Appendix XIII -
Hints, Tips, and Workarounds

Following are a number of 'WORKAROUNDS' for the 464 machine, that is, routines which allow the 464 to act as specified in the '464 FIRMWARE SPECIFICATION'.

Soft End Of File

Reading characters from the disc using CAS IN CHAR when it is redirected to the AMSDOS routine can run into problems caused by the routine returning an error when it reads the end of file character #1A. This can be avoided by patching the jumpblock so that the end of file error is detected and ignored. The following program does this.

```
SAVE_ENTRY:     DEFS 3                  ;Space to save jumpblock entry
;
INSTALL:
        LD      A,(CAS_IN_CHAR + 0)
        LD      HL,(CAS_IN_CHAR + 1)
        LD      (SAVE_ENTRY + 0),A
        LD      (SAVE_ENTRY + 1),HL     ;Save original contents
;
INTERCEPT:
        LD      A,#C3
        LD      HL,NEW_CAS_IN_CHAR
        JR      PATCH                   ;A/HL = jump to new routine
;
RESTORE
        LD      A, (SAVE_ENTRY + 0)
        LD      HL, (SAVE_ENTRY + 1)    ;A/HL jump to original routine
;
PATCH
        LD      (CAS_IN_CHAR + 0), A
        LD      (CAS_IN_CHAR + 1), HL
        RET

;
NEW_CAS_IN_CHAR
        PUSH HL
```

```
CALL    RESTORE          ;Put original jump back because
                         ;AMSDOS requires it to be executed
                         ;in its original position!
CALL    CAS_IN_CHAR      ;Read the character
PUSH    AF
CALL    INTERCEPT        ;Continue intercepting jumpblock
POP     AF
POP     HL
RET     C                ;Quit if OK
RET     Z                ;Quit if ESC
CP      #1A
SCF
CCF
RET     NZ               ;Quit if a real error
OR      A
SCF
RET                      ;Pretend OK if soft EOF
```

Before reading from the file the user should call INSTALL and from then on CAS IN CHAR will return character #1A just like any other character. Note that INSTALL must only be called once, otherwise the original contents of the jumpblock entry will be lost! The patch will be lost if the external commands TAPE, TAPE.IN, DISC or DISC.IN are executed.

MODE switching on V1.0 firmware

Some programs, such as SORCERY+, run with different parts of the screen in different modes. It is possible to do this using the firmware but it is necessary to intercept the SCR MODE CLEAR indirection so that the screen is not cleared each time the mode is changed. On V1.l firmware all that is necessary is to patch a RET instruction into the first byte of the SCR MODE CLEAR indirection. On V1.0 firmware this would result in all the inks being set to the background colour! This can be overcome by using the following routine to install the users own ink refresh routine:

```
LD      HL,SCR_MODE_CLEAR
LD      (HL),#C9
LD      HL,EVENT_BLOCK       ; Points to 9 byte block
LD      DE,INK_ROUTINE       ; points to Interrupt routine
LD      B,#81
CALL    KL_NEW_FRAME_FLY     ; Add Frame Flyback Interrupt
RET
```

```
INK_ROUTINE:  LD      DE,INK_VECTOR        ; Points to INK settings
              JP      MC_SET_INKS

EVENT_BLOCK:  DEFS 9

INK_VECTOR:   DEFB 4,4,10,19,12,11,20,21,13,6,30,31,7,18,25,26,5
```

Detecting the DDI-1 on a 464

If you are writing a program which needs to detect whether a DDI-1 interface is connected to the computer or not, then the way to accomplish this is to issue a DISC command, if this is not found then the DDI-1 is not connected.

The following program shows how to do this under machine code.

```
FIND_DISC:      LD      HL,DISC_COMMAND
                CALL    KL_FIND_COMMAND
                SBC     A,A
                RET
DISC_COMMAND:   DEFM 'DIS','C'+#80
```

Calling FIND DISC will return A = #FF if a DDI-1 is connected or A = 0 if a DDI-1 is not connected).

Express Asynchronous Events

A problem was discovered with Express Asynchronous Events, in that, the COUNT byte should always be reset (to 0) upon termination of the Event, otherwise the event will not be kicked again. This can be done simply by adding the following program to the event routine:

```
...
...
...
LD   HL,EVENT_BLOCK+2
LD   (HL),#00
RET
```

Printing characters above 127 and suppressing the double line feed

The following patches are to enable characters greater than 127 to be printed, and that the line feed character automatically sent after a carriage return can be suppressed and so stop the infamous double line feed problem on printers.

