1 INTRODUCTION

EXOS is the ROM operating system for the ENTERPRISE micro-computer. It provides an interface between an application program and the hardware of the machine. The main feature of EXOS is a channel based input/output system. The input/output system allows device independent communication with a range of devices including:

1. The display. This may be configured in a variety of modes.

2. The keyboard. Includes the built in joystick and programable function keys.

3. A screen editor with word processing capabilities.

4. Cassette tape.

5. RS232 type serial interface.

6. Centronics compatible parallel interface.

7. Comprehensive four source stereo sound generator.

8. Integral three wire network interface.

Provision is also made for the addition of other devices together with associated driving code so that the system is simply expandable.

The input/output system only supports devices and includes no implicit file handling. Any file handling needed (such as with cassette tape) is provided by the device handler itself. A disk device handler would provide random access file handling facilities but would interface using exactly the same function calls. ..

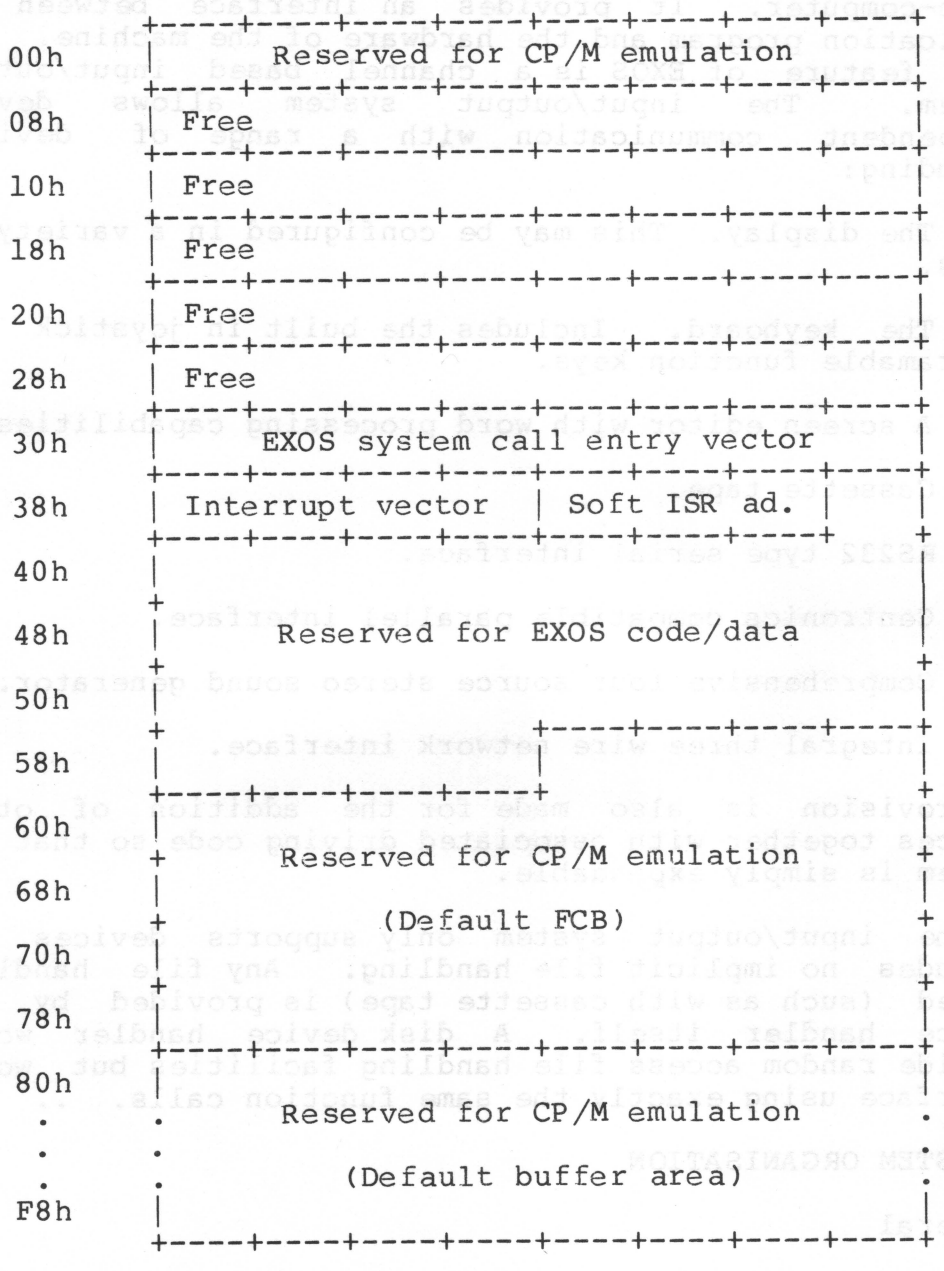
2 EXOS SYSTEM ORGANISATION

2.1 General

EXOS resides in ROM using a segment that is normally not mapped into the Z80 address space. Communication with EXOS is achieved through vectors in the first 256 bytes of RAM (in the Z80 address space). This RAM segment must always be present in the first page of the Z80 address space as it contains the interrupt vector and other system information required to allow EXOS to service interrupts and system calls.

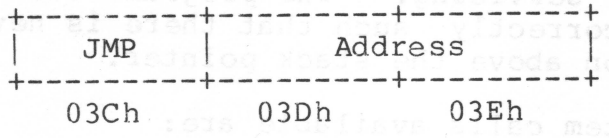
2.2 Use of zero page

The following information is stored in the first 256 bytes of RAM:



2.2 Software Interrupts

The "Soft ISR adr" entry is the address of a user supplied routine to service certain software generated interrupts. The address has the form:



Location 03Ch contains a Z-80 absolute jump instruction and must not be changed by the user. **Address** is the address of the software interrupt service routine. If the address is zero then software interrupts will be ignored. The segment containing the service routine must always be kept in the correct page of Z-80 memory space as it can be entered anytime as a result of an interrupt.

The software interrupt can respond to a variety of events selected by the software, such as STOP key pressed or data arrived on the network.

2.3 System calls

Programs may communicate with EXOS through the "RST 30h" instruction. The code at location 30h is not a simple jump instruction as this code is responsible for selecting the EXOS ROM page before EXOS can be entered. The space from 3Fh to 5Bh will be used for additional code needed for EXOS entry. This entire area from 30h to 5Bh should not be modified by the user at all, except for the SOFT ISR address at 3Dh & 3Eh.

The different EXOS calls are defined by use of a function code. This code is a single byte value and should be stored immediately following the "RST 30h" instruction. Parameters to the EXOS calls are normally passed in the Z80 registers. Buffers and strings are passed by address. Return values from EXOS are also normally passed back in registers. If no values are returned in either register BC or DE then the old contents will be overwritten anyway. However registers HL, IX, IY and the alternate register set (including AF') will remain unchanged by all EXOS calls (with an exception for the ALLOCATE CHANNEL BUFFER call which returns a pointer in IX). Register AF returns a status code.

