Amstrad

Notepad

COMPUTER

I/O SPECIFICATION

Introduction

The following notes describe the low-level operation of the Amstrad Notepad computers. They are intended for third-party developers who want to program the Notepad in machine code.

As always, I will try to help out if anyone has questions about this but I cannot give an absolute guarantee to be able to provide support on the low-level operation of the machine.

It is our intention that these firmware routines and system variables should be maintained in future issues of the software but we cannot give an absolute guarantee about this.

Cliff Lawson
Notepad project manager
Amstrad Plc
169 Kings Road
Brentwood
Essex
CM14 4EF
ENGLAND

CIS: 75300,1517
email: cliffl@amstrad.com
amstrad@cix.compulink.co.uk
Phone: (+44) 277 208341
Fax: (+44) 277 208065

Originally written by Cliff Lawson in 1992
Reset and styled by Tony Smith (@smittytone) in 2025

I/O Specification for Amstrad NC100

All numbers are in hexadecimal unless suffixed with a "b" for binary or "d" for decimal. Address line numbers A19, A18, etc are in decimal.

SUMARY

Address	Comment	R/W
E0-FF	Not Used	
D0-DF	RTC (TC8521)	R/W
C0-C1	UART (uPD71051)	R/W
B0-B9	Key data in	R
A0	Card Status etc.	R
90	IRQ request status	R/W
80-8F	Not Used	
70	Power on/off control	W
60	IRQ Mask	W
50-53	Speaker frequency	W
40	Parallel port data	W
30	Baud rate etc.	W
20	Card wiat control	W
10-13	Memory management	R/W
00	Display memory start	W

IN DETAIL

Address = 00 Write only

Start address of display memory

bit 7	A15
bit 6	A14
bit 5	A13
bit 4	A12
bits 3-0	Not Used

On reset this is set to 0.

The display memory for the eight-line NC computers consists of a block of 4096 bytes where the first byte defines the state of the pixels in the top left-hand corner of the screen. A single bit set means the pixel is set to black. The first byte controls the first eight dots with bit 7 controlling the bit on the left. The next 59 bytes complete the first raster line of 480 dots. The bytes which define the second raster line start at byte 64 to make the hardware simpler so bytes 60, 61, 62 and 63 are wasted. There are then another 64 bytes (with the last four unused) which defines the second raster line and so on straight down the screen. That is (all numbers decimal):

Byte	0	1	2	..	59	60	..	63
Bit Number	76543210	76543210	76543210	..	76543210	76543210	..	76543210
Pixel Number	01234567	89012345	67890123	..	23456789	wasted	..	wasted
(read bottom	00000000	00111111	11112222		77777777			
to top decimal)	00000000	00000000	00000000		44444444			

....and so on for subsequent lines. (Second line = bytes 64..127 etc.)

Address = 10..13 Read/Write
Memory management control

10	controls 0000-3FFF
11	controls 4000-7FFF
12	controls 8000-BFFF
13	controls C000-FFFF

On reset all are set to 0.

For each address the byte written has the following meaning:

bit 7-6	Together they select ROM, internal RAM, card RAM 00b = ROM 01b = internal RAM 10b = card RAM
bits 5-0	Determine address lines 19 to 14.

Therefore, 00 is the first 16K of ROM, 01 is the second 16K, etc.
40 is the first 16K of internal RAM, 41=second 16K, etc.
80 is the first 16K of card RAM, 81=second 16K, etc.

So, for example, if you want to switch the third 16K of internal RAM so the processor sees it at 4000-7FFF you would output the value 42 to I/O address 11. 42 has bits 7,6 = 01b and bits 5-0 are 000010b which is the third 16K of internal RAM.

Address = 20 Write only
Memory card wait state control

bit 7	1 for wait states, 0 for no wait

On reset this is set to 1. The bit should be set if the card RAM/ROM is 200nS or slower.

Address = 30 Write only
Port baud rate etc.

bit 7	Select card register, 1=common, 0=attribute
bit 6	Parallel interface Strobe signal
bit 5	Not Used
bit 4	uPD4711 line driver, 1=off, 0=on
bit 3	UART clock and reset, 1=off, 0=on
bits 2-0	Set the baud rate as follows: 000 = 150 001 = 300 010 = 600 011 = 1200 100 = 2400 101 = 4800 110 = 9600 111 = 19200

On reset all data is set to 1.

If programming the UART directly ensure that TxD clock is operating x16.

Address = 40 Write only
Parallel interface data

The byte written here is latched into the parallel port output register. To print it you must then take the Strobe signal (I/O address 30 bit 6) low and then high again. If the printer sends ACK this may generate an IRQ if the mask bit is set in I/O address 60 (IRQ mask).

Address = 50..53 Write only
Sound channels period control

50	Channel A period low
51	Channel A period high
52	Channel B period low
53	Channel B period high

On reset all data is set to FF. The top bit in the high byte (51 and 53) switches the resepective sound generator on or off - 1=off, 0=on.

The frequency generated is determined as:

$$\text{Frequency} = \frac{1{,}000{,}000d}{\text{data} * 2d * 1.6276d}$$

So if the data word programmed into 50 and 51 was 7800 (ie 50=0, 51=78) then the frequency generated would be:

$\text{freq} = \frac{1{,}000{,}000d}{7800h * 2d * 1.6276d}$	$= \frac{1{,}000{,}000d}{30720d * 2d * 1.6276d}$	$= \frac{1{,}000{,}000d}{99{,}999.7d}$	= 10Hz

---

Address = 60 Write only
Interrupt request mask

bits 7-4	Not Used
bit 3	Key Scan interrupt (every 10mS)
bit 2	ACK from parallel interface
bit 1	Tx Ready from UART
bit 0	Rx Ready from UART

On reset all bits are 0. For each bit, 1 = allow that interrupt source to produce IRQs, 0 = interrupt source is masked.

Address = 70 Write only
Power off control

bits 7-1	Not Used
bit 0	1 = no effect, 0 = power off

On reset this is set to 1.

