10 Service ROMs

Service ROMs are ROMs which contain code which is entered via the *service entry point*. Service ROM code will always be executed in the ROM itself i.e. always in the I/O processor c.f. language ROMs. The calls made by the operating system to service ROMs are called *paged ROM service calls* but will usually be referred to as just ‘service calls’.

The type of software which might be implemented in service type ROMs are filing systems, user printer drivers, extension VDU commands and languages; In fact just about anything. It should be noted that extreme care should be taken to implement routines in service ROMs correctly.

To understand how software can be incorporated into a paged ROM, be interfaced correctly with the operating system and thus executed at the appropriate time an understanding of paged ROM service calls is essential.

When a hard reset occurs the operating system makes a note of where physical ROMs exist in paged ROM sockets. Subsequently as the machine carries out its various tasks each time something which may be of significance to software in paged ROMs occurs these ROMs are given an opportunity to respond.

10.1 Paged ROM service calls

The mechanism by which this is performed is as follows. The operating system pages in each paged ROM in turn starting with that ROM in the highest priority socket (paging is performed by writing a value to a hardware latch, the hardware responds to the value written to this location and performs the relevant switching of the chip select signals). If the ROM has a service entry point this code is executed. Before entering the code the accumulator is loaded with a *reason code*, the X register will contain the current ROM number (a ROM is thus able to tell which socket it is in) and the Y register will be loaded with any further relevant information. The paged ROM can return control to the operating system following an RTS instruction. If the ROM has responded

and does not wish any further action to be taken, the accumulator should be set to zero to claim the call otherwise all registers should be unchanged. Below is a list of the reason codes which may be presented to a paged ROM when a service call is performed.

Reason code &00: No operation

No operation, this service call should be ignored because a higher priority ROM has already claimed it.

Reason code &01: Absolute filing system space claim

This call is made during a reset. The operating system is interrogating each ROM to determine how much workspace memory would be required if that ROM was called. This workspace is available temporarily while the filing system ROM is active. Pages &E00 and above are available as a fixed area on the BBC micro and the Electron. Each paged ROM is entered with A=&01 , X=ROM number and Y=top of fixed area. For the highest priority ROM on a BBC micro the Y register will contain &E. The Y register value should be increased in accordance to the requirements of the ROM. If the Y register setting is sufficient or greater than required then the service routine should return the Y register unaltered.

Before using this workspace, the new filing system ROM should deselect the old filing system with OSFSC with A=6 (indirected through (&20E), see section 5.7); and the workspace must be claimed with OSBYTE &8F, X=&0A (see Reason Code &0A of this section).

Reason code &02: Relative space claim

This call is made by the operating system during a reset to determine how much private RAM workspace is required by each ROM. The position of this private area will start from the top of the absolute space claimed by the ROMs and on the relative space

claimed by higher priority ROMs. This call is made with the Y register containing the value of the first available page. This value should be stored in the ROM workspace table at &DFO to &DFF (for ROMs 0 to 15 respectively) and the Y register returned increased by the number of pages of private workspace required.

Reason code &03: Auto-boot call

This call is issued during a reset to allow each service ROM to initialise itself. This enables the highest priority filing system to set up its vectors automatically rather than require explicit selection with a *command. To allow lower priority sevices to be selected the service ROM should examine the keyboard and initialise only if either no key is pressed or if its own ROM specific key is being pressed (e.g. D+BREAK for Acorn DFS). If the ROM initialises it should attempt to look for a boot file (typically !BOOT) to RUN, EXEC or LOAD if the Y register contains zero. This call is made during a reset after the start-up messages have been printed.

Reason code &04: Unrecognised *command

When a line of text is offered to the command line interpreter (CLI) the operating system will pass on any unrecognised command firstly to each of the paged ROMs and then if still unrecognised to the currently active filing system. When the unrecognised command is offered to the paged ROMs this service call is made.

Entry parameters:

A=&04
X=ROM number
Y contains an offset which if added to the contents of &F2 and &F3 point to the beginning of the text with the asterisk and leading spaces stripped off and terminated with a carriage return

On exit:

Registers restored
A=0 if recognised

Filing systems should not intercept filing system commands (which will be common to all filing systems) using this service call but may intercept some filing system utilities (e.g. *DISC, *NET).