EXOS maintains a separate system stack in the EXOS workspace and therefore needs very little stack space in the program area. However, at least 8 bytes should always be available beyond the top of the stack. Even if no EXOS calls are made, this space is required for interrupt servicing. The program stack should also be managed correctly such that there is never any wanted information above the stack pointer.

The system calls available are:

Code	Function	
0	System reset	(user only function)
1	Open channel	(user only function)
2	Create channel	(user only function)
3	Close channel	(user only function)
4	Destroy channel	(user only function)
5	Read character	
6	Read block	
7	Write character	
8	Write block	
9	Channel read status	
10	Set and read channel information	
11	Perform special function on channel	
16	Read/Write/Toggle EXOS Variable	
17	Capture channel	
18	Re-direct channel	
19	Set default device name	
20	Return system status	
21	Link device	(user only function)
22	Read exos boundary	
23	Set user boundary	
24	Allocate segment	
25	Free segment	
26	Scan ROMs	(user only function)
27	Allocate channel buffer	(device only function)
28	Explain error code	

All EXOS calls return a status value in register "A". This value is zero if the call was completely successful and non-zero otherwise. The S and Z flags are set to reflect the value of A prior to the return (except for a RESET SYSTEM call which never returns with A non-zero anyway). EXOS function call 28 can be used to provide a simple explanation string for certain error codes (see later).

3 Memory Organisation

3.1 The ENTERPRISE Addressing Scheme

The Z-80 has a 16-bit address bus [0000h - FFFFh] which allows direct addressing of 64k of memory. In order to increase this capacity to 4M the DAVE chip provides an interface to the ENTERPRISE's 22-bit address bus through four internal 8-bit registers (the page registers).

The top two bits of the Z-80 address define four "Z-80 pages" each of 16k and each with a corresponding page register. Thus Z-80 page-0 contains addresses 0000h - 3FFFh, page-1 contains 4000h - 7FFFh, etc. The eight bits from the appropriate page register replace the top two bits of the Z-80 address to give a 22-bit system address.

In this manner the ENTERPRISE memory is divided into 256 16k segments any one of which can be made to appear in any of the four Z-80 pages by putting its segment number into the appropriate page register.

The top four segments (FCh - FFh) provide the built in RAM in the 64k machine. These segments are also the only ones which the NICK chip can use for screen memory and because of this are referred to as the "Video RAM". The internal ROM 32k ROM is made up of segments 00h and 01h, which are reflected by the hardware in segments 02h and 03h. The cartridge occupies segments 04h to 07h.

3.2 EXOS Segment Allocation

When the system is started up, or a cold reset is done, EXOS determines the amount of available RAM and tests each segment. Faulty segments are treated as if they do not exist, leaving the remaining RAM usable. The system will not function, however, unless at least 32k of RAM, including segment FFh, is working.

EXOS maintains an allocation of segments between five states :- free, allocated to the system, allocated to a device, allocated to the user, or shared between the user and the system. There is always one segment (the system segment) allocated to the system, details of which will be given later. This will contain EXOS and device variables and data areas. This will always be the top video RAM segment (segment 0FFh).

One RAM segment (the page zero segment) is kept outside this allocation scheme. This segment contains the EXOS entry points as defined above and must always be in Z-80 page zero when interrupts are enabled or EXOS is called. On a 64k machine this will be the lowest video RAM segment (segment 0FCh). On a machine with any extra RAM, it will be a non-video segment. Apart from the first 256 bytes it may be used by the user. It cannot be freed (see later).

All other RAM segments in the system will initially be free. The contents of Z-80 pages one and two when the applications program is entered will be undefined.

Two EXOS function calls are provided to control RAM allocation - allocate segment and free segment. These calls may be made by the user or by a device driver. The user will only be able to free segments allocated to him and a device driver will only be able to free segments allocated to a device (although there is no check on which device is freeing it).

A new segment may be obtained by a device driver or the user by making an "allocate segment" function call. If there is a free segment then it will be marked as allocated to the user or device as appropriate and its segment number returned. If there is no free segment but there is unused space in the last segment allocated to the system then this segment may be allocated to the user as a shared segment (see the next section). A device will never be allocated a shared segment. If there is no free space at all then a non-zero status code will be returned. Non-video RAM segments will be allocated in preference to video RAM segments if there is a choice.

Whenever EXOS requires more RAM (when a channel is opened) it may grab a free segment if one exists and allocate it to itself. It always frees segments as soon as possible when it requires less memory (when channels are closed). EXOS will always use video RAM segments if any are available.

3.3 Shared Segment

There can be at most one segment in the system which is shared between the user and the system. When the user makes an "allocate segment" EXOS call, if there are no free segments and there is currently no shared segment, then the user will be allocated a segment which EXOS is using part of. This is then the shared segment.

The shared segment contains two boundaries, the USER boundary and the EXOS boundary. EXOS uses the segment from the top down as far as the EXOS boundary. The user is allowed to use from the bottom up as far as the USER boundary, which will never be higher than the EXOS boundary.

The part of the segment (if there is any) between the two boundaries is no-man's land, neither EXOS nor the user should use it. If EXOS requires more RAM to open a channel then it can move the EXOS boundary down as far as the USER boundary and similarly the user can move his boundary up as far as the EXOS boundary to get more space.

To manipulate the boundaries in the shared page two EXOS calls are provided - SET USER BOUNDARY and READ EXOS BOUNDARY.

The page may become un-shared at any time if EXOS no longer needs it, in which case a "read exos boundary" call will indicate that there is no shared segment. Also the user may give up the page entirely to EXOS with a suitable free segment EXOS call. If there is no shared segment when the user does a read system boundary EXOS call then the position that the EXOS boundary would be in a shared page if the user were to be allocated one, is returned, with a segment number of zero to indicate that there is no shared segment. This is useful for finding out how much RAM in the system is free.

Note: The pointers which mark the boundaries both point to the byte above the boundary (since a boundary should be considered as being between two bytes). They will be in the range 0000h...3FFFh, and can be equal.

3.4 Device RAM Areas and Device Descriptors

Every device in the system has a device descriptor which defines the name of the device, the address of its code and various other details. EXOS keeps a linked list of these descriptors in RAM. Details of the format will be given below.

There are three types of device in the system which differ in how they are linked into the chain and how their RAM is allocated.

3.4.1 Built In Devices

Built in devices are those whose code is in the internal ROM with the EXOS kernel. At startup time their device descriptors are copied into RAM and linked into the chain. Each built in device has a fixed area of RAM in the system segment allocated to it. These areas are determined at assembly time and thus may be addressed absolutely since these devices are linked with the EXOS kernel.

3.4.2 External Devices

These are devices whose code is contained in expansion ROMs, either in the cartridge socket or on the expansion stack. They are linked into the system in the same way as, and immediately after, the built in devices. They will each be allocated an area of RAM immediately below the copy of the device descriptor in system RAM. The size of this RAM area is specified in the device descriptor in the ROM.