Address = 90 Read/Write
IRQ status

bits 7-4	Not Used
bit 3	Key scan
bit 2	ACK from parallel interface
bit 1	Tx Ready interrupt
bit 0	Rx Ready interrupt

When an interrupt occurs this port should be read to determine the source of the interrupt. The bit will be set to 1 to identify the interrupting device. The interrupt can then be cleared by writing 0 to that bit.

Address = A0 Read only
Memory card/battery status

bit 7	Memory card present. 0 = yes, 1 = no
bit 6	Card write protected. 1 = yes, 0 = no
bit 5	Input voltage. 1 if >= to 4 Volts
bit 4	Memory card battery. 0 = battery is low
bit 3	Alkaline batteries. 0 if >= 3.2 Volts
bit 2	Lithium battery. 0 if >= 2.7 Volts
bit 1	Parallel interface BUSY. 0 if busy
bit 0	Parallel interface ACK. 1 if ACK

Address = B0 – B9 Read only
Keyboard data

B0..B9	Each key of the 64 on the keyboard will set a bit in one of these bytes while pressed.

The gate array scans the keyboard every 10mS and then generates an interrupt. The program should then read these 10 I/O locations to determine which key(s) is pushed. When I/O address B9 is read the key scan interrupt is cleared automatically and the next scan cycle will start from B0.

Address = C0 Read/Write
UART control/data (uPD71051)

C0	UART data register
C1	UART status/control register

The UART is the NEC uPD71051. Programmers are advised to study the data sheet for that chip for more information. The Serial interface requires that the uPD4711 line driver chip be truned on by writing a 0 to bit 4 of I/O address 30. While turned on power consumption increases so this should only be done when necessary.

Address = D0 Read/Write
Real Time Clock chip (TM8521)

D0..DC	Data
DD	Control register
DE	Control register (Write only)
DF	Control register (Write only)

See data sheet of chip for more information.

Notes for External Program Writers

To get external programs executed on the Notepad you could either POKE them into memory in BBC BASIC (or even use its built-in Z80 assembler) and then CALL the entry point. However, this does have the drawback of needing to transfer the code back to the machine each time it crashes (as it inevitably will).

The simplest way to develop for the Notepad is to get a PCMCIA drive for your PC and write a binary image direct to the card using that. If this isn't possible then small programs (up to 16K) can be developed by transferring the binary card image into the Notepad using Xmodem from the PC. Then use the "Make program card" feature in the File Transfer menu to write that file onto a newly formatted PCMCIA RAM card.

In either case, to run the resultant code, you just press Function-X (eXecute) and the first 16K page of the RAM card will be switched to the Z80 memory map at C000..FFFF. A Check is made that location C200 holds the ASCII text "NC100PRG" and also that locations C210..C212 contains a long jump to C220. All being well, the Z80 starts executing code at C210 so that, once you have control, you can take over completely if you wish (driving all hardware functions directly). Most people will probably want to cooperate with the in built firmware as it provides most of the routines that one would require anyway.

* The ASCII text "NC100PRG" must appear at C200h
* Program origin is C210h
* Program MUST start with jp C220h
* The program name is at C213h, max 12 characters, zero terminated

```
                org         C200h
                db          "NC100PRG"
                org         c210h
                jp          start
                db          "PROGRAM NAME",0
                org         C220h

        start   ...
```

Available workspace is A000h to A3FFh (shared with other programs), also A800h to AFFFh (this is overwritten if selectfile is called).

The program MUST handle yellow events: either
* Exit when Stop is pressed, or
* Check for yellow event with kmgetyellow and return if carry set

Serious developers may be interested in contacting Ranger Computers Ltd on (+44) 604 589200 as they can produce a device that looks like RAM to a PC but ends in a PCMCIA header plug that connects directly to the Notepad's card slot and the "PC RAM" appears as card RAM to the Notepad.

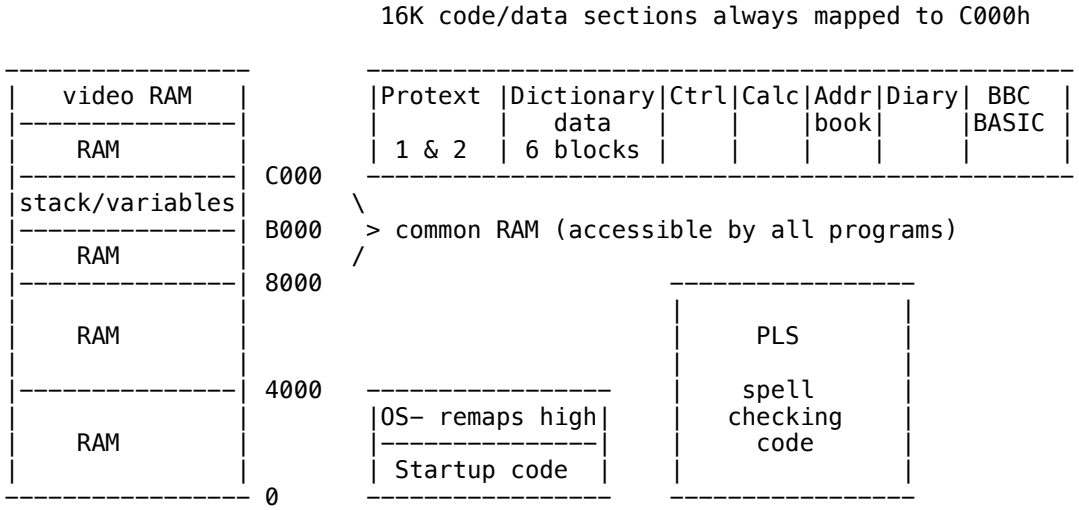
The following sequence is a working(!) piece of code written for the AVMACZ80 assembler on a PC, which, when assmembled produces a binary file that can be programmed onto a PCMCIA card and executed. The program just reads keys and prints them back until "Q" is pressed.