First the main patch code which is used in both programs should be set up as follows:

```
INITIALISE:
        LD      A,(MC_WAIT_PRINT)
        LD      HL,(MC_WAIT_PRINT+1)
        LD      (NEW_PRINT),A
        LD      (NEW_PRINT+1),HL        ;patch the jumpblock to end of new code
        LD      A,#C3
        LD      HL,PRINT_PATCH
        LD      (MC_WAIT_PRINT),A
        LD      (MC_WAIT_PRINT+1),HL    ; redirect the jumpblock to the new code
        RET
```

Then the PRINT PATCH code for printing characters above ASCII 127 is as follows:

```
PRINT_PATCH:
        CP      128
        JR      C,NEW_PRINT             ; character below 128 then print it
        AND     #7F                     ; mask off bit 7
        LD      (CHAR),A                ; store away the character
        PUSH    HL                      ; HL is preserved on exit
        LD      HL,ESCAPE               ; point HL at escape sequence

        LD      B,5
PRINT_LOOP:
        PUSH    BC
        LD      A,(HL)                  ; get character from sequence
        CALL    NEW_PRINT               ; send character to printer
        INC     HL                      ; bump HL to next character
        POP     BC
        DJNZ    PRINT_LOOP              ; do this 5 times
        PO P    HL                      ; restore HL
        RET                             ; return to calling program
```

```
NEW_PRINT:
        DEFS    3                       ; Storage for MC_WAIT_PRINTER
                                        ; jumpblock
ESCAPE  DEFM    27,">"                  ; Sets the printer to alternate character set

CHAR    DEFB    0                       ; storage for original character being sent

        DEFM    27, "< "                ; Sets the printer into normal character set
```

NOTE the escape sequences (ESC= and ESC>) are used with the AMSTRAD DMP2000 printer and may have to be changed for other printers, (e.g. a number of printers use ESC4 and ESC5 instead). Check with your printer user manual.

The PRINT PATCH code to suppress the line feed after a carriage return appears below:

```
PRINT_PATCH:
        CP      10
        JR      NZ,NOT_LF               ; Jump if character not a line feed
        LD      A,(CHAR)                ; restore last character printed
        CP      13                      ; test if last char. was a 13 (CR)
        LD      A,0
        LD      (CHAR),A                ; zeroise the last char. printed store
        SCF                             ; set carry flag, char. printed OK
        RET     Z                       ; Return to calling program if last char.
                                        ; printed was a 13 (CR)
        LD      A.10                    ; If last char. was not a 13 (CR) then print
                                        ; a 10 (Line Feed)
NOT_LF:
        LD      (CHAR),A                        ; store character
NEW_PRINT:
        DEFS    3                       ; Execute printing of character
                                        ; and return to calling program
CHAR:   DEF13   0                       ; storage for I ast character printed
```

So to use either the LF suppressor or the printing characters above 127 routines, the code INITIALISE and the relevant PRINT PATCH should be tied together and then assembled at a convenient location in RAM. Once initialized they will work until a call to MC RESET PRINTER which re-initializes the original indirection jumpblock at MC WAIT PRINTER.

American CPC6128: Frame Flybacks and Interrupts

This technical note discusses the relationship between frame flybacks and interrupts on the NTSC version of the CPC464/664/6128. Currently the only production model affected is the American version of the CPC6128 - all other markets use PAL/SECAM models and this note does not apply to them.

There was an error in the original hardware specification for the CPC464/664/6128 in that the value to be loaded into the Vertical Total Adjust register in the HD6845 (register 5) was incorrectly given as 6 for the NTSC version whereas it should have been 4. Thus the various ROMs produced for the CPC464/664/6128, when used with an NTSC system, load an incorrect value into the 6845 whenever a full reset occurs; for example when the machine is first powered on, or the RESET_ENTRY firmware call is made.

PAL/SECAM systems work correctly and, fortunately, the only effect of the incorrect value on NTSC systems is to cause the interrupt associated with frame flyback to occur at exactly the same time as the frame flyback pulse becomes true. With the correct NTSC value the interrupt will occur 125 microseconds after frame flyback becomes true - corresponding to PAL/SECAM systems and the description given in Section 1.

What this means in practice is that on an American CPC6128 a program which tests the PPI Frame Flyback signal (for example by calling MC_WAIT_FLYBACK) will not see frame flyback become true before the interrupt occurs, but must rely upon frame flyback still being true when the processing associated with the interrupt is complete. If the interrupt processing takes too long, the program will appear to 'Lock Up' because it never sees frame flyback set true.