When the device code is called IY will point to the device descriptor (in Z-80 page-2) and so the RAM can be accessed relative to IY. If "n" bytes are allocated then they can be accessed at:

IY-4, IY-5 ... IY-4-(n-1)

This area of RAM will always be within the one segment and once allocated will never move or be de-allocated, and will be filled with zeroes before calling the devices initialisation routine. Note that in fact built in devices can also be allocated an area of RAM in this way but this is unlikely to be used as it is generally more convenient to address RAM absolutely.

The later section on external ROM scanning will explain exactly which of the 256 possible segments will be examined for external devices.

3.4.3 User Devices

User devices can be linked into the system at any time by an EXOS call giving the address of a complete device descriptor in RAM. (As opposed to the incomplete ones contained in the ROMs). They can be allocated an area of device RAM in the system segment when they are linked in. IX will be pointed to this RAM when the initialisation entry point of the device is called. The device must then remember this address since it will never be told it again. If "m" bytes are allocated then they can be accessed at:

IX-1, IX-2 ... IX-m

Like the device RAM for external devices, this RAM can never move or be de-allocated, and will be filled with zeroes before calling the devices initialisation routine. Note that when the device's initialisation routine is called again (at warm reset time for example) IX will not point to this RAM area so the device will have to remember that it has already been initialised once.

3.5 Channel RAM areas and Channel Descriptors

Every open channel has an area of "CHANNEL RAM" allocated to it. This is allocated when the channel is opened and de-allocated when it is closed. The size of it is determined by the device when the channel is opened and although it cannot change in size after this time, it can be moved around. This can only occur when other channels are opened or closed or a new device is linked in. The device driver is always informed when the channel RAM for any channels open to it is moved.

3.5.1 Allocating Channel RAM

When a channel is opened to a device, EXOS will call the OPEN CHANNEL entry point of the device. At this stage no RAM is allocated. Before it finally returns to EXOS the device driver must make an "allocate channel buffer" EXOS call. This will cause EXOS to allocate the requested amount of channel RAM for this channel. It will return an error code if there is insufficient RAM available, which can then be returned to the user. It actually requires a bit more RAM than the amount specified by the device as it also creates a channel descriptor for this channel.

When the channel RAM is setup the device specifies two 16 bit sizes the sum of which is the total channel RAM size. One size specifies the amount of the RAM which must be in one segment and the other specifies the size of the remainder of the RAM which may be allowed to extend into other segments. The latter of these is ignored (taken to be zero) except for a video device. A pointer to the allocated buffer is returned in IX and the segment containing the start of the buffer will be put in Z-80 page-1.

EXOS moves these channel RAM areas around to make efficient use of memory. Whenever the device driver code is entered with an EXOS channel call, IX will point to the appropriate channel RAM, which will be in Z-80 page 1. When EXOS decides to move this RAM (which can only happen when a channel is opened or closed) it will move it and then call the BUFFER MOVED entry point of the device code to allow the device to adjust any pointers which it may need to. This is particularly important for the video driver as it must update the line parameter table. Interrupts are disabled from just before moving a buffer and returning from the BUFFER MOVED routine of the device.

Note that register IX actually points at the byte after the last one in the RAM area. This is a consequence of the way that EXOS builds its RAM downwards. Thus the bytes which the device driver can use (assuming "m" bytes of channel RAM) are:

(IX-1), (IX-2) ..... (IX-m)

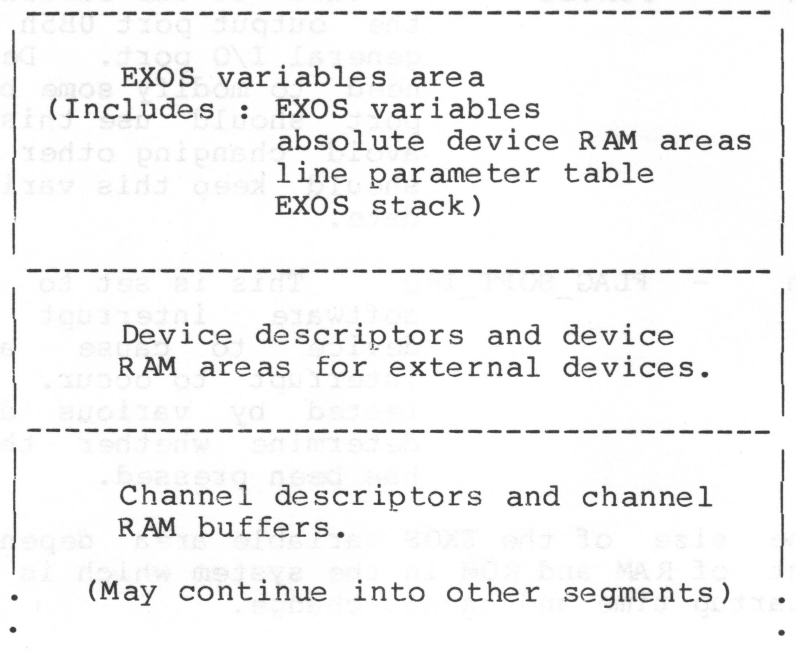
If the allocated buffer extends into a second segment (only allowed for a video device) then this will have a segment number of one less than the first one - and so on for further segments.

3.5.2 Channel Descriptors

As mentioned before EXOS creates a channel descriptor for each channel when it is opened. This is in fact located at IX+0 upwards, but there are no fields in it which are of interest to the device driver so its layout is not defined here.

3.6 EXOS memory usage

EXOS uses RAM from the top of system memory downwards. As far as possible it keeps the video RAM for itself and allocates non-video RAM to the user. The layout of the EXOS system segment (which is always segment 0FFh) is as below:



The very top of the EXOS variable area contains a few defined values which are guaranteed not to move in future versions of EXOS. They are listed here with the address where a device will see them (in Z-80 page-2). If they are to be accessed by an applications program, then the correct segment must of course be paged in.

0BFFFh - USR_P3 \
0BFFEh - USR_P2 \ These are the contents of the four
0BFFDh - USR_P1 / paging registers when EXOS was last
0BFFCh - USR_P0 / called

0BFFA/Bh - STACK_LIMIT Devices which need more than the default 100 bytes of RAM can let their stack grow down as far as the contents of this variable.

0BFF8/9h - RST_ADDR The address of a warm reset routine which must be in the page-zero segment. If this is zero then a cold reset will be done.

0BFF6/7h - ST_POINTER The Z-80 address of the status line memory. The 42 bytes from this address onwards are the status line (see video driver spec ET11/9).

0BFF4/5h - LP_POINTER The Z-80 address of the start of the line parameter table. The first line parameter block will be the status line one.