Reason code &05: Unknown interrupt

An interrupt which is not recognised by the operating system or which has been masked out by software will result in this call being generated. A service ROM which services devices which might cause interrupts should interrogate such devices to determine if they have generated this interrupt. If the interrupt has been recognised and processed the accumulator should be returned with zero to prevent other ROMs being offered the interrupt. The routine should terminate with an RTS not an RTI.

Reason code &06: BRK has been executed

If a BRK instruction is encountered this call will be generated before indirecting through the BRK vector (BRKVEC, &202). BRKs are usually used to indicate that an error condition has occurred, service ROMs are informed before the current language is able to respond to the BRK via the BRKVEC.

Entry parameters:

A=&06
X=ROM number
Y is undefined but should be preserved
&F0 contains the value of the stack pointer
&FD and &FE point to the error number which is not necessarily in the current ROM (OSBYTE &BA yields this ROM number)

On exit:

All registers should be preserved

Reason code &07: Unrecognised OSBYTE call

When an OSBYTE call has been made and is not recognised by the operating system it is offered to the paged ROMs by this service call. The contents of the A, X and Y registers at the time of the OSBYTE call are stored in locations &EF, &F0 and &F1 respectively.

Reason code &08: Unrecognised OSWORD call

This service call is performed in response to the user issuing an OS WORD call not catered for in the operating system. The contents of the A, X and Y registers at the time of the call are stored in locations &EF, &F0 and &F1 respectively. Unrecognised OSWORD calls with accumulator values greater than or equal to &E0 are offered to the user vector (USERV, &200). An OS WORD call with A=7 (equivalent to the SOUND command in BASIC) given an unrecognised channel will also generate this service call.

Reason code &09: *HELP command interception

When the *HELP command is passed through the CLI this service call is generated. The remainder of the command line is pointed to by the address stored in locations &F2 and &F3 plus an offset in Y. Each ROM is required to respond to this call. If the remainder of the command line is blank the ROM should print its name and version number followed by a list of subheadings to which the ROM will respond.

e.g. Acorn DFS (version 0.90) outputs:

```
DFS 0.90
  DFS
  UTILS
```

Indicating that this ROM responds to *HELP DFS and *HELP UTILS

If the rest of the command line is not blank the service routine should compare it against its subheadings and if a match occurs should output the information under that subheading.

e.g. Acorn DES responds to *HELP UTILS with:

```
DFS 0.90
  BUILD <fsp>
  DISC
  DUMP <fsp>
  TYPE <fsp>
```

If there is more than one item on a line then the ROM should deal with them individually. All registers should be preserved across the service routine.

Reason code &0A: Claim absolute workspace

This service call originates from a paged ROM which requires the use of the absolute workspace. When a filing system ROM is active and requires use of this workspace it should perform an OSBYTE call &8E with X=&0A which will generate this service call. The previous owner of the absolute workspace is then able to save any valuable contents of this memory in its own private data area in the relative workspace. The previous owner should also update a flag within its private data area indicating that it no longer owns the absolute workspace.

The active filing system is selected independently of the ownership of the absolute workspace. Thus while a filing system ROM may have ownership of this workspace the tape filing system may be selected (the tape ES does not require any absolute workspace). Problems may arise when the active filing system paged ROM is called upon but does not have ownership of the absolute workspace. The active filing system should then issue this service call to obtain the use of the absolute workspace. This call should only be made by a filing system starting (see also Reason code &01).

Reason code &0B: NMI released

This service call also originates from paged ROMs and should be generated by performing an OSBYTE call &8F. This call should be issued when a ROM no longer requires the NMI. This releases the zero page locations &A0 to &A7 and the space for the NML routine in page &D00. On entry the Y register contains the filing system number of the previous owner (see OSARGS, section 5.2) and this should be compared to the ROM' s own identity before reasserting control of the NMI.

Reason code &0C: NMI claim

This call should be generated by a paged ROM using OSBYTE &8F when it wishes to take posession of the NMI. The service call should be generated passing &FF in the Y register (i.e. OSBYTE A=&8F, X=&0C and Y=&FF). The current owner should relinquish control returning its filing system number in the Y register in response to this call.