Notice that exit from the program is just by a RET back to the operating system that called it:

```
;=============================================================================================

          include "nc100jmp.inc"              ;The list of firmware routine
                                              ;addresses given later in this
                                              ;file.


          DEFSEG    Fred, CLASS=CODE, START=0

          SEG       Fred                      ;Seg will be linked to RUNSAT C000h.

          jp        start                     ;Put a jump at the start in case this code is
                                              ;ever programmed into a ROM page where the entry
                                              ;will almost certainly be made at the more
                                              ;normal C000.
          ds        509                       ;Waste first 512 bytes of card to start at C200.
;
; Following 16 bytes are Arnor's header for card at C200
;
          db        "NC100PRG",0,0,0,0,0,3,0,1
;
; Card program must start with this long jump at C210
;
          jp        start                     ;Tthis is at C210h.
          db        "CLIFFS PROG",0,0         ;Pad with zeros to C220h.

start:
          call      kmreadchar
          ld        a,c
          cp        "q"
          jr        z,finish
          call      txtoutput
          jr        start

finish:
          ret
          end


;=============================================================================================
; Code is assembled with:
;         AVMACZ80 TEST.ASM
; which produces a .OBJ file which is then linked to produce a .HEX file
; with the command
;         AVLINK @TEST.LNK
; where TEST.LNK contains:
;         TEST.HEX=TEST.OBJ -RUNSAT(Fred, 0C000h)
; Finally the Intel .HEX file is converted to .BIN using a HEX2BIN converter.
; The .BIN file is either written to the PCMCIA card using a PC based
; card drive or it can be Xmodemed across to the Notepad and written to
; the card using "Make program card". Finally, Function-X executes it.
;=============================================================================================
```

In other assemblers you may not have "segments" and must use a direct ORG to locate code at C000 but watch out for the resultant .HEX file being padded out with 48K of "0"s from 0000 to BFFF

Notepad Memory Map



Alphabetic List of Routine Entry Points

To use any one of these routines just load the registers as described in the following and then call the relevant address. Although the running of the routine may involve a different ROM bank being switched in, this mechanism is invisble to the caller. So, for example, to print a capital A one might use:

```
txtoutput           EQU       B833
                    LD        A,"A"
                    CALL      txtoutput
```

```
col1                    equ                 B818h
col1text                equ                 B81Bh

diskservice             equ                 BA5Eh

editbuf                 equ                 B800h

fclose                  equ                 B890h
fdatestamp              equ                 B8C9h
ferase                  equ                 B893h
fgetattr                equ                 B8CFh
finblock                equ                 B896h
finchar                 equ                 B899h
findfirst               equ                 B89Ch
findnext                equ                 B89Fh
fnoisy                  equ                 B917h
fopenin                 equ                 B8A2h
fopenout                equ                 B8A5h
fopenup                 equ                 B8A8h
foutblock               equ                 B8ABh
foutchar                equ                 B8AEh
fquiet                  equ                 B91Ah
frename                 equ                 B8B1h
fseek                   equ                 B8B4h
fsetattr                equ                 B8CCh
fsize                   equ                 B8B7h
fsizehandle             equ                 B8BAh
ftell                   equ                 B8BDh
ftesteof                equ                 B8C0h

heapaddress             equ                 B87Eh
heapalloc               equ                 B881h
heapfree                equ                 B884h
heaplock                equ                 B887h
heapmaxfree             equ                 B88Ah
heaprealloc             equ                 B88Dh

kmcharreturn            equ                 B803h
kmgetyellow             equ                 B8D2h
kmreadkbd               equ                 B806h
kmreadchar              equ                 B9B3h
kmsetexpand             equ                 B809h
kmsettickcount          equ                 B80Ch
kmsetyellow             equ                 B8D5h
kmwaitkbd               equ                 B80Fh

lapcat_receive          equ                 B8D8h
lapcat_send             equ                 B8DBh

mcprintchar             equ                 B851h
mcreadyprinter          equ                 B854h
mcsetprinter            equ                 B857h

padgetticker            equ                 B872h
padgettime              equ                 B875h
padgetversion           equ                 B8DEh
padinitprinter          equ                 BA4Fh
padinitserial           equ                 B85Ah
padinserial             equ                 B85Dh
padoutparallel          equ                 B860h
padoutserial            equ                 B863h
```

```
padreadyparallel        equ                 B866h
padreadyserial          equ                 B869h
padresetserial          equ                 B86Ch
padserialwaiting        equ                 B86Fh
padsetalarm             equ                 B878h
padsettime              equ                 B87Bh
pagemodeon              equ                 BA49h
pagemodeoff             equ                 BA4Ch

readbuf                 equ                 B812h

selectfile              equ                 B8C3h
setdta                  equ                 B8C6h

testescape              equ                 B815h
textout                 equ                 B81Eh
textoutcount            equ                 B821h
txtboldoff              equ                 B83Fh
txtboldon               equ                 B842h
txtclearwindow          equ                 B824h
txtcuroff               equ                 B827h
txtcuron                equ                 B82Ah
txtgetcursor            equ                 B82Dh
txtgetwindow            equ                 B830h
txtinverseoff           equ                 B845h
txtinverseon            equ                 B848h
txtoutput               equ                 B833h
txtsetcursor            equ                 B836h
txtsetwindow            equ                 B839h
txtunderlineoff         equ                 B84Bh
txtunderlineon          equ                 B84Eh
txtwrchar               equ                 B83Ch
```

General Notes

* Most routines return carry set if successful.
* Unless otherwise stated assume AF corrupted, other registers preserved.
* "All registers preserved" includes flags, but NOT alternate registers; the ALTERNATE register contents can NEVER be assumed to be preserved (they are used as scratch registers in time-critical routines).

==========
editbuf = B800
==========
Line editor with options.
Zero-terminated string may be passed in buffer (HL) - this will display the initial contents.

ENTRY HL = pointer to input buffer.
       B = size of buffer (excluding terminating zero).
       A = flags. b6 = 1 -> Dotty background (character 176).
                   b5 = 1 -> Edit unless characters entered.
                   b4 = 1 -> Delete trailing spaces.
                   b3 = 1 -> Input not echoed.
                   b2 = 1 -> Terminate entry.
                   Other bits must be set to zero.
EXIT   c = 0, z = 1: ESC pressed
       c = 1, z = 1: empty string input
       c = 1, z = 0: at least one character entered
       HL preserved
       BC = last key token (or -1 if ESC used to terminate)

==========
kmcharreturn = B803
==========
Returns a token to the keyboard buffer

ENTRY BC = the token
EXIT   All registers preserved

==========
kmreadkbd = B806
==========
Gets a key token if there is one, does not wait (checks put-back character and expands macros).
Returns tick event tokens if enabled.