0BFF3h - PORTB5 This is the current value in the output port 0B5h which is a general I/O port. Devices which need to modify some bits of this port should use this value to avoid changing other bits, and should keep this variable up to date.

0BFF2h - FLAG_SOFT_IRQ This is set to a non-zero software interrupt code by a device to cause a software interrupt to occur. It is also tested by various devices to determine whether the stop key has been pressed.

The size of the EXOS variable area depends on the amount of RAM and ROM in the system which is determined at startup time and cannot change.

The size of the device RAM area depends on what external and user devices are linked into the system. The size of this can only change when a user device which requests some system RAM is linked into the system.

The channel buffer area is very dynamic since it can change whenever a channel is opened or closed. The channel buffers may occupy any number of segments. The system will of course ensure that channel buffers for the video device are kept in the internal video RAM.

4. Applications Program Interface

4.1 Initialisation

4.1.1 Locating Expansion ROMs

Any expansion ROM which is intended to be recognised by EXOS must start with the following:

```
0000:     DB    'EXOS_ROM'
0008:     DW    DEVICE_CHAIN          ;May be zero (see later)
000A:                                 ;Entry point of ROM.
```

When the machine is first switched on, or when a cold reset is performed the entire 4 Mega-byte memory space is examined to look for expansion ROMs. To avoid problems with ROMs reflecting due to incomplete decoding, not every possible 16k segment is examined. Basically each 256k boundary is examined, thus allowing one expansion ROM in each expansion unit on the stack, and then the cartridge slot is examined.

The cartridge slot occupies segment numbers 4, 5, 6 and 7. Each of these segments is tested for a ROM, but when a ROM is found the first 256 bytes of it are compared byte-for-byte with segment 4, and the new ROM ignored if it is identical. This ensures that if a single 16k cartridge which reflects four times is plugged in then it will only be found once, but still allows a mother board with decoding to support one to four 16k cartridges plugged in any of its four slots.

4.1.2 Selecting an applications ROM

The segment numbers of all expansion ROMs which are found are put into a list, which also includes the internal ROM. Under certain circumstances EXOS will call the entry point of all the ROMs at 0C00Ah in turn, passing an "action code" in register C and various other parameters in other registers. This is referred to as a "ROM SCAN" and will be discussed in more detail later with details of the various action codes.

At cold reset time a ROM scan is done with action code 1. Any ROM which contains a main applications program which wants to take control at startup (such as BASIC or LISP) should thus use this as its cue to start. To do this is simply does not return from the entry point call, but must do a reset EXOS (with flags set to 01100000b) call before it can do anything else. Having done this call it must then set up its own stack somewhere in the page-zero-segment re-enable interrupts and is then free to make any EXOS calls.

4.2 Scanning the ROMs

When a ROM scan is done each ROM in the list will be called in turn, with the internal ROM being called last. In certain cases a ROM may not return from this call if it wishes to start itself up as an applications program. This should only occur in response to action codes 1 or 2 below. Normally a ROM will examine the action code and possibly other parameters and then either return immediately, or carry out some function and then return.

If a ROM does return then it must preserve the action code in C and any other parameters passed in registers B or DE, to allow the ROM scan to continue. If the ROM has carried out some service or is returning some parameters and wants to prevent any other ROMs from also trying to do this, it should return with the action code in register C set to zero which all other ROMs will ignore. If this is done then registers B and DE can be corrupted. All other registers (AF, HL, AF', BC', DE', HL', IX and IY) can be corrupted by ROM service routines.

The action codes provided are:

0. Do nothing - Must preserve BC and DE.
1. Cold reset
2. User string
3. Help string
4. Unknown EXOS variable
5. Unknown error code

Action codes above this should not occur and so may be ignored.

4.2.1 Action code 1 - Cold Reset

This was described above. No parameters are passed and any ROM which wants to take control at power on should not return.

4.2.1 Action code 2 - User string

This action code results from a user SCAN ROMs function call. It is passed a pointer to a string in register DE. This string will have a length byte first and will be stored in a buffer in Z-80 page-2 (the system segment). The first word of this string will have been uppercased and register B will contain a count of how many bytes there are in this first word.

The first word is the name of a command, service or program. If the ROM does not recognise this name then it should return from the call, preserving BC and DE. If it does recognize the name then it should respond to it, possibly interpreting the rest of the string as parameters, returning with register C=0 unless it wishes other ROMs to also respond to this command.

The ROM can interpret this action code as a cue to start up if it recognises a suitable string. For example the strings "BASIC", "LISP" and "FORTH" will be interpreted in this way by the appropriate language cartriges. In this case the ROM will not return from the call but behaves as if it received action code 1 (see earlier).

The ROM may recognise the string as a command or service in which case it will carry out some operations which may involve I/O (all normal user EXOS calls including open and close channel and even another SCAN ROM call can be made) and then return. It may be useful for such services to make use of the default channel number facility (using channel number 255 - see later) since it cannot otherwise know what channels are open because it does not know from what applications program it was called.

4.2.1 Action code 3 - Help string

This action code also results from a user SCAN ROMs function call, where the first word of the string was "HELP". The "HELP" (and any trailing spaces) will have been removed from the string and then the rest of the string treated exactly as if it was the original string received from the user. The effect of this is that the ROM sees a string with the first word of it uppercased with the length of this first word in B. If this string received is null (so the original string was just "HELP"), then the ROM should just write its name and version number to the default channel (using channel number 255) and return with BC and DE preserved. If the string is not null then if the first word is any of the action code 2 commands recognised by this ROM then any help information about that command should be printed and register C set to zero.

4.2.1 Action code 4 - Unknown EXOS variable

This action code results when a read/write/toggle EXOS variable call was made with a variable number not-recognised by the internal ROM. Expansion ROMs can therefore implement additional EXOS variables which may be useful for expansion devices. The parameters are (see also the spec of the EXOS call later on):

B = 0, 1 or 2 for READ, WRITE and TOGGLE (ones complement)
E = EXOS variable number (Always >= 128)
D = New value to be written (only if B=1)

If the variable number is not recognised then the ROM should return with BC and DE preserved. If the variable number is one supported by this ROM then the function should be performed and the following parameters returned:

C = 0
D = New value of EXOS variable

4.2.1 Action code 5 - Unknown error code

This action code results from a user "explain error code" function call. The error code is passed in register B and if it is recognised by this ROM then a pointer to a string should be returned as follows:

B = Segment number containing message
C = 0
DE = Address of message string (length byte first).

Error codes in the range 0C0h to 0FFh are reserved for internal EXOS use and so should not be generated by external devices or ROMs although they can be interpreted by this function to re-define the built in error codes.

4.3 Software Interrupts

Software interrupts provide a way for the user to be alerted to various events occuring within EXOS. When a device (probably in its interrupt routine) detects an event which should cause a software interrupt, it sets a byte in the EXOS variable area (FLAG_SOFT_IRQ) to the code indicating what type of interrupt it was. This byte is also available as an EXOS variable.