Reason code &0D: ROM filing sytem initialise

When the ROM filing system (RES) is activated in response to a *ROM command this call will be issued when a file is being searched for. On entry the Y register contains 15 minus the ROM number of the next ROM to be scanned. If this ROM number is less than the current ROM' s ID this call should be ignored. Otherwise the active ROM number should be stored in &F5 (in the form 15-ROM number) where the RFS active ROM number is stored. The current ROM should indicate that the service call has been claimed by returning zero in the accumulator and should store a pointer to the data stored within the ROM in locations &F6 and &F7 set aside for use by the RFS.

See chapter 11.

Reason code &0E: ROM filing system get byte

This service call may be issued after a ROM containing RFS data has been initialised with service call &0D, A ROM should respond only if it is the active RFS ROM as indicated by the value in location &F5 (stored in the form 15-ROM number). The fetched byte should be returned in the Y register.

See chapter 11.

Reason code &0F: Vectors claimed

This service call should be generated by any paged ROM (using OSBYTE &8F) which has been initialised and then changed any operating system vector. This call warns paged ROMs that a vector change has occurred.

Reason code &10: SPOOL/EXEC file closure warning

This service call should be produced by the operating system prior to closure of any SPOOL or EXEC files when there is a change of the current filing system. This enables any paged ROM using such a file to respond to the possibly premature closure of these files. SPOOL/EXEC file closure may be prevented by returning a zero in the accumulator otherwise all registers should be preserved.

Reason code & 11 : Font implosion/explosion warning

When OSBYTE &14 is used to change the RAM allocation for user defined characters this service call is issued. This call is issued to warn languages that the OSHWM has been changed and thus the user RAM allocation has changed.

Reason code &12: Initialise filing system

This call enables third party software to switch between one or more filing systems without having to issue *commands. A program may want to switch between two filing systems in order to transfer files. A filing system ROM should respond to this call if the value in the Y register corresponds to its filing system number. All filing systems should allow files to be open while inactive and so on receiving this call should restore any such files.

Reason code &13: Character placed in RS423 buffer

This call is made when the Electron OS has placed a character in the RS423 buffer. Expansion software handling RS423 hardware should respond to this call. If not claimed the operating system purges the RS423 buffer.

Reason code &14: Character placed in printer buffer

This call is made when the Electron OS has placed a character in the printer buffer. Expansion software controlling printer hardware should respond to this call.

Reason code &15: 100 Hz poll

The Electron operating system will provide a 100 Hz polling call for the use of paged ROMs. A paged ROM requiring this call should increment the polling semaphore using OSBYTE &16 (22) on initialisation and decrement it using OSBYTE &17 (23) when it no longer requires polling. The operating system will issue this service call when the semaphore is non-zero. The semaphore itself may be read using OSBYTE &B9 (185). This facility is implemented mainly so that hardware devices may be supported as a background task without being interrupt driven. This would be suitable for hardware not requiring particularly urgent servicing.

The Y register contains the semaphore value, and should be decremented by the service routine if it is being polled. If a service routine finds it has decremented the Y register to zero, it should claim the call (set A to 0) to improve machine speed (there are no more ROMs which require polling).

Reason code &16: A BEL request has been made

When the external sound flag (OSBYTE &DB/219) is set this call is issued by the OS in response to an ASCII BEL code being output (VDU 7). This is to enable the external sound system to respond appropriately.

Reason code &17: SOUND buffer purged

This call is made when an external sound system is flagged on the Electron and an attempt has been made to purge any of the SOUND buffers.

Reason code &FE: Post initialisation Tube system call

The operating system makes this call during a reset after the OSHWM has been set. The Tube service ROM responds to this by exploding the user defined character RAM allocation.

Reason code &FF: Tube system main initialisation

This call is issued only if the Tube hardware has been detected. This call is made prior to message generation and filing system initialisation.

The fact that these calls are shared by all the service ROMs can lead to wide spread consequences if a service call is misused by one of the ROMs. The programmer should consider the consequences of his ROM claiming calls (or not claiming calls) when present.

10.2 Service ROM example

The program below is a ROM based version of the enlarged printer buffer program originally described in chapter 6, and will only be of use to machines with the Plus 1 expansion. It is short by paged ROM standards but the assembler program is not a short example.