ENTRY None
EXIT   c = 1: BC = token (B = 0 for simple character).
       c = 0: no key token available.

==========
kmreadchar = B9B3
==========
This routine is the same as kmreadkbd but macros are exapnded and one or two other "behind the scenes" tasks are performed. By using this routine you can be sure that the Ctrl+Shift+S screen dump mechanism works in your code

==========
kmsetexpand = B809
==========
Defines a macro string.

ENTRY BC = macro token (between 256 and 383).
       HL points to new macro string (first byte is the length, followed by the string - need not be zero terminated)
EXIT   c = 1 if macro defined successfully.
       c = 0 if insufficient room in buffer.
       (The buffer size is user configurable)

```
====================
kmsettickcount = B80C
====================
Enables the ticker event.
There are 100 ticks per second.
When a ticker event occurs, t.tickevent is returned by kmreadkbd.

ENTRY        HL = number of ticks before first event.
             DE = number of ticks between events.
EXIT         All registers preserved.


================
kmwaitkbd = B80F
================
Waits for a key token, uses kmreadkbd (checks put-back character and expands macros).
Returns tick event tokens if enabled.

ENTRY        None
EXIT         c = 1: BC = token (B=  0 for simple character).


==============
readbuf = B812
==============
Line editor. See also editbuf.

ENTRY        HL = pointer to input buffer (empty).
             B = size of buffer (excluding terminating zero).
EXIT         c = 0, z = 1: ESC pressed.
             c = 1, z = 1: Empty string input.
             c = 1, z = 0: At least one character entered.
             HL preserved.
             BC = last key token (or -1 if ESC used to terminate).


=================
testescape = B815
=================
Tests whether an ESC key has been pressed (STOP or FUNCTION).
Waits for a key if one is found in the keyboard buffer.

ENTRY        None.
EXIT         c = 1 if no ESC key in buffer.
             c = 1 if ESC key in buffer but STOP not pressed.
             c = 0 if ESC key in buffer and STOP then pressed.
             A is preserved.


===========
col1 = B818
===========
If cursor is at start of a line do nothing, otherwise move cursor to start of next line (within
window).

ENTRY        None.
EXIT         None.


===============
col1text = B81B
===============
Same as textout, but calls col1 first.


==============
textout = B81E
==============
Displays string.

ENTRY        HL : pointer to zero-terminated string.
             ************************************************************
             WARNING - HL must not point into an upper ROM!
             ************************************************************
EXIT         None.
```

```
==================
textoutcount = B821
==================
As textout, returns character count in B.


====================
txtclearwindow = B824
====================
Clears current window and moves cursor to top left.


================
txtcuroff = B827
================
Removes the cursor from the screen.

ENTRY        None.
EXIT         All registers preserved.


===============
txtcuron = B82A
===============
Displays the cursor on the screen.

ENTRY        None.
EXIT         All registers preserved.


==================
txtgetcursor = B82D
==================
Returns the cursor position.

ENTRY        None.
EXIT         H = column (between 0 and 79).
             L = row (between 0 and 7).


==================
txtgetwindow = B830
==================
Returns the window coordinates.

ENTRY        None
EXIT         H = left column (between 0 and 79).
             L = top row (between 0 and 7).
             D = right column (between 0 and 79).
             E = bottom row (between 0 and 7).
             c = 0 if window is whole screen.
             c = 1 if a smaller window has been.


================
txtoutput = B833
================
Displays a character or acts on control code.

ENTRY        A = Character.
                         A = 7  : beep
                         A = 10 : LF
                         A = 13 : CR
             All other values displayed as character (PC char. set).
EXIT         All registers preserved.


==================
txtsetcursor = B836
==================
Moves the cursor.

ENTRY        H = column (between 0 and 79).
             L = row (between 0 and 7).
EXIT         None.
```

```
==================
txtsetwindow = B839
==================
Defines a new window.

ENTRY        H = left column (between 0 and 79).
             L = top row (between 0 and 7).
             D = right column (between 0 and 79).
             E = bottom row (between 0 and 7).
EXIT         None.


================
txtwrchar = B83C
================
Displays a character.

ENTRY        A = character. All values displayed (PC char. set).
EXIT         All registers preserved.


======================
txtboldoff      = B83F
txtboldon       = B842
txtinverseoff   = B845
txtinverseon    = B848
txtunderlineoff = B84B
txtunderlineon  = B84E
======================
These six routines enable or disable various display attributes. They have no entry conditions and
preserve all registers.


==================
mcprintchar = B851
==================
Sends a character to the printer.

ENTRY        A = character.
EXIT         c = 1 if successful.
             c = 0 if not sent.
             A preserved.


======================
mcreadyprinter = B854
======================
Tests whether the printer is ready.

ENTRY        None.
EXIT         c = 0 if busy.
             c = 1 if ready.
             A preserved.


==================
mcsetprinter = B857
==================
Sets the printer type to be used by mcprintchar and mcreadyprinter.

ENTRY        A = printer type: 0 = parallel, 1 = serial
EXIT         None.


====================
padinitserial = B85A
====================
Initialises the serial port using the global configured settings.
Turns on the UART and 4711.
Do not call this until needed - to prolong battery life.

ENTRY        None.
EXIT         None.
```

```
==================
padinserial = B85D
==================
Reads a character from the serial port.

ENTRY        None.
EXIT         c = 1 if successful, A = character.