Nothing else occurs until EXOS is about to return to the user's code, either directly from the interrupt routine, or from an EXOS call if the program was executing in EXOS when the interrupt occurred. Devices should look at FLAG_SOFT_IRQ and cause a premature return if it contains the value ?STOP (20h) to ensure prompt response to the stop key, returning status code .STOP.

When execution is about to return to the user, if he has defined a soft ISR address in page-0 then this routine will be jumped to instead of returning to the user's program. The environment will be exactly as it would be if the return to the user had been made, with the correct paging and the user's stack active. The return address to the user will still be on the stack so the routine may return to the main program. If it does return then all registers must be preserved as it could be an effective interrupt.

It is not necessary for the software interrupt routine to return if it doesn't want to, it can cause some sort of warm re-start of the user's program.

The service routine can find out the software interrupt code by reading an EXOS variable CODE_SOFT_IRQ. This is in fact a copy of the code set up by the device since the code itself is reset to zero before jumping to the routine to avoid multiple responses to the software interrupt. If more than one software interrupt occurs before the soft ISR can be called then only one will be acknowledged.

All sources of software interrupts from built in devices can be enabled or disabled by setting appropriate EXOS variables, or making special function calls. The codes from built in devices are:

10h...1Fh	-	?FKEY....	Keyboard function key pressed.
20h	-	?STOP	Keyboard STOP key pressed
21h	-	?KEY	Keyboard any key pressed
30h	-	?NET	Network data received

4.4 EXOS Variables

There are a number of variables defined which may be set, read or toggled by an EXOS function call. These variables control many different aspects of the system, particularly in controlling options for devices. Each one is 8 bits and is identified by an 8 bit variable number. Some are provided for expansion devices and are not of interest to the user. The currently defined variables are (subject to change):

0 - IRQ_ENABLE_STATE b0 - set to enable sound IRQ.
b2 - set to enable 1Hz IRQ.
b4 - set to enable video IRQ.
b6 - set to enable external IRQ.
b1,3,5 & 7 must be zero.

1 - FLAG_SOFT_IRQ. This is the byte set non-zero by a device to cause a software interrupt. It could also be set by the user to cause a software interupt directly. This variable is also available at a fixed address as specified in section 3.6.

2 - CODE_SOFT_IRQ. This is the copy of the flag set by the device and is the variable that should be inspected by a software interrupt service routine to determine the reason for the interrupt.

3 - DEF_TYPE Type of default device.
0 => non file handling device (eg. TAPE)
1 => file handling device (eg. DISK)

4 - DEF_CHAN Default channel number. This channel number will be used whenever a channel call is made with channel number 255.

5 - LOCK_KEY Current keyboard lock status.

6 - CLICK_KEY 0 => Key click enabled

7 - STOP_IRQ 0 => STOP key causes soft IRQ
<>0 => STOP key returns code

8 - KEY_IRQ 0 => Any key press causes soft IRQ, as well as returning a code.

9 - RATE_KEY Keyboard auto-repeat rate in 1/50 second

10 - DELAY_KEY Delay 'til auto-repeat starts.
0 => no auto-repeat

11 - TAPE_SND 0 => Tape sound enabled


```
12 - WAIT_SND    0 => Sound driver will wait when a
                      queue is full
                <>0 => Sound driver returns .SQFUL error ...
13 - MUTE_SND    0 => internal speaker active.
                <>0 => internal speaker disabled.
14 - BUF_SND     Size of sound envelope storage in
                 'phases', for next sound channel
                 opened.

15 - BAUD_SER    Defines serial baud rate.
16 - FORM_SER    Defines serial word format.

17 - ADDR_NET    Network address of this machine
18 - NET_IRQ     0 => Data received on network will
                 cause a software interrupt.
19 - CHAN_NET    Channel number of network block
                 received

20 - MODE_VID    Video mode      \  These variables select
21 - COLR_VID    Colour mode      \ the characteristics of
22 - X_SIZ_VID   X page size      /  a video page when it
23 - Y_SIZ_VID   Y page size     /   is opened.

24 - ST_FLAG     0 => Status line is displayed

25 - BORD_VID    Border colour of screen
26 - BIAS_VID    Colour bias for palette colours 8...16

27 - VID_EDIT    Channel number of video page for editor
28 - KEY_EDIT    Channel number of keyboard for editor
29 - BUF_EDIT    Size of edit buffer (in 256 byte pages)

30 - FLG_EDIT    Various flags to control editor
                 function - yet to be defined.

31 - SP_TAPE     Non-zero to force slow tape saving.
32 - PROTECT     Non-zero to make cassette write out
                 protected file.
33 - REM1       \ State of cassette remote controls,
34 - REM2       / zero is off, non-zero is on.
```


5. EXOS System Calls.

Below are details of all the EXOS function calls. Function codes 1 to 11 are the EXOS channel calls and cause the appropriate entry point of the appropriate device driver to be called. When the device is entered the channel number will be in register A and be in the range 1...255. The parameters in registers BC and DE will be passed direct to the device driver. Registers IY and IX will be set to point to the appropriate device RAM and channel RAM areas respectively. EXOS saves all registers and so any registers may be corrupted by devices except where parameters are returned in them.

The format of all strings is a length byte, followed by the bytes of the string. The length byte is the number of characters in the string and does not include the length byte itself. A null string is a single byte of zero. All name strings are checked for legality and uppercased before being used by the system or passed to devices. This includes filenames and device names for "open channel" function calls and the first word of name strings for the "locate ROMs" function call.

Where parameters are in fact pointers to memory care must be taken with the memory paging. The correct segment must be in the Z-80 memory page and the buffer pointed to must not go over a 16k page boundary. To access this memory the device driver must determine which segment was in the appropriate Z-80 page when EXOS was called. Four bytes in the EXOS variable area are defined to contain the values of the four paging registers when EXOS was last called so the segment can be extracted from here.

As a convenience for built in devices a common subroutine called GET_SEGMENT is provided which is passed a caller's Z-80 address in DE and will put the appropriate segment in Z-80 page 1. It will return a modified address in DE which points to the required address in Z-80 page 1. Note that this will page out the channel RAM, but the device RAM will still be available in page 2 (for built in & external devices). This routine only alters registers AF and DE.

5.1 Filename syntax

For open and close channel a filename string is required. The syntax of this is:

[[device-name] [unit-number] :] [filename]

where [] denotes an optional part. The device name (if present) is a up to 28 letters, upper or lower case. The unit number (if present) is a decimal number in the range 0 to 255 and may optionally have the character "-" seperating it from the device name, or may immediately follow the device name. The COLON must be present if either the device name or unit number is present to seperate the filename. The filename itself is a string of up to 28 characters from the following:

Upper or lower case letters (not distinguished)
Digits
\/-_.

If the device name is present but no unit number then a default unit number of 1 is used. If no device name is present then the default device name ("TAPE" on the basic machine) is used either with the supplied unit number if there is one or with the default unit number set by the set default device name EXOS calls.