This program should only be taken as an illustration of the use of some of the service calls described above : it does not conform to paged service ROM standards, as it uses Econet zero page workspace. This may be of little consequence to the vast majority of Electrons, but properly implemented service ROMs should *never* assume that they won' t be used with any particular system configuration.

```
 10 REM Assembler program printer buffer ROM

 20 DIM code% &400
 30 INSV=&22A:nI=&2A/2
 40 RMV=&22C:nR=&2C/2
 50 CNPV=&22E:nC=&2E/2
 60 ptrblk=&90
 70 ip_ptr=ptrblk+2
 80 op_ptr=ptrblk+4
 90 old_bfr=&880
100 begin=old_bfr
110 end=old_bfr+2
120 wrkbt=old_bfr+4
130 size=old_bfr+5
140 vec_cpy=old_bfr+6
150 line=&F2
160 OSASCI=&FFE3
170 OSBYTE=&FFF4

180 FOR I=4 TO 7 STEP 3
190 P%=&8000:O%=code%
200 [
210 OPT I

220 .romstrt  EQUB 0              \ null language entry point
230           EQUB 0
240           EQUB 0
250           JMP service         \ service entry point
260           EQUB &8?            \ ROM type byte, service ROM
270           EQUB (copyr-romstrt)\ offset to copyright
```

```
280          EQUB 0              \ null byte
290 .title   EQUB &A             \ title string
300          EQUS "BUFFER"
310          EQUB &0             \ null byte
320          EQUS "1.00"         \ version string
330          EQUB &D             \ carriage return
340 .copyr   EQUB 0              \ terminator byte
350          EQUS "(C)1984 Mark HoLmes"\ copyright message
360          EQUB 0              \ terminator byte

370 \ End of ROM header, start of code

380 .name    EQUS "REFFUB"       \ command name

390 \ Service handling code, A=reason code, X=ROM id &

400 .service CMP #4              \ is reason unknown command?
410          BEQ command         \ if so goto PAGEcommand'
420          CMP #9              \ is reason *HELP
430          BEQ help            \ if so goto 'help'
440          CMP #2              \ is reason private wrkspace
450          BEQ wkspclm         \ if so goto PAGEwkspclm'
460          CMP #3              \ is reason autoboot call
470          BNE notboot         \ if NOT goto PAGEnotboot'
480          JMP autorun         \ BEQ autorun, out of range
490 .notboot RTS                 \ other reason, pass on

500 \ Unknown command, is it *BUFFER ?
510 \ (command Line address in &F?,&F3 (line) + offset Y)

520 .command  TYA:PHA:TXA:PHA    \
530          LDX #6              \ X=length of name
540 .loop1   LDA (line),Y        \ A=next letter of command
550          CMP name-1,X        \ compare with my name
560          BNE notme           \ not equal, goto PAGEnotme'
570          INY                 \ for next letter of command
580          DEX                 \ for next Letter of name
590          BNE loop1           \ if X<>0 round again
600          BEQ parmch          \ 6 Letters matched, do jump
610 .notme   PLA:TAX:PLA:TAY     \ no match, restore registrs
620          LDA #4              \ restore reason code
630          RTS                 \ pass on call

640 \ *HELP response (parameters as for call above)

650 .help    TYA:PHA:TXA:PHA     \ save registers
660          LDX #0              \ use X as index counter
670 .loop2   LDA title,X         \ A=next Letter from title $
680          BNE over1           \ if A<>0 jump next instrctn
690          LDA #&20            \ replace 0 by space char.
700 .over1   JSR OSASCI          \ write character
```

```
 710          INX                 \ increment index counter
 720          CPX #(copyr-title)  \ end of title ?
 730          BNE loop2           \ if not get another char.
 740          PLA:TAX:PLA:TAY     \ restore registers
 750          LDA #9              \ restore A
 760          RTS                 \ pass on *HELP call

 770 \ Oportunity to claim private workspace
 780 \ (Y=lst page free, call inc's Y by no. pages claimed)

 790 .wkspclm TYA                 \ copy page no. to A
 800          STA &DFO,X          \ table for ROMs' workspace
 810          PHA                 \ save page no. on stack
 820          LDA #&FD
 830          LDX #0
 840          LDY #&FF            \ OSBYTE call to read last
 850          JSR OSBYTE          \ BREAK type
 860          CPX #0              \ X=0 after soft reset
 870          BEQ softrst         \ soft brk, dont reset size
 880          LDA #8              \ 8 pages for printer buffr
 890          STA size            \ location for buffer size
 900 .softrst CLC                 \ clear carry, for add
 910          PLA                 \ original Y on stack
 920          ADC size            \ A=A+?size
 930          TAY                 \ Y=A
 940          LDX &F4             \ X=ROMid
 950          LDA #2              \ restore A (reason code)
 960          RTS                 \ pass on workspace call

 970 \ *BUFFER command issued, reset buffer size

 980 .parmch  LDA (line),Y        \ get char. from cmnd line
 990          CMP #&D             \ car.ret.? end of line ?
1000          BNE ok_init         \ if not, cont. line input