             c = 0 if no character read.


====================
padoutparallel = B806
====================
Sends a character to the parallel port.

ENTRY        A = character.
EXIT         c = 1 if successful.
             c = 0 if not sent.
             A preserved.


==================
padoutserial = B863
==================
Sends a character to the serial port.

ENTRY        A = character.
EXIT         c = 1 if successful.
             c = 0 if not sent.
             A preserved.


======================
padreadyparallel = B866
======================
Tests whether the parallel port is ready.

ENTRY        None.
EXIT         c = 0 if busy.
             c = 1 if ready.
             A preserved.


====================
padreadyserial = B869
====================
Tests whether the serial port is ready.

ENTRY        None.
EXIT         c = 0 if busy.
             c = 1 if ready.
             A preserved.


====================
padresetserial = B86C
====================
Turns off the UART and 4711.
Call this when finished using the serial port to prolong battery life.

ENTRY        None.
EXIT         None.


======================
padserialwaiting = B86F
======================
Tests whether there is a character waiting to be read from the serial port.

ENTRY        None.
EXIT         c = 1 if character waiting.
             c = 0 if no character waiting.
```

```
==================
padgetticker = B872
==================
Returns the address of a 4-byte 100Hz ticker.

ENTRY        None.
EXIT         HL is the address of the least significant byte.


=================
padgettime = B875
=================
Reads the time and date from the RTC.

ENTRY        HL points to a 7-byte buffer to use.
EXIT         HL preserved.
             Data returned as above (see padsettime).


==================
padsetalarm = B878
==================
Sets the ALARM date and time (within next month).

ENTRY        HL points to 3-byte data area:
             byte 0 = date.
                  1 = hour.
                  2 = minute.
EXIT         None.


=================
padsettime = B87B
=================
Sets the RTC date and time.

ENTRY        HL points to 7-byte data area:
             bytes 0,1 = year (low,high).
                     2 = month.
                     3 = date.
                     4 = hour.
                     5 = minute.
                     6 = second.
EXIT         None.


==================
heapaddress = B87E
==================
Obtains the address of a memory block for a given memory handle.

ENTRY        DE = memory handle.
EXIT         HL = pointer to memory block.


================
heapalloc = B881
================
Allocates a block of memory from the heap.

ENTRY        DE = number of bytes to allocate.
EXIT         HL = memory handle in range [1,63] if successful.
             HL = 0 if failed.

Note:        heapaddress must be used to get a pointer to the memory block. Unless the block is
             locked with heaplock, heapaddress must be called each time the memory block is used.
             IT MAY HAVE MOVED.
```

```
===============
heapfree = B884
===============
Frees a block of memory.

ENTRY        DE = memory handle, returned by heapalloc or heaprealloc.
EXIT         None (preserves HL,BC).

Note:        The memory handle passed must be a valid handle returned by heapalloc or heaprealloc.
             This is not validated.


===============
heaplock = B887
===============
Locks or unlocks a memory block.

ENTRY        DE = memory handle.
             BC = non zero - the block is locked. It will not be moved until unlocked so fixed
                  addresses can be used as pointers into the block.
             BC = 0 - the block is unlocked.


==================
heapmaxfree = B88A
==================
Returns the largest block size that can be allocated.

ENTRY        None.
EXIT         HL = largest free block size in bytes.


==================
heaprealloc = B88D
==================
Changes the size of an allocated memory block.

ENTRY        DE = memory handle.
             BC = new size for memory block.
EXIT         HL = zero if failed to reallocate. The old block will not be freed but could have
                  moved.
             HL = non-zero if successful.

Note:        If the block is being expanded, it must be assumed that the base of the memory block
             will be moved (even if the block cannot actually be expanded) so heapaddress must be
             called afterwards. If the block is being contracted, the base will not move.


=============
fclose = B890
=============
Closes a file.

ENTRY        DE = file handle.
EXIT         c = 1 if successful.
             c = 0 if failed.


=============
ferase = B893
=============
Erases a file.

ENTRY        HL = zero-terminated filename.
EXIT         c = 1 if OK.
             c = 0 if error (file not found).
```

```
===============
finblock = B896
===============
Reads a block from a file.

ENTRY        DE = file handle.
             HL = buffer.
             BC = number of bytes to read (> 0).
EXIT         c = 1 if end of file not reached.
             c = 0 if eof (or error?).
             BC = number of bytes read.
             HL = address after last byte read.

KNOWN BUG (1.00,1.01)
             finblock does not set the file position so repeated calls will always read from the
             start of the file. Workaround: call fseek after calling finblock to set the pointer.


============
finchar B899
============
Reads a byte from a file.

ENTRY        DE = file handle.
EXIT         c = 1 if successful, A = character
             c = 0, A corrupt if end of file reached.
             Other registers preserved.


================
findfirst = B89C
================
Finds first file. setdta must have been called first.

ENTRY        None.
EXIT         HL = 0 if no files.
             HL = pointer to file info structure if file found:
             Offset 0     filename, zero-terminated (up to 12 characters long).
             Offset 13    is attribute byte.
             Offset 14-15 is the file size in bytes.


===============
findnext = B89F
===============
Finds next file. findfirst must have been called first.

ENTRY        None.
EXIT         HL = 0 if no more files.
