probably lose data in any file buffers currently open and will probably result in a crashed machine.

Having taken control of the vectors you must point each required vector to a new address in 'page &FFXX. Point for instance OSARGS (at &214) at &FF1E, take the vector number, the address of the existing vector being &200+2*vector number and point this vector at &FF00+3*vector number.

Having pointed the vectors to the extended vector processor in the &FFXX area, you must point the extended vectors at the actual ROM routines. The base address of these vectors is' found with OSBYTE 168 with X=0 and Y=&FF. Returning X=low byte and Y=high byte of the base vector address. This gives the address at which the new vector table should start. Make up the vectors at base+3*vector number in the format:

Low byte, High byte, Physical ROM number

The ROM number to be put into the table is stored at &F4. When you have finished this process you should call OSBYTE 143 with X=&F to tell all sideways ROMs that the filing system vectors have been grabbed, allowing other filing systems to ensure that they are fully de-selected and marked as such.

It is necessary to re-direct the OSFCM routine as you will have to detect when the vectors are grabbed back by another ROM. To gain control over the filing system entries you will need to redirect it anyway.

MORE ON FILING SYSTEMS

The OSFCM routine (JMP(&21E)) is used to send commands to the current filing system. It uses call numbers like the other routines with parameters in X,Y.

A=

00	*OPT X,Y
01	EOF check, handle in X
02	*/<name>
03	*command
04	*RUN
OS	*CAT
06	Filing system vectors being taken
07	Return range of handles used
08	About to process a *command

The OSFCM Commands are explained in more detail as follows:

00

*OPT X,Y has been entered, X,Y contain the parameters. DFS uses this entry for *OPT1 and *OPT4.

01

Check for an end of file condition for the file whose handle is in X. The OS has received an OSBYTE 127 to check for an end of file and so is going to the filing system to find out. Set X to &FF if there is an EOF, otherwise set X=00.

02

*/<name> has been given to the OS. This is usually treated as *RUN <name> as */ is a standard abbreviation for *RUN. You may however treat it differently if needed.

03

* command has been given that, was not recognised by the OS or grabbed through service call 4. You should at this point detect for commands like *DELETE or *COPY and process as required. If you cannot internally recognise the command then you should attempt to *RUN the appropriate file. This action should only occur if you can detect the existence of the file in a few seconds. The user will not want to wait for a few minutes while you check to find his command *COBY before he realises that he has spelt it incorrectly. XY point to the command line, X-low byte, Y-high byte.

04

*RUN <name> has been received. Load and execute the file. You must also set up the pointer to be returned by OSARGS A=l, Y=0 to point to the rest of the command line. XY point to the name.

05

*CAT has been called so print out either a list of files on your medium or print out information on the next file found if you control something slow. XY point to the rest of the command line for parameters such as a drive number.

06

Another filing system is about to take over so you should de-activate yourself by closing *EXEC and *SPOOL files and writing buffers to the media. You may wish to mark yourself as switched off in your private workspace and perhaps release NMI etc.

07

Return the range of file handles that can be returned by your filing system. X=lowest, Y-highest. DFS will return X=17, Y-21.

08

This is sent each time the OS is about to process any command sent to OSCLI. This is used to implement the *ENABLE command in the DFS.

STANDARD PARAMETER SCANNING

The OS contains routines to read the parameters of a * command that interpret 11 etc. in the standard way, thus saving you the effort of writing your own routines. (&F2), Y is used to point to the string to be decoded. This is incremented as required automatically.

FFC2

Initialise reading of the string. If the carry flag is set on entry then the reading of the string will return carry set when a delimiter character such as space is found, If carry is clear on entry then only return will cause the carry flag to be set on exit from the read routine. This call returns with the zero flag set if the string's first character is a terminator. A and Y are corrupted and X is preserved.

FFC5

Reads the next character from the string, automatically decoding control characters etc. (&F2),Y is automatically updated. The carry flag will be set on exit if the type of terminator set by the initialisation is found. X is preserved, Y updated, and A contains the character returned.