1010          LDA #1              \ no parameters so set
1020          JMP defauLt         \ default buffer size
1030 .ok_init INY                 \ increment index counter
1040          CMP #&20            \ was char. a space?
1050          BEQ parmch          \ if so get next character
1060          SEC                 \ set carry for subrtact
1070          SBC #&30            \ A=AASNASC"0"
1080          CMP #0              \ was character zero
1090          BEQ deinit          \ if so, switch off
1100          BMI rngerr          \ char.<0, out of range
1110          CMP #6              \ compare char. to 6
1120          BPL rngerr          \ A>=6, out of range
1130 .defauLt CLC                 \ clear carry for ASL
1140          ASL A:ASL A:ASL A   \ A=A*8
1150          STA size            \ store for buffer size
1160 .prntmes LDA #&87            \ Use OSBYTE &87 to read
1170          JSR OSBYTE          \ current screen MODE
```

```
1180          TYA                 \ A=Y
1190          TAX                 \ X=A
1200          LDY #&F8            \ Use OSBYTE &FF to write
1210          LDA #&FF            \ MODE selected on reset
1220          JSR OSBYTE          \ (i.e. MODE preserved)
1230          TAX                 \ X=&FF
1240 .loop6   INX                 \ increment index counter
1250          LDA message,X       \ A=next byte of message
1260          JSR OSASCI          \ print character
1270          CMP #&D             \ was it carriage return
1280          BNE loop6           \ if not get next character
1290          PLA:TAX:PLA:TAY     \ restore registers
1300          LDA #0              \ claim call, 0 reason code
1310          RTS                 \ return
1320 .message EQUB &A             \ message string
1330          EQUS "Press BREAK to change buffer size"
1340          EQUB &D
1350 .rngerr  LDX #&FF            \ set index counter
1360 .loop7   INX                 \ increment index counter
1370          LDA errdata,X       \ A=character from string
1380          STA &100,X          \ copy to bottom of stack
1390          CMP #&FF            \ was byte terminator
1400          BNE loop7           \ if not Loop again
1410          JMP &100            \ goto &100(BRK)
1420 .errdata EQUB 0              \ BRK opcode
1430          EQUB 0              \ error number 0
1440          EQUS "Invalid buffer size"\error
1450          EQUB 0              \ message string end
1460          EQUB &FF            \ terminator byte

1470 \ Routine for deselecting buffer ROM routines

1480 .deinit  LDA #3              \ VDU3, just in case
1490          JSR OSASCI
1500          SEI                 \ disable interrupts
1510          LDY #0
1520          STY size            \ size=0
1530 .loop8   LDA vec_cpy,Y       \ Load old vector contents
1540          STA INSV,Y          \ store in vector
1550          INY                 \ increment index counter
1560          CPY #6              \ copied 6 bytes yet
1570          BNE loop8           \ if not Loop again
1580          CLI                 \ enable interrupts
1590          JMP prntmes         \ print message + return

1600 \ Initialise buffer routines automaticalLy

1610 .autorun TYA:PHA:TXA:PHA     \ preserve registers
1620          LDA size            \ A=buffer size in pages
1630          BEQ no_init         \ A=0, don't initialise
1640          LDA #&84            \ HIMEM OSBYTE number
```

```
1650          JSR OSBYTE          \ make call
1660          STY end             \ store page address
1670          LDA #&83            \ OSHWM OSBYTE number
1680          JSR OSBYTE          \ make call
1690          CPY end             \ is OSHWM > HIMEM
1700          BCC room            \ if so continue
1710          JMP no_room         \ no room so cause error
1720 .room    JSR init            \ call initialise routine
1730 .no_init PLA:TAX:PLA:TAY     \ restore registers
1740          LDA #3              \ restore A
1750          RTS                 \ return
1760 .init    LDA #&A8
1770          LDX #0
1780          LDY #&FF            \ OSBYTE to read address of
1790          JSR OSBYTE          \  extended vector table
1800          STX ptrblk          \ set up zero page Locations
1810          STY ptrblk+1        \  for indirect indexed adr.
1820          LDY #3*nI           \ offset into table (INSV)
1830          LDA #ins AND &FF    \ address of new routine
1840          SEI                 \ disable interrupts
1850          STA (ptrblk),Y      \ copy address to vector
1860          INY                 \ Y=Y+1
1870          LDA #ins DIV &100   \ high byte of address
1880          STA (ptrblk),Y      \ copy to extended vector
1890          INY                 \ Y=Y+1
1900          LDA &F4             \ A=ROMid
1910          STA (ptrblk),Y      \ complete extended vector
1920          INY                 \ Y=Y+1
1930          LDA #rem AND &FF    \ REMV new routine address
1940          STA (ptrblk),Y      \ lo byte to extended vector
