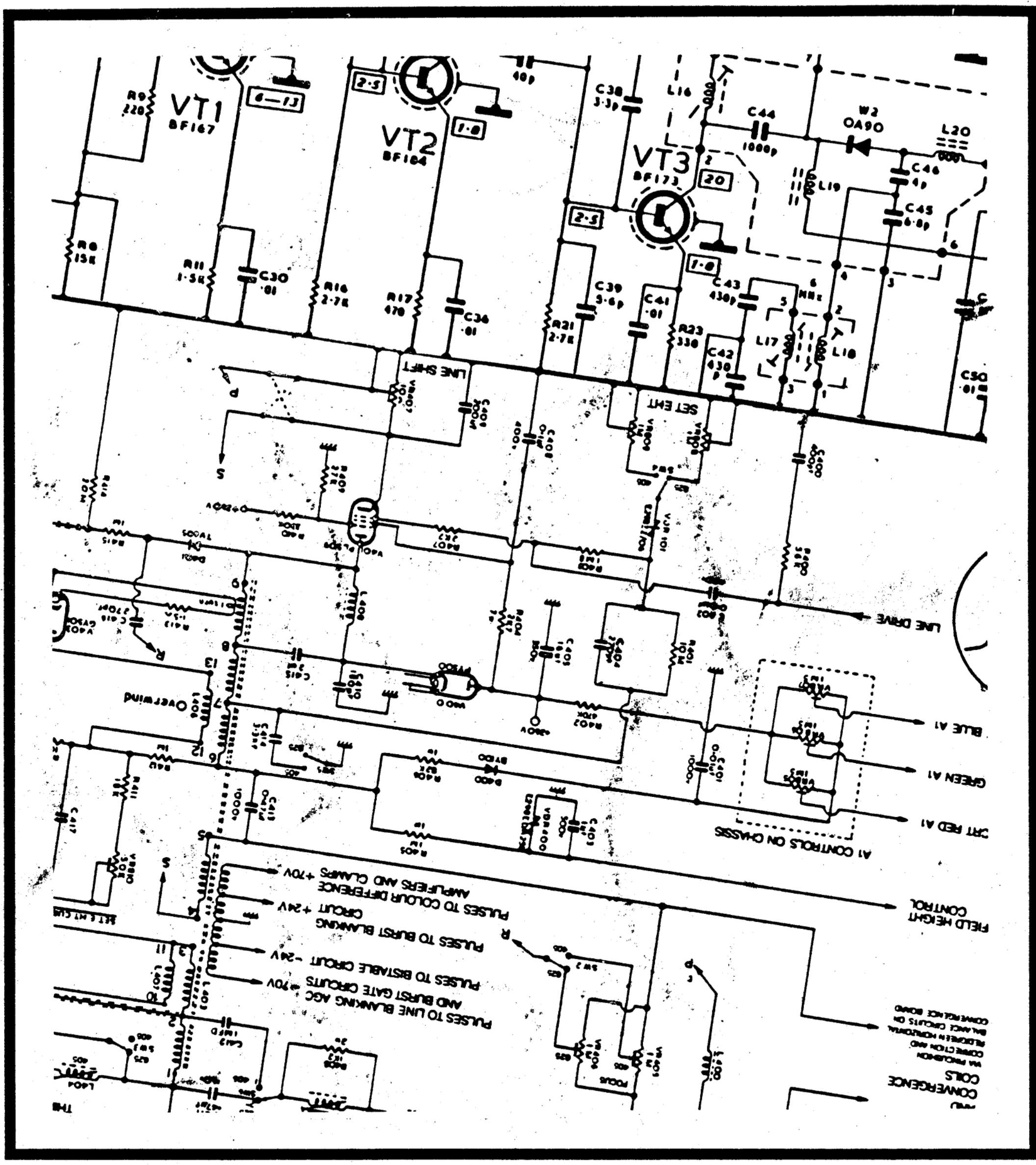
THE FIRMWARE GUIDE

by Bob Taylor and Thomas Defoe

for the Amstrad CPC and Plus computers



The essential programmers' guide includes:

- a complete memory map of all system variables
- descriptions of all the firmware calls and indirections
- a list of the entire Z80 commands and their op-codes
- a chart for conversion of 6128 to 464 memory addresses
- the 'undocumented' maths firmware routines
- binary–decimal–hexadecimal conversion tables

Produced by the original authors of Print-Out

Also available...