Function 0 - System reset

Parameters: C = Reset type flags
Returns: A = Status (always zero but flags not set)
Interrupts disabled

This call causes a reset of the operation system. The flags passed in register C control exactly what the RESET does, as below.

b0 ... b3 must be zero

b4 - Set => Forcibly de-allocate all channel RAM, and re-initialise all devices. User devices will be retained and device segments will not be de-allocated.

b5 - Set => As bit-4 but also re-link in all built in and extension devices. User devices will therefore be lost. Device segments are not de-allocated.

b6 - Set => De-allocate all user RAM segments.

b7 - Set => Warm reset. This is equivalent to pressing the RESET button on the machine, bits 0 & 1 are ignored and the function will not return. It will cause a jump to the warm reset address if defined or a complete system re-start if not.

Note that the flags are not set to be consistent with the status code (which is always zero anyway) and registers BC', DE' and HL' are corrupted by this EXOS call. Also a side effect of the call is that interrupts are disabled.

Function 1 - Open channel

Parameters: A channel number (must not be 255)
DE pointer to filename string
Returns: A status

The status return in A indicates whether the open was successful. The format of the filename string is specified later. It includes device name, unit number and filename.

The filename and unit number are passed to the device driver for interpretation and many devices will just ignore them. If the device is one which supports filenames then it will return an error code if the file specified does not already exist. Some devices require options to be selected (by special function calls) before the channel can be used. Also some devices require parameters to be specified by setting EXOS variables before a channel can be opened.

If the device is one which allows multiple device drivers of the same name to exist (such as disks) then the unit number is used to determine which driver is called. For example for disks unit numbers 0 and 1 correspond to the first driver, 2 and 3 to the second and so on.

Note that before this call gets through to the device routine, the filename component of the string pointed to by DE will have been copied to a new string which DE now points to, without the device name or number, uppercased and checked for illegal characters. This string will be in the system segment which is located in Z-80 page-2 when the device is called so it is not necessary to do any paging in order to access it. The unit number will be passed as a byte in register C.

For the open channel function to be successfully completed, the device must allocate itself a channel buffer before it returns.

Note also that the channel number will have been incremented by one before being passed through to the device so it will be in the range 1...255 and can never be zero as far as the device is concerned.

Function 2 - Create channel

Parameters: A channel number (must not be 255)
DE pointer to filename
Returns: A status

The create function is similar to the open function except that if the device supports filenames then the file will be created if it doesn't exist, and an error code returned if it does. It is identical to OPEN CHANNEL for all built in devices except the cassette driver.

Function 3 - Close channel

Parameters: A channel number (must not be 255)
Returns: A status

The close function flushes any buffers and de-allocates any RAM used by the channel. Further reference to this channel will not be allowed. The devices entry point is called before the channel RAM is de-allocated.

Function 4 - Delete channel

Parameters: A channel number (must not be 255)
Returns: A status

The destroy function is similar to the close function except that on a filename device the file is deleted. It is identical for all built in devices.

Function 5 - Read character

Parameters:	A	channel number
Returns:	A	status
	B	character

The read character call allows single characters to be read from a channel without the explicit use of a buffer. If no character is ready then it waits until one is ready.

Function 6 - Read block

Parameters:	A	channel number
	BC	byte count
	DE	buffer address
Returns:	A	status
	BC	bytes left to read
	DE	modified buffer address

The read block function reads a variable sized block from a channel. The block may be from 0 to 65536 bytes in length and can cross segment boundaries. Note that the byte count returned in BC is valid even if the status code is negative, although not if it is an error such as non-existent channel. This allows a partially successful block write to be re-tried from the first character which failed.

Function 7 - Write character

Parameters:	A	channel number
	B	character
Returns:	A	status

The write character function allows single characters to be written to a channel.

Function 8 - Write block

Parameters:	A	channel number
	BC	byte count
	DE	buffer address
Returns:	A	status
	BC	bytes left to write
	DE	modified buffer address

The block write function allows a variable sized block to be written to a channel and is similar to block read. The byte count returned in BC is valid even if the status code is negative, provided that control got as far as the device driver.

Function 9 - Channel read status

Parameters:	A	channel number
Returns:	A	status
	C	00h if character is ready to be read FFh if at end of file 01h otherwise.

The read channel status function call is used to allow polling of a device such as the keyboard without making the system wait until a character is ready.

Function 10 - Set and Read Channel Status

Parameters:	A	channel number
	C	Write flags
	DE	pointer to parameter block (16 bytes)
Returns:	A	status
	C	Read flags

This function is used to provide random access facilities and file protection on file devices such as disk or a RAM driver. The format of the parameter block is:

bytes:	0...3	- File pointer value (32 bits)
	4...7	- File size (32 bits)
	8	- Protection byte (yet to be defined)
	9...15	- Zero. (reserved for future expansion)

The assignment of bits in the read and write flags byte is as below. The specified action is taken if the bit is set.

	WRITE FLAGS	READ FLAGS
b0	Set new file pointer value	File pointer is valid
b1	not used (0)	File size is valid
b2	Set new protection byte	Protection byte is valid
b3...b7	not used (0)	always 0

This allows the file pointer and/or the protection byte to be set independently, or just to be read. Not all devices need to support this function. If a device doesn't support it then it should return a .NOFN error code.

Function 11 - Special function

Parameters:	A	channel number
	B	sub-function number
	C	unspecified parameter
	DE	unspecified parameter
Returns:	A	status
	C	unspecified parameter
	DE	unspecified parameter

This function call allows device specific functions to be performed on a channel. Its use is discouraged as it necessarily makes the program device dependent,

Typical functions performed by this call would be:

- Display a given page on the screen
- Program a function key
- Flush the network buffer.

Further details of the special functions available can be found in the individual device driver specifications.

Function 16 - Read, Write or Toggle EXOS Variable

Parameters:	B = 0	To read value
	= 1	To write value
	= 2	To toggle value
	C =	EXOS variable number (0...255)
	D =	New value to be written (only for write)
Returns:	A =	Status
	D =	New value of EXOS variable

This function allows EXOS variables to be set or inspected. These variables control various functions of the system and specific devices. Note that the value is returned in D even for write and toggle. A list of currently defined EXOS variables was given earlier. Expansion ROMs can implement additional EXOS variables but only in the range 128...255.

Function 17 - Capture channel

Parameters:	A -	Main channel number
	C -	Secondary channel number (0FFh to cancel capture)
Returns:	A -	Status

The capture channel function causes subsequent read function calls (read character, read block and read status) to the main channel, to read data instead from the secondary channel. When the function call is made the main channel must exist but no check is made on the secondary channel number.

The capture applies to all subsequent input from the main channel number until either the secondary channel is closed or gives any error (such as end of file) or the main channel is captured from somewhere else. The effect of the capture can be cancelled by giving a secondary channel number of 0FFh which is not a valid channel number.