1950          INY                 \ YY+1
1960          LDA #rem DIV &100   \ Hi byte of new routine
1970          STA (ptrblk),Y      \ place in extended vector
1980          INY                 \ Y=Y+l
1990          LDA &F4             \ A=ROMid
2000          STA (ptrblk),Y      \ complete REMV 3 byte vect.
2010          INY                 \ Y=Y+1
2020          LDA #cnp AND &FF    \ repeat, store address of
2030          STA (ptrblk),Y      \  new CNPV routine in the
2040          INY                 \  extended vector together
2050          LDA #cnp DIV &100   \  with ROM number.
2060          STA (ptrblk),Y
2070          INY
2080          LDA &F4
2090          STA (ptrblk),Y
2100          TAX                 \ X=ROMid
2110          LDY #0              \ Y=0
2120 .Loop3   LDA INSV,Y          \ Aold vector contents
2130          STA vec_cpy,Y       \ copy to workspace
2140          INY                 \ Y=Y+1
2150          CPY #6              \ copied 6 bytes yet ?
```

```
2160          BNE Loop3           \ if not loop again
2170          LDA &DFO,X          \ Aworkspace addr. hi byte
2180          STA begin+1         \ store in zero page
2190          CLC                 \ clear carry for add
2200          ADC size            \ add begin+size
2210          STA end+1:DEC end+1 \ store in zero
2220          LDY #&10            \ lo byte of begin
2230          STY  begin          \  (room for return vect's)
2240          LDY #&FF            \ lo byte of end
2250          STY end             \ store in zero page
2260          JSR rstptrs         \ reset ip+op ptrs
2270          LDA #nI*3           \ for the extended vector
2280          STA INSV            \  system the vectors must
2290          LDA #nR*3           \  now point to &FFOO +
2300          STA RMV             \  vector number*3
2310          LDA #nC*3
2320          STA CNPV
2330          LDA #&FF
2340          STA INSV+1
2350          STA RMV+1
2360          STA CNPV+1
2370          CLI                 \ enable interrupts
2380          RTS                 \ return
2390 .no_room CLI                 \ clear interrupts
2430 .Loop9   LDA nrmerr,X        \ fetch next byte of data
2440          STA &100,X          \ store at bottom of stack
2450          INX                 \ increment index counter
2460          CMP #&FF            \ reached terminator ?
2470          BNE Loop9           \ if not loop again
2480          JMP &100            \ execute BRK (not in ROM)
2490 .nrmerr  EQUB 0              \ BRK instruction opcode
2500          EQUB 0              \ error number 0
2510          EQUS "Not enough room for print buffer, Press
BREAK"
2520          EQUB 0              \ string terminator
2530          EQUB &FF            \ data end

2540 \ Purge buffer by setting i/p + o/p ptrs to buffer start

2550 .rstptrs LDA begin           \ lo byte bufr start address
2560          STA ip_ptr          \ store input pointer
2570          STA op_ptr          \ store output pointer
2580          LDA begin+1         \ hi byte of buffer start
2590          STA ip_ptr+1        \ store input pointer
2600          STA op_ptr+1        \ store output pointer
2610          RTS                 \ return

2620 .wrngbfl PLA:PLP:JMP (vec_cpy)\ old INSV routine

2630 \ New insert char. into buffer routine

2640 .ins     PHP:PHA             \ save 5 and A on stack
```

```
2650          CPX #3              \ is buffer id 3 ?
2660          BNE wrngbfl         \ if not pass to old routine
2670          PLA:PLP:PHA         \ not passing on, tidy stack
2680          LDA ip_ptr          \ A=lo byte of input pointer
2690          PHA                 \ store on stack
2700          LDA ip_ptr+1        \ A=hi byte of input pointer
2710          PHA                 \ store on stack
2720          LDY #0              \ Y=O so ip_ptr incremented
2730          JSR inc_ptr         \  by the inc_ptr routine
2740          JSR compare         \ compare the two pointers
2750          BEQ insfail         \ if ptrs equal, buffer full
2760          PLA:PLA:PLA         \ don't need ip_ptr copy now
2770          STA (ip_ptr),Y      \ A off stack, insrt in bufr
2780          CLC                 \ insertion success, C=0
2790          RTS                 \ finished
2800 .insfail PLA                 \ buffer was full so must
2810          STA ip_ptr+1        \  restore ip_ptr which was
2820          PLA                 \  stored on the stack
2830          STA ip_ptr
2840          PLA
2850          SEC                 \ insertion failes so C=1
2860          RTS                 \ finished

2870 .wrngbf2 PLP:JMP (vec_cpy+2) \ old REMV routine

2880 \ New remove char. from buffer routine

2890 .rem     PHP                 \ save status register
2900          CPX #3              \ is buffer id 3 ?
2910          BNE wrngbf2         \ if not use OS routine
2920          PLP                 \ restore status register
2930          BVS examine         \ V=1, examine not remove
2940 .remsr   JSR compare         \ compare i/p and o/p ptrs