A program tape and disc have been produced to accompany this guide. Each contains a set of programs that have been designed to make using your CPC or Plus computer easier — in particular, these include several routines to enable you to program your computer in Machine Code. The list of programs is printed below:

- a full-featured assembler, which uses standard Z80 mnemonics — the source code is incorporated into a BASIC listing in order to enable routines to be easily edited, saved and called
- an extensive monitor, which allows you to single step through a machine code routine, or to set the computer to simulate running the program, complex branching is possible, and it is possible to alter any registers; the program also disassembles code and allows you to investigate your CPC's memory and settings
- the ROMCALL and RAMCALL programs from this guide are also included
- a program to time the number of T-states taken by an instruction
- an RSX that provides the 464 with an AUTO command similar to that used on the 6128 (instead of an asterisk being printed if a line already exists, the contents of the line are displayed and are ready for editing)
- programs to enable or disable ROMs, name discs for easy reference, provide an on-screen clock, list any peripherals that may be attached to your CPC or Plus, and a routine to allow Plus owners to use their 8-bit printer port fully
- also includes a useful selection of short routines

When bought individually, the program tape costs £2.50 and the program disc costs £4.50, and both of these prices include postage and packing. For more details, either ask the place where you bought this guide, or contact the publishers direct.

Published by

Bob Taylor & Thomas Defoe
8 Maze Green Road
Bishop's Stortford
Herts CM23 2PJ

Introduction

WELCOME TO THE FIRMWARE GUIDE

Computer programming is one of the most satisfying hobbies as it is a rare opportunity to invent, develop and test your own ideas and see them come to fruition. Competent programming is a skill which is not easy to master, but once learnt, it will give hours of pleasure — there are many people who spend hours inventing new coding tricks, solving complex problems or just trying out an idea.

Fortunately, when Amstrad developed the CPC and Plus computers, they let the user access many of the computer's routines and use the Firmware in their own programs. Experienced programmers will no doubt write faster routines which are more efficient, or have some special feature, yet these extra facilities can easily be patched in using the Firmware Jumpblock.

For many years, Amstrad produced the definitive guide to the insides of the CPC but sale of this was stopped almost three years ago. Since then, the Firmware Manual has been regarded as an antique by those who are fortunate enough to own a copy. Nevertheless, the original guide had some omissions, notably the absence of information on the system variables and the Z80 processor inside every CPC or Plus.

This guide is not intended to explain how to program in machine code, but we hope that it will supply the information needed to make the most of the Amstrad's capabilities when writing your own programs.