Function 18 - Re-direct channel

Parameters:	A	-	Main channel number
	C	-	Secondary channel number (0FFh to cancel redirection)
Returns:	A	-	Status

The re-direct function causes subsequent output sent to the main channel with write character or write block function calls to be sent to the secondary channel instead. The redirection lasts until the secondary channel is closed or returns an error or the main channel is redirected somewhere else. A secondary channel number of 0FFh will cancel any redirection of the main channel.

Function 19 - Set default device name

Parameters:	DE	-	device name pointer (without a colon)
	C	-	device type 0 => non file handling
			1 => file handling
Returns:	A	-	status

The set default device name function specifies a device name and (optionally) a unit number which will be used in an OPEN or CREATE function call if none is specified by the user. Initially the default name will be TAPE-1 but will be set to DISK-1 if a disk device is linked in. The specified device name and unit number are checked for legality (ie. no invalid characters) but not for existence in the device chain.

If a string with only a unit number, such as "45" is specified then the new number will become the default but the default name will be un-changed.

The "device type" given in register C is simply copied to the "device type" EXOS variable. This will be zero in the default machine because the default device is "TAPE" which is not a file handling device. If a disk unit is connected then the device type will be set to 1.

Function 20 - Return system status

Parameters: DE -> Parameter block, 8 bytes.
Returns: A = Status code, always 0.
B = Version number (currently 1)
DE - unchanged

This function returns the version number of the system and various parameters which describe the RAM segment usage in the system. The parameters returned are, in order:

0. Shared segment number (0 if no shared segment)
1. Number of free segments.
2. Number of segments allocated to the user, excluding the page-zero segment and the shared segment (if there is one).
3. Number of segments allocated to devices.
4. Number of segments allocated to the system, including the shared segment if there is one.
5. Total number of working RAM segments.
6. Total number of non-working RAM segments.
7. *** Not currently used ***

Function 21 - Link Device

Parameters: DE - Pointer to RAM in Z-80 space containing device descriptor.
BC - Amount of device RAM required.
Returns: A - status

The link device function causes the device descriptor pointed to by DE to be linked into the descriptor chain. The descriptor will be put at the start of the chain and any existing device with the same name will be disabled. DE must point at the TYPE field of the descriptor and the descriptor must not cross a segment boundary. Once linked in the user must ensure that the device code and descriptor are not corrupted until a RESET function call with bit-5 set (to un-link user devices) has been made.

The amount of RAM requested will be allocated in the system segment. Whenever the device is called this RAM area will be pointed to by IY (although it can never move). IY will point at the byte just beyond the end of the RAM so if, for example, two bytes were requested then the two bytes would be (IY-1) and (IY-2).

Function 22 - Read EXOS Boundary

```
Parameters:   none
Returns:      A  - status  (Always zero)
              C  -  Shared  segment number.  0  if
                 there is no shared segment.
              DE -  EXOS boundary in shared segment
                 (0..3FFFh)
```

The read EXOS boundary function returns the offset within the currently shared segment of the lowest byte which the system is using. If there is no shared segment then DE will point to where the EXOS boundary would be if a shared segment were allocated.

Function 23 - Set User Boundary

```
Parameters:    DE  - Offset  of new USER  boundary.
               (0...3FFFh)
Returns:       A  - Status
```

The set user boundary function allows the user to move the USER boundary within the currently shared segment. If there is no shared segment then this function is not allowed. The boundary may not be set higher than the current EXOS boundary.

Function 24 - Allocate Segment

```
Parameters:   none
Returns:      A  - status
              C  - Segment number
              DE - EXOS boundary within segment
```

The allocate segment function allows the user to obtain another 16K segment for his use. If a free segment is available then it will be allocated and status returned zero with segment number in C and DE will be 4000h.

If there are no free segments but the user can be allocated a shared segment, then the segment number will be returned in C and DE will be the initial EXOS boundary. In this case a non-zero positive status code (.SHARE) will be returned. The user boundary is initially set equal to the EXOS boundary.

If there are no free segments and there is already a shared segment then an negative status code will be returned.

If this function call is made by a device driver then the segment will be marked as allocated to a device and a shared segment can not be allocated.

Function 25 - Free segment

```
Parameters:   C - Segment number
Returns:      A - status
```

The free segment function allows the user to free a 16k segment of RAM. The segment must be currently allocated to the user or be shared. The page-zero segment cannot be freed as it was never allocated explicitly with an ALLOCATE SEGMENT call.

If this function call is made by a device driver then it must be to free a segment which was allocated to a device driver with an ALLOCATE SEGMENT call. There is no checking of which device is freeing the segment - Devices are supposed to be well behaved.

Function 26 - SCAN ROM

```
Parameters:   DE = Pointer to application name string
Returns:      A = Status
```

This is the function described earlier which allows the user to cause a ROM scan to occur with action code 2 (or 3 if the first word of the string is "HELP"). This allows external ROM services to be used and also to transfer from one applications ROM to another. Device drivers are not allowed to make this call.

Function 27 - Allocate Channel Buffer

```
Parameters:     DE - Amount of buffer which must be
                     in one segment
                BC - Amount of buffer which needn't
                     be in one seg.
Returns:        A - status
                IX -> Points newly allocated buffer
              PAGE1   contains the new buffer segment
```

The allocate channel buffer function is provided only for devices and may not be called by the applications program. It is used to provide a channel with a RAM buffer when it is opened. The "multi segment size" passed in register BC is ignored for non-video devices since they must have their channel buffer all in one segment. So for non-video devices BC need not be loaded before making the call

Function 28 - Explain Error Code

Parameters: A - Error code which needs explaining
DE - Pointer to buffer for string (80 characters)

Returns: A = 0
DE - Unchanged

This function allows an EXOS error code to be converted into a short text message. All external ROMs are given a chance of doing the translation, and some error codes are explained by the internal ROM. If the string returned is of zero length then it is an error code which no ROM was willing to explain.

To ensure compatibility with later versions new error codes should be below 0C0h since 0C0h to 0FFh are reserved for internal EXOS use, although not all are used at present.

5 Device Driver Interface

5.1 Format of Device Descriptors

The format specified here is of a complete device descriptor in RAM. Whenever a device is entered IY is set to point to the TYPE byte of the descriptor in Z-80 page-2. However the segment containing the descriptor is not explicitly paged in here, since this page must contain the system segment. In in the case of built in and external devices, the descriptor will always be in the system segment and so it will be accesible via IY. This is not the case however for user devices since the descriptor for them could be anywhere. It is assumed that user devices know where their descriptor is (or else don't care).

IY - 3 : NEXT_LOW \
IY - 2 : NEXT_HI > 24-bit address of next descriptor in the chain. The address will be in Z-80 page-1. End of chain marked by Segment number 0.
IY - 1 : NEXT_SEG /

IY + 0 : TYPE - Must be zero. To allow for future expansions.