2950          BEQ empty           \ if the same, buffer empty
2960          LDY #2              \ Y=2 so that increment ptr
2970          JSR inc_ptr         \  routine inc's op_ptr
2980          LDY #0              \ YO, for next instruction
2990          LDA (op_ptr),Y      \ fetch character from bufr
3000          TAY                 \ return it in Y
3010          CLC                 \ buffer not empty, C=0
3020          RTS                 \ return
3030 .empty   SEC                 \ buffer empty, C=1
3040          RTS                 \ return
3050 .examine LDA op_ptr          \ examine only, so store a
3060          PHA                 \  copy of the o/p pointer
3070          LDA op_ptr+1        \  on the stack to restore
3080          PHA                 \  ptr after fetch
3090          JSR remsr           \ fetch byte from buffer
3100          PLA                 \ restore ptr from stack
3110          STA op_ptr+1        \ (if buffer was empty
3120          PLA                 \ C=1 from fetch call)
```

```
3130          STA op_ptr
3140          TYA                 \ examine requires ch. in A
3150          RTS                 \ finished

3160 .wrngbf3 PLP:JMP (vec_cpy+4) \ old CNPV routine

3170 \ New count/purge buffer routine

3180 .cnp     PHP                 \ save status reg. on stack
3190          CPX #3              \ is buffer id 3 ?
3200          BNE wrngbf3         \ if not pass to old subr
3210          PLP                 \ restore status register
3220          PHP                 \ save again
3230          BVS purge           \ if V1, purge required
3240          BCClen              \ if CO, amount in buffer
3250          LDA ip_ptr          \ o/w free space request
3260          PHA
3270          LDA ip_ptr+1        \ store ipptr on stack
3280          PHA
3290          LDX #0              \ X=0 for use as counter
3300          STX wrkbt           \ wrkbt0 for hi counter
3310          LDY #0              \ YO, so ip_ptr incr'd
3320 .loopl   JSR inc_ptr         \ increment ipptr
3330          JSR compare         \ does it equal op_ptr
3340          BEQ finshdl         \ if so countfree space
3350          INX                 \ XX+1
3360          BNE no_inc          \ if X=0 don't inc wrkbt
3370          INC wrkbt           \ hi byte of count inc'd
3380 .no_inc  JMP loop1           \ Loop round again
3390 .finshdl PLA                 \ restore ip_ptr off
3400          STA ip_ptr+1
3410          PLA
3420          STA ip_ptr
3430          LDY wrkbt           \ Yhi byte of free space
3440          PLP                 \ restore status register
3450          RTS                 \ finished
3460 .len     LDA op_ptr          \ store op_ptr on stack
3470          PHA
3480          LDA op_ptr+1
3490          PHA
3500          LDX #0              \ X=0 for use as counter
3510          STX wrkbt           \ wrkbt0 hi byte of count
3520          LDY #2              \ Y? so op_ptr incremented
3530 .loop2   JSR compare         \ are ptrs equal ?
3540          BEQ finshd2         \ if so buffer empty
3550          JSR inc_ptr         \ increment op_ptr
3560          INX                 \ increment count
3570          BNE no_inc2         \ if X=0 then increment hi
3580          INC wrkbt           \  byte of count
3590 .no_inc2 JMP loop2           \ loop round again
3600 .finshd2 PLA                 \ restore op_ptr off stack
```

```
3610          STA op_ptr+1
3620          PLA
3630          STA op_ptr
3640          LDY wrkbt           \ Yhi byte of length
3650          PLP                 \ restore status register
3660          RTS                 \ finished
3670 .purge   JSR rstptrs         \ reset i/p & o/p pointers
3680          PLP                 \ restore status register
3690          RTS                 \ return

3700 \ Increment pointer routine. YO op_ptr, Y? ipptr

3710 .inc_ptr CLC                 \ clear carry for add
3720          LDA ip_ptr,Y
3730          ADC 111
3740          STA ip_ptr,Y
3750          LDA ip_ptr+1,Y
3760          ADC #0
3770          STA ip_ptr+1,Y      \ pointerpointer+1
3780          CMP end+1           \ hi byte reached buffr end?
3790          BNE home            \ if not finish
3800          LDA ip_ptr,Y
3810          CMP end             \ Lo byte reached end ?
3820          BNE home            \ if not finish
3830          LDA begin           \ reached end of buffer
3840          STA ip_ptr,Y        \ so reset pointer to
3850          LDA begin+1         \ start address of buffer
3860          STA ip_ptr+1,Y
3870 .home    RTS                 \ return

3880 \ Compare pointers, if equal Z1 don't care otherwise

3890 .compare LDA ip_ptr+1
3900          CMP op_ptr+1        \ compare ptr high bytes
3910          BNE return          \ if not equal return
3920          LDA ip_ptr
3930          CMP op_ptr          \ compare pointr low bytes
3940 .return  RTS                 \ return

3950 ]
3960 NEXT
3970 OSCLI"*S.BRM "+STR$~code%+" "+STR$~O%
```