             HL as findfirst if file found.


==============
fopenin = B8A2
==============
Opens a file for input.

ENTRY        HL points to zero-terminated filename.
EXIT         c = 1 if successful, DE = file handle.
             c = 0 if failed (file not found), DE corrupt if error.
             A corrupt.
             Other registers preserved.

===============
fopenout = B8A5
===============
Opens a file for output.

ENTRY        HL points to zero-terminated filename.
EXIT         c = 1 if successful, DE = file handle
             c = 0 if failed (out of memory/too many files/file exists), DE corrupt if error.
             A corrupt.
             Other registers preserved.
```

```
==============
fopenup = B8A8
==============
Opens a file for input and output. The file must exist already.

ENTRY        HL points to zero-terminated filename.
EXIT         c = 1 if successful, DE = file handle.
             c = 0 if file not found, DE corrupt if error.
             A corrupt.
             Other registers preserved.


================
foutblock = B8AB
================
Writes a block to a file.

ENTRY        DE = file handle.
             HL = buffer.
             BC = number of bytes to write (> 0).
EXIT         c = 1 if OK.
             c = 0 if error.
             BC = number of bytes written.
             HL = address after last byte written.


===============
foutchar = B8AE
===============
Writes a byte to a file.

ENTRY        DE = file handle.
             A = character.
EXIT         c = 1 if successful.
             c = 0, A corrupt if end of file reached.
             A corrupt.
             Other registers preserved.


==============
frename = B8B1
==============
Renames a file

ENTRY        HL = zero-terminated old filename.
             DE = zero-terminated new filename.
EXIT         c = 1 if OK.
             c = 0 if error (file not found).


============
fseek = B8B4
============
Moves the file pointer to a position within a file.

ENTRY        DE = file handle.
             BC = offset from start of file.
EXIT         c = 1 if successful.
             c = 0 if offset past end of file (pointer not changed).

KNOWN BUG (1.00,1.01)
             Leaves error messages enabled (fnoisy). Workaround: call fquiet after fopenout if
             necessary.


============
fsize = B8B7
============
Finds size of file.

ENTRY        HL = zero-terminated filename.
EXIT         c = 1, HL = size in bytes, if found.
             c = 0 if not found.
```

```
==================
fsizehandle = B8BA
==================
Finds size of an open file.

ENTRY        DE = file handle.
EXIT         HL = size in bytes.


============
ftell = B8BD
============
Returns the value of the file pointer.

ENTRY        DE = file handle.
EXIT         HL = current file position.


===============
ftesteof = B8C0
===============
Tests whether end of file has been reached.

ENTRY        DE = file handle.
EXIT         c = 1 if not eof.
             c = 0 if eof.


=================
selectfile = B8C3
=================
Displays the file selector (clears the screen first). Shows all files and allows a selection to be
made using the cursor keys and RETURN.

ENTRY        None.
EXIT         c = 1 if a file selected (RETURN pressed), HL = filename.
             c = 0 if STOP pressed.


=============
setdta = B8C6
=============
Set memory block to be used by findfirst/findnext.

ENTRY        DE = address of buffer (at least 35 bytes long). Buffer must be in
                  common RAM (8000-BFFF).
EXIT         None.


=================
fdatestamp = B8C9
=================
Sets file date/time to current date/time.

ENTRY        HL = zero terminated filename.
EXIT         c = 1 if successful.
             c = 0 if not found.


===============
fsetattr = B8CC
===============
Sets the attribute byte for a file open for output.
If the file is open for input only there is no effect.

ENTRY        DE = file handle
             C = attribute byte:
                 bit 0 = system file.
                 bit 1 = hidden file.
                 bit 2 = BASIC program.
                 bit 3 = binary file.
EXIT         c = 1 if successful.
             c = 0 if not found.
```

```
===============
fgetattr = B8CF
===============
Returns attribute byte of file.

ENTRY        HL = zero-terminated filename.
EXIT         c = 1, A = attribute, if found.
             c = 0, if not found.
             HL preserved.


==================
kmgetyellow = B8D2
==================
Ascertains whether a 'yellow event' is pending (so called because the FUNCTION key is coloured
yellow). A yellow event occurs:
(i) When the user has pressed one of the the FUNCTION+key combinations that cause
    an immediate context switch (FUNCTION+red, FUNCTION+green, FUNCTION+blue, FUNCTION+menu), or
(ii) When the machine is powered up and (because the option to
     preserve context has not been set) needs to return to the main menu.

ENTRY        None.
EXIT         c = 1, BC = token if yellow event pending. An application should exit normally as
             quickly as possible. Any UNSAVED FILES should be SAVED AUTOMATICALLY!
             c = 0, BC = 0 if no yellow event pending.

Note:        Each of the yellow event keys return the ESC token (2FCh). An application should call
             kmgetyellow whenever an ESC is read, this distinguishes between a yellow event and an
             ordinary ESC.


==================
kmsetyellow = B8D5
==================
Sets up a yellow event. Specialised use only.

ENTRY        BC = a yellow event token.
EXIT         None.


====================
lapcat_receive = B8D8
====================
Reads a character from the parallel port using Lapcat protocol.

ENTRY        None.
EXIT         c = 1 if successful, A = character.
             c = 0 if no character read.


==================
lapcat_send = B8DB
==================
Sends a character to the parallel port using Lapcat protocol.

ENTRY        A = character.
EXIT         c = 1 if successful.
             c = 0 if error.


====================
padgetversion = B8DE
====================
Gets the firmware version number.

ENTRY        None.
EXIT         HL = version number (*100). Thus 103 indicates version 1.03.
```