IY + 1 : IRQFLAG - A byte specifying which interrupts this device should service. If it is zero then the interrupt entry point (see below) need not be valid. The layout of the byte is:

b1 - Set for servicing sound interrupts
b3 - Set for servicing 1Hz interrupts
b5 - Set to service video interrupts
b7 - Set to service external interrupts
b0,2,4 & 6 - unused, must be zero

IY + 2 : FLAGS - General flags byte. Currently only one flag defined:

b0 - Set if device is video device. Channel RAM must be in video RAM.
b1-b7 - Unused - must be zero.

IY + 3 : TAB_LOW \
IY + 4 : TAB_HI > 24-bit address of the start of a table of entry points to the device.
IY + 5 : TAB_SEG / Must be in Z-80 page-1.

IY + 6 : UNIT_COUNT - How many unit numbers this device can serve. See text for details, zero for all built in devices.

IY + 7 : NAME - The device name. The first byte is a length byte, defining the number of characters (0..28) in the name, excluding the length byte itself. The name can include upper case letters, digits, underlines and periods.

5.2 User Devices

When a user device is linked in a pointer to a device descriptor must be given. This points to the TYPE field of the descriptor. The three bytes before this will be set up by EXOS and therefore need not be initialised before linking in. The full 24-bit entry table pointer must be set up to point to the correct segment.

5.3 Built in and External Devices

The internal ROM and any expansion ROMs contain a chain of device descriptors which is scanned at startup time to link in all the devices. There is a pointer to the first descriptor at offset 0008h in the ROM. Each descriptor in the chain must give all fields except the three byte link (NEXT_LOW, NEXT_HI, NEXT_SEG) and the segment of the entry point table (TAB_SEG). The entry point table is assumed to be in the same segment as the descriptor chain and this entry will be filled in when the descriptor is copied to RAM.

The format of the chain of descriptors is:

```
        XX_NEXT_LOW  \  Pointer to next descriptor.  Must
        XX_NEXT_HI   /  be in z-80 page-1.Points at SIZE field.

        XX_RAM_LOW   \  Size of RAM area required (see text)
        XX_RAM_HI    /

        TYPE         \
        IRQFLAG       \
        FLAGS          |
        TAB_LOW        |     These fields are copied to RAM.
        TAB_HI         >     They are as defined earlier
          not used     |
        UNIT_COUNT     |
        NAME           |
          .           /
          .          /

 --->   SIZE           Number of bytes in descriptor
                       (see text).
```

The end of the chain is marked by a pointer of 0000h. Thus if a ROM contains no devices there should be a 0000h at offset 0008h in the ROM. The pointer points at the size field which is a single byte of the number of bytes in the descriptor from the TYPE field to the end of the NAME. This is variable since the name can be of variable length.

The XX_RAM_LOW/HI field is a 16-bit number specifying the size of device RAM area which should be allocated immediately below the descriptor in RAM. It must include two bytes to provide the link at the start of the descriptor and is given in twos complement form. Thus for the built in devices which will require no RAM here, it should have a value of 0FFFEh (-2). The RAM area can be accessed at (IY-4), (IY-5), etc.

5.4 Device Entry Points

Each device descriptor contains a pointer to a table of entry points for the device. Each entry point is a 16-bit address which must be in Z-80 page-3 and be in the same segment as the entry point table itself.

The format of the entry point table is:

0. Interrupt entry point (Need only be valid if IRQFLAG is non-zero).
1. OPEN CHANNEL
2. CREATE CHANNEL
3. CLOSE CHANNEL
4. DESTROY CHANNEL
5. READ CHARACTER
6. READ BLOCK
7. WRITE CHARACTER
8. WRITE BLOCK
9. CHANNEL READ/WRITE STATUS
10. SET AND READ CHANNEL STATUS
11. SPECIAL FUNCTION
12. Initialisation entry point
13. BUFFER MOVED entry point

On entry to any of these routines, the code will be executing in Z-80 page-3. Page-2 will contain the system segment which includes the EXOS stack which will be active. The entry is made by a CALL instruction in page-2 and so the return can be an ordinary RET instruction. Page-0 will contain the page-zero segment with the interrupt and function call entry points in (interrupts will be enabled). For entry points 3 to 11 and 13, page-1 will contain the channel descriptor and start of the channel RAM buffer for the appropriate channel. For other entry points the contents of page-1 are undefined (although for OPEN and CREATE it will contain the channel descriptor after the ALLOCATE CHANNEL BUFFER call has been made).

EXOS does sufficient checking of the stack pointer to ensure that whenever a device is entered , it can use 100 bytes of stack (this number may change) and still have enough stack to make EXOS calls and service interrupts. A device's interrupt routine is assured of 50 bytes (also subject to change) with enough left to make a SET EXOS VARIABLE call.

When page-1 contains the channel descriptor, IX will point at the start of the channel RAM as defined earlier. IY will point to the devices RAM area (depending on the type of the device).

Entry points 1 to 11 correspond to the EXOS channel calls and parameters are mostly passed straight from the caller. There is an exception for OPEN and CREATE in which the parameters are processed slightly (see earlier). All of them must return an EXOS status code in register A but the flags need not be set consistent with this code since this is done by EXOS.

5.5 BUFFER MOVED Entry Point

This entry point is provided mainly for the video driver which has to keep the line parameter table up to date when buffers are moved, but will also be needed by other devices. If it is not needed it can simply point to a RET instruction (no status code is returned).

It is called by EXOS immediately after a channel buffer has been moved. On entry IX will point to the new position of the channel RAM in Z-80 page-1 in the usual way and A will contain the channel number. Also BC will contain the amount that the buffer was moved by. This is only a 16-bit number and so its sign is ambiguous since the buffer could have been moved by almost 64k either way. This should not be a problem since apart from the video device only the offset (ie lower 14 bits) will ever be relevant.

5.6 Interrupts

EXOS handles all types of interrupts. When an interrupt occurs EXOS examines the hardware to determine which of the sources of interrupt caused it. It then scans through all devices calling the interrupt entry point of each one which had the appropriate bit set in the IRQFLG byte in its descriptor (see earlier).

If two different types of interrupt occur at the same time then they will both be serviced, although one of them will have to wait until the other has been done. There is no nesting of interrupts.

On entry to a device interrupt routines, all registers including the index registers and the alternate register set will have been saved so may be corrupted. On entry, register A will contain a single bit set to indicate which type of interrupt is being serviced, since an interrupt service routine may service more than one type of interrupt.

Interrupts will be disabled before entering any interrupt service routine and it is anticipated that the routine will not re-enable them. Also the service routine should not attempt to reset the interrupt hardware since EXOS does this itself after all devices have been serviced.

There is an EXOS variable called IRQ_ENABLE_STATE which controls which sources of interrupts are enabled. If this is updated it will not affect the actual enable state of the interrupts until the next interrupt has been serviced.

+++++ End of Document +++++