When this program is run, the ROM image blown into an EPROM and then inserted in an Electron with a Plus 1 expansion an enlarged printer buffer of 2k is automatically initialised.

Typing ‘*BUFFERn’ with n from 1 to 5 selects a buffer size of n*2K at the next BREAK. ‘*BUFFER0’ deselects the enlarged buffer and re-initialises the normal OS routines. ‘*BUFFER’ (no parameters) reselects the default buffer size (2K).

10.3 Extended Vectors

In the example above the operating system buffer maintenance vectors had to be set to point to routines held within the service ROM. The operating system supports a system of extended vectors to enable each of the OS vectors to point to routines held in paged ROMs.

Each OS vector is identified by a number which may be calculated by subtracting &200 (the vector space base address) from the vector address and dividing by two (each vector is two bytes).

The operating system vector should be pointed to a routine at &FF00 plus the vector number multiplied by 3. This routine will use a three byte vector stored in the extended vector space (this address returned by OSBYTE &A8) with an offset of the buffer number multiplied by 3. This vector should contain the address of the routine in the paged ROM followed by its ROM number.

The procedure for a paged ROM to intercept a vector is:

(a) Determine buffer number n
(b) Establish extended vector space, V using OSBYTE &A8
(c) Store new routine' s address in (V+3*n)
(d) Store ROM number following address
(e) Make copy of OS vectors contents if required for return
(f) Store address (&FF00+3*n) in OS vector (&200+2*n)

It is usually a good idea to disable interrupts during this changeover so that an interrupt routine is not able to use the vector in the middle of the change.