==========
diskservice = BA5E
==========
Calls a Ranger disk routine.

```
ENTRY        C = number of routine to call.
             A, HL, DE passed to the disk routine.
EXIT         c = 1 if successful, HL may contain returned value.
             c = 0 if failed, A = error code (see Ranger documentation).

             C = 0        r_test
                 3        r_begin
                 6        r_change_disk
                 9        r_check_disk
                 C        r_get_cd
                 F        r_set_cd
                 12       r_set_dta
                 15       r_find_first
                 18       r_find_next
                 1B       r_save_file
                 1E       r_retrieve_file
                 21       r_set_attrib
                 24       r_create_directory
                 27       r_remove_directory
                 2A       r_delete_file
                 2D       r_rename_file
                 30       r_finish
                 33       r_disk_space
                 36       r_install
                 39       r_park_heads
                 3C       r_format_track
                 3F       r_format_done
                 42       r_save_wordstar
                 45       r_save_ascii
                 48       r_begin_program
                 4B        r_load_program
```

System Variables

The following are the RAM based variables used by the operating system. It is hoped that they will always use these locations in subsequent versions of the software - but this is not guaranteed.

```
B000    copyofmmu0              ds    1             ; Copy of MMU0 since it's a write-only port.
B001    copyofmmu1              ds    1             ; Copy of MMU1 since it's a write-only port.
B002    copyofmmu2              ds    1             ; Copy of MMU2 since it's a write-only port.
B003    copyofmmu3              ds    1             ; Vopy of MMU3 since it's a write-only port.

B004    gotcontext              ds    1
B005    __savepearlmmu          ds    1             ; Extra vars needed in case we mustn't save
                                                    ; context.
B006    __saveaf                ds    2
B008    __savehl                ds    2
B00A    saveaf                  ds    2             ; To save context, we need to save all
                                                    ; the registers...
B00C    savebc                  ds    2
B00E    savede                  ds    2
B010    savehl                  ds    2
B012    saveix                  ds    2
B014    saveiy                  ds    2
B016    savepc                  ds    2
B018    savesp                  ds    2
B01A    saveafdash              ds    2
B01C    savebcdash              ds    2
B01E    savededash              ds    2
B020    savehldash              ds    2
B022    savemmu0                ds    1             ; ...and the memory state.
B023    savemmu1                ds    1
B024    savemmu2                ds    1
B025    savemmu3                ds    1
B026    savecritpc              ds    2
B028    savecritsp              ds    2
B02A    savingcontext           ds    1
B02B    nmimagic                ds    4
B02F    nmichksums              ds    8             ; Checksum bytes of first 8 roms.
B037    criticalpc              ds    2             ; Save pc,sp for recovery from NMI
                                                    ; during IRQ.
B039    criticalsp              ds    2

B03B                            ds    80            ; A small stack which we only use
                                                    ; in initialisation.
                                                    ; It can't sensibly overlap with anything
                                                    ; in case we get an NMI requring immediate
                                                    ; shut down after saving context.
                                                    ; Subsequent power on will have to
                                                    ; restore the context.
B08B    initstack
B08B    diagnostics?            ds    1             ; Flag used in start-up,
                                                    ; non-zero to do diagnostics.
B08C    saveprinstat            ds    1
B08D    kbdstate1               ds    10            ; 1 bit per key, 1 = down 0 = up;
                                                    ; corresponds to matrix.
B097    kbdstate2               ds    10

PADKEYBUFLEN                    equ   32            ; This MUST be 2^n for positive integer n.
B0A1    padkeybuf               ds    PADKEYBUFLEN*2

B0E1    padnextin               ds    1             ; Offset into padkeybuf.
B0E2    padnextout              ds    1
B0E3    padbufempty             ds    1             ; Non-zero if empty.
B0E4    lastkbdstate            ds    2
B0E6    thiskbdstate            ds    2
B0E8    caps.state              ds    1             ; 0 = off FF = on.
B0E9    savecaps                ds    1
B0EA    justswitchedon?         ds    1

; variables above here are preserved after timeout

PADSERBUFLEN                    equ   32            ; This MUST be 2^n for positive integer n.
B0EB    padserbuf               ds    PADSERBUFLEN
B10B    padsernextin            ds    1
B10C    padsernextout           ds    1
B10D    padserbufempty          ds    1
```

```
B10E    padserin_xoff             ds    1             ; Non-zero when XOFF has stopped inward
                                                      ; transmission.
B10F    padserout_xoff            ds    1             ; Non-zero when XOFF has stopped outward
                                                      ; transmission.
B110    disablexonxoff            ds    1             ; Non-zero to disable software handshake.
B111    ackirq                    ds    1             ; Set non-zero when ACK interrupt occurs.

B112    rptdelay                  ds    1             ; Centisecs.
B113    rptrate                   ds    1             ; Centisecs.
B114    rpttimer                  ds    1             ; Countdown timer for key repeat.
B115    keytorepeat               ds    1             ; Key number.
B116    rptkeystates              ds    1             ; Shift states.

B117    rtcbuf                    ds    13
B124    d.alarmday                ds    6             ; Alarm day,hour,min ready for rtc chip
B12A    alarmhappened             ds    1             ; Non-zero when alarm has gone off,
                                                      ; message pending.
B12B    alarmhappenedgotmsg       ds    1             ; Non-zero when alarm has gone off,
                                                      ; got message & pending.
B12C    soundcounter              ds    1             ; Non-zero if we're playing a tune.
B12D    soundptr                  ds    2             ; Lointer to array of frequency, duration.
B12F    soundrepcount             ds    1
B130    soundrepptr               ds    2
B132    poweroffminutes           ds    1             ; Configured time to power off.
B133    minutesleft               ds    1
B134    minutecounter             ds    2
B136    eventhappened             ds    1
B137    preservecontext           ds    1             ; 0 = return to main screen at power on.
B138    dontpreservecontext       ds    1             ; 1 = dont preserve (diag/batt).

B139    mainprog                  ds    1             ; 6 = inbasic, 128 = inexternal
                                                      ; (foreground program id).
B13A    currentprinter            ds    1             ; 0 = parallel, 1 = serial.
B13B    currentmenu               ds    2             ; Pointer to current menu.
B13D    wasmenusel                ds    1             ; After kmwaitchar this is 1 if menu used,
                                                      ; 0 if not.
                                                      ; Need this in fsel to know whether
                                                      ; redraw needed.

B13E    lastsecond                ds    1             ; Checked to see whether to update the time
B13F    clockon?                  ds    1             ; Used in Protext,
                                                      ; non-zero when clock is enabled.

B140    sdumpname                 ds    4             ; s.a, s.b, s.c etc. for screen dump name