In order to minimise the possibility of ‘Lock Ups’ occuring Amstrad have ensured that all American CPC6128 machines are fitted with the type of 6845 which does not have a programmable frame flyback pulse length. Therefore the frame flyback will last for a fixed 1000 microseconds rather than the programmed time of 500 microseconds. The 500 microseconds period is in fact quite sufficient for the system routines invoked by the frame flyback interrupt; 1000 microseconds will allow a number of user routines to also occur at that time without any difficulty.

Clearly though, software for American CPC6128s which has much to do at frame flyback time, or which wishes to avoid flickering effects on the top few lines of the screen, must arrange to place the correct NTSC value into register 5 of the 6845 using code equivalent to the following:

```
SET_NTSC    LD      B,#F5
            IN      A,(C)       ; Read PPI port B
            AND     #10         ; Inspect LK4
            RET     NZ          ; Return if not NTSC: No action required
            DI                  ; Need exclusive access to CRTC
            LD      BC.#BC05
            OUT     (C),C       ; Set CRTC address to register 5
            LD      BC.#BD04
            OUT     (C),C       ; Set Vertical Total Adjust to 4
            EI                  ; End of exclusive access
            RET
```

Using Interrupts with Z80 Peripherals

Z80 support chips such as PIO, SIO, DART, DMA, and CTC have an elaborate interrupt priority system involving the connection of the IEO output of one chip to the IEI input of the next in a daisy-chain.

When a chip wishes to interrupt it inspects its IEI input. If this is '1' then no higher priority device is interrupting and the chip may pull on the interrupt request signal. It will also set IEO to '0' so that lower priority devices are aware of its request. If IEI into a chip goes to '0' then the chip will not interrupt until the higher priority devices have been serviced.

When the CPU is actually interrupted an interrupt acknowledge bus cycle occurs and the highest priority interrupting device (the one with IEI = 1 and a reason to interrupt) assumes that it is being serviced and disables its interrupt. This means that interrupt service routines have the option to issue an EI instruction to allow immediate response to higher priority interrupts.

When interrupt servicing is complete a RETI instruction must be issued. This causes a support chip with an interrupt under service to redetermine its interrupt status and the state of IEI and to set IEO accordingly.

Section 11.2 clearly describes the scheme used for external interrupting devices. This relies upon the external device continuing to interrupt during the service routine so that it can be distinguished from the internal ticker interrupt which is automatically disabled as soon as the interrupt acknowledge bus cycle occurs. From the above description it will be clear that Z80 support chips do not meet this condition and thus their interrupts cannot be used. Hardware designers should also note that interrupts should be disarmed by an OUT to the RESET PERIPHERALS channel (# F8FF).

The code given below does two things, it arranges for a RETI instruction to be issued after every interrupt to ensure that all chips which assumed that they were being serviced will reassert their interrupt request. This is important where several Z80 support chips are involved because there is no provision for IE0-IEI connections between add-on devices. Secondly, a RETI is issued immediately before deciding whether an interrupt is internal or external which will mean that Z80 support chips will renew their interrupt request and the firmware will correctly determine that the interrupt is external.

Set up an external device service routine by intercepting the indirection at #0003B. Remember to make this interception code relocatable.

Add the following Z80 support chip code only if a mark 1 ROM versions 0, 1 or 2 is fitted (This means all existing CPC464 664/6128 machines - see KL PROBE ROM).

#0038 (ROM or RAM) originally contains:

```
JP    ADDRESS_X
```

ADDRESS_X + 5 will be in RAM, not under a ROM, and originally contains:

```
LD          A,C
SCF
EI
EX          AF,AF'
DI
```

replace the five bytes at ADDRESS_X + 5 by:

```
CALL        NEW_CODE
RETI
```

replace NEW CODE et seq (which must not be under a ROM) by:

```
        LD          A,C
        SCF
        EX          AF,AF'
        CALL        LABEL_1
        DI
        JP    ADDRESS_X + 10
LABEL_1:
        EI
        RETI
```

Note that there is no suitable indirection or jumpblock into which the new code can be added, so that it is necessarily somewhat more contorted than the usual sort of code which one adds. Note also that this code is only for use with existing ROMs such as are fitted to the CPC464/664/6128. Any future compatible machines will not support it - so it is most important that the program to install the code cheeks the ROM version number before proceeding. Steps will be taken to ensure newer ROM versions will not need alteration in this way.

Note that the old interrupt code and the indirection at #003B will be replaced when KL CHOKE OFF, MC BOOT PROGRAM or MC START PROGRAM are run (viz when a new foreground or background program is executed). Fortunately these are also the routines which issue RESET PERIPHERALS request.