; force d.workspace to an 8 byte boundary
B148    d.workspace               ds    8             ; For massaged copy of symbol data
                                                      ; (eg. inverse/underline).

B150    d.datebuf                 ds    9+MAXMONTHLEN ; 27 January 1992
B162    d.asciitime               ds    12            ; hh:mm:ss xm\0

B16E    currentcfg                ds    cfg.len

B1BA    g.outstream               ds    1             ; bit 0 for screen,
                                                      ;     1 for printer,
                                                      ;     2 for file.
B1BB    g.h.outfile               ds    2             ; File handle for charout if bit 2 set.
B1BD    g.pos                     ds    1             ; Current column number (charout).

B1CE    def.fname                 ds    MAXPNLEN+1    ; Name of current file being edited.
                                                      ; First byte not zero if document open
                                                      ; (yellow/red goes to edit mode,
                                                      ; transfer from addrbook works).
B1DD    def.first                 ds    1

        ; DO NOT CHANGE THE LAYOUT OF THE FIRST 21 BYTES
        0024                      len.findinfo        equ 36
        000D                      o.findinfo.attr     equ 13
        000E                      o.findinfo.size     equ 14
        0010                      o.findinfo.time     equ 16
        0012                      o.findinfo.date     equ 18
        0023                      o.findinfo.mhandle  equ 35
```

```
B1DE    d.findinfobuf           ds len.findinfo

        0002                    o.file.size         equ 2
        0005                    o.file.mhandle      equ 5
        000D                    o.file.attr         equ 13

        000D                    o.direntry.attr     equ 13
        000E                    o.direntry.size     equ 14
        0010                    o.direntry.time     equ 16
        0012                    o.direntry.date     equ 18

;       char name[13];          /* 12 chars plus \0 (the file we found) */
;       char attribute;
;       uint size;              /* Filesize can't be bigger than 64K */
;       uint time,date;         /* If we allow time & date stamping */
;       char flags;             /* Memory block flags */
;       char handle;            /* Memory block handle */

;*****************************************************************************
; PEARL.TXT DATA

; The following 8 bytes are saved for each stream

B202    d.thisstream            ds    8-8
B202    d.colrow                ds    2-2           ; Keep next 2 together.
B202    d.row                   ds    1             ; 0-based within window.
B203    d.col                   ds    1             ;

B204    d.winlefttop            ds    2-2           ; Keep next 2 together.
B204    d.wintop                ds    1
B205    d.winleft               ds    1

B206    d.winsize               ds    2-2           ; Keep next 2 together.
B206    d.winheight             ds    1             ; Height - 1.
B207    d.winwidth              ds    1             ; Width - 1.

B208    d.winset?               ds    1             ; Non-zero if window.
B209    d.state                 ds    1             ; bit 7 if inverse on.

; The following are recalculated from the above (in txtstrselect)

B20A    d.colrowcount           ds    2-2           ; Keep next 2 together.
B20A    d.rowcount              ds    1
B20B    d.colcount              ds    1             ; How many more cols to print on this line.
B20C    d.stream                ds    1             ; Current stream number.
B20D    d.fastpos               ds    2             ; Needed for quick screen update.

B20F    d.streamwsp             ds    8*NSTREAMS    ; 8 streams of 8 bytes each.

B24F    d.dateptr               ds    2             ; Non null for expanding time/date.
B251    d.kmcharret             ds    2             ; Returned character.
B253    d.kstate                ds    2             ; Key locks state.
B255    d.caslocks              ds    1             ; Shift states set by sticky key press.
B256    d.sticky                ds    1             ; Non-zero in sticky key mode.
B257    d.yellow                ds    1             ; Low byte of yellow/other key token
                                                    ; stored by p.xlattoken which then
                                                    ; returns ESC.
B258    d.calcmode              ds    1             ; Non-zero if keyboard in calculator mode.

B259    d.kmexplen              ds    1             ; Expansion string length.
B25A    d.kmexpptr              ds    2             ; Expansion string pointer.
B25C    d.expbuffer             ds    2             ; Address of expansion key buffer.
B25E    d.expbufptr             ds    2             ; Pointer to free byte.
B260    d.expbufend             ds    2             ; Last byte in buffer.

B2A1    macro_buf               ds    256

; file selector variables

B3A7    fs_clicat               ds    1             ; Non-zero if CAT command, not fsel.
B3A8    fs_showsizes            ds    1             ; Non-zero if showing file sizes (pad
                                                    ; default = off).
B3A9    fs_showsys              ds    1             ; Non-zero if showing system files.
B3AA    fs_curfile              ds    1             ; Current file number offset from top left.
B3AB    fs_topleftfile          ds    1             ; File number displayed top left.
B3AC    fs_numcols              ds    1
B3AD    fs_colwidth             ds    1
```

```
B3AE    fs_numshown             ds    1
B3AF    fs_maxfiles             ds    1             ; Max files that can be shown.

FS_NUMROWS                      .equ  7             ; Display rows.
FS_NUMCOLS                      .equ  5
FS_COLWIDTH                     .equ  16

B3B0    fs_handle               ds    2

FS_NUMSHOWN                     .equ  FS_NUMCOLS*FS_NUMROWS
                                                    ; Number of files shown.
B3B2    fs_numfilerows          ds    1             ; Rows of files in CAT command.
B3B3    fs_startlist            ds    2             ; Start of file list;
                                                    ; zero if doing unsorted list.
B3B5    fs_startdir             ds    2             ; Start of directory entries.
B3B7    fs_endlist              ds    2
B3B9    fs_numfiles             ds    1             ; Number of files in directory.
B3BA    fs_lastshown            ds    1             ; Last filenumber currently shown
                                                    ; s_topleft + FS_NUMSHOWN
B3BB    tickcount               ds    4             ; 32-bit counter needed for Basic.
B3BF    tickscleftuntilevent ds 2
B3C1    tickreloadvalue         ds    2
B3C3    tickeventpending        ds    1
B3C4    countdowntimer          ds    2

B3C6    savestream              ds    1

B3C7    password                ds    5             ; Encrypted.
B3CC    pwbuf                   ds    5             ; Clear.
B3D1    realpwbuf               ds    5             ; The real password saved for encrypting.
B3D6    haspassword             ds    1             ; Non-zero if has password.
        ; passwdmsg             ds    2
B3D7    passwdlen               ds    1
B3D8    passwordlocked          ds    1             ; Non-zero if locked (disallow soft reset).
B3D9    editingsecret           ds    1             ; Non-zero when editing secret file
                                                    ; (can't delete it).
B3DA    inmenu?                 ds    1             ; Non-zero when inside menu -
                                                    ; macros disabled.

B3DB    macro_count             ds    1
B3DC    recording?              ds    1
B3DD    macro_token             ds    2

B3DF    printfailed             ds    1             ; Flag set by mccheckprinter.
                                                    ; Stops "finished printing" message.
B3E0    wasmemoryerr            ds    1
B3E1    inprotext               ds    1             ; Used in file selector,
                                                    ; 0 = was Fn-L,
                                                    ; non-zero = Fn-2
```