Bob Taylor and Thomas Defoe, 1992

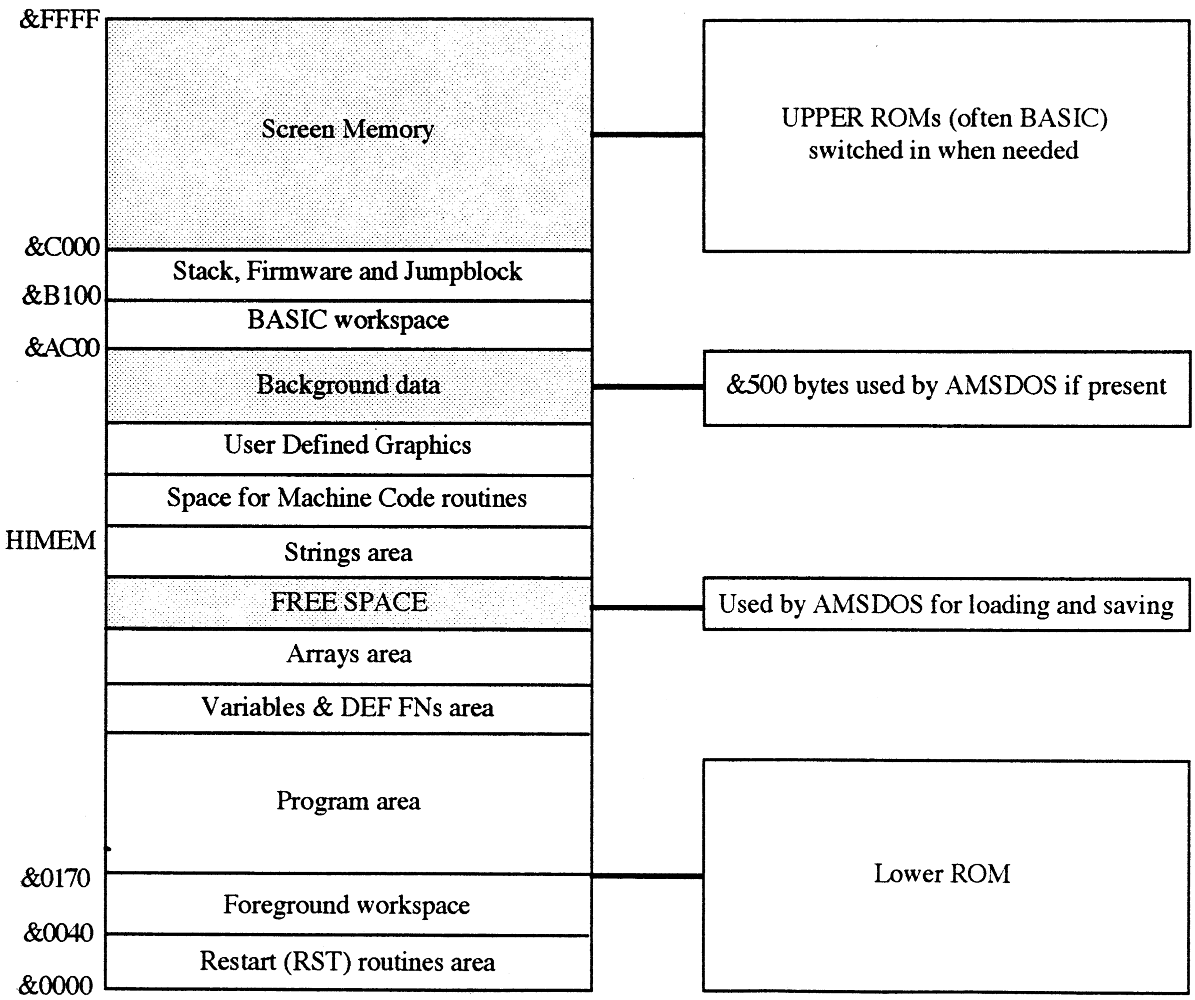
The Firmware Guide — Index

Use of memory by the Operating System

The following list of memory addresses and their uses has been compiled over a number of years, mainly from personal investigation. It does not claim to be definitive, since no accurate source seems to be available to the average computer user, and so may be inaccurate or deficient at certain points; also, some of the areas described have uses additional to those listed. We have tried to make it as accurate as possible, to enable others to use to the full those facilities which present themselves via this information.

- Addresses and values are present in memory with the low byte first
- The term 'above' means higher in memory
- Areas with numbers of bytes of either &00 or &FF given in brackets, may be safe to use for machine code routines etc, as may the tape area, and the Sound ENT and ENT areas if these are unused
- The first column given is the address (for the 6128) of the memory being considered, while the second column gives the equivalent 464 address — unfortunately the 464 differs from the 6128 for most addresses, so if one address is omitted, the system variable is not available for that machine.
- The next column gives the size allocated in bytes. Addresses on the right hand side enclosed in brackets are of System Variables which hold the address of the bytes being explained. With addresses or values anywhere in the text, the value shown is for the 6128; a value in italics is for the 464 only

Overview of the CPC's memory



6128	464	Size	Comments on the memory locations	
&0000	&0000	&40	**Restart block:**	
&0000	&0000		RST 0: complete machine reset	
&0008	&0008		RST 1: LOW JUMP: in-line two byte address: b0 to b13=address; b14=Low ROM disabled; b15=Upper ROM disabled	
&000B	&000B		LOW PCHL: HL has address as RST 1	
&000E	&000E		PCBC INSTRUCTION: BC has address to jump to	
&0010	&0010		RST 2: SIDE CALL: in-line two byte address: b0 to b13=address–&C000; b14 to b15=offset to required ROM (used between sequenced Foreground ROMs)	
&0013	&0013		SIDE PCHL: HL has address as RST 2	
&0016	&0016		PCDE INSTRUCTION: DE has address to jump to	
&0018	&0018		RST 3: FAR CALL: in-line three byte address block: bytes 1 and 2 hold the address; byte 3 holds the ROM select address	
&001B	&001B		FAR PCHL: as RST 3, but HL has address; C has ROM select	
&001E	&001E		PCHL INSTRUCTION: HL has address to jump to	
&0020	&0020		RST 4: RAM LAM: LD A,(HL) from RAM with ROMs disabled	
&0023	&0023		FAR CALL: as RST 3, but HL has address of three byte address block	
&0028	&0028		RST 5: FIRM JUMP: in-line two byte address to jump to	
&0030	&0030		RST 6: User restart; default to RST 0	
&0038	&0038		RST 7: Interrupt entry (KB/Time etc)	
&003B	&003B		External interrupt (default to RET)	

&0040	&0040	&130	**ROM lower foreground area:** BASIC input area (tokenised)	(&AE62 – *&AE7F*)
&016F	&016F		end of BASIC input area	(&AE64 – *&AE81*)
&0170	&0170		**BASIC working area for program, variables, etc** (see opposite)	
&0170	&0170		Program area	
			Variables and DEF FNs area	(&AE66,&AE68 – *&AE83,&AE85*)
			Arrays area	(&AE6A – *&AE87*)
			Free space	(&AE6C – *&AE89*)
			end of free space	(&B071 – *&B08D*)
			Strings area	
			end of Strings area (=HIMEM)	(&AE5E,&B073 – *&AE7B,&B08F*)
			Space for user machine code routines	
			end of user space, byte before user defined graphics area	(*&AE7D*)
		n*8	User defined graphics area	(&B736 – *&B296*)
			end of UDG area	(&AE60 – *&B096*)
			ROM Upper reserved area, expandible during KL ROM WALK, including:	
		r*4	ROM chaining blocks (arranged as follows):	
&A6FC	&A6FC	4	AMSDOS chaining block:	
&A6FC	&A6FC	2	address of next ROM block in chain (or &0000 if the last in chain)	
&A6FE	&A6FE	1	ROM Select address	
&A6FF	&A6FF	1	&00	

&A700	&A700	&500	**AMSDOS reserved area**	(&BE7D,&BBE8 – *&BE7D,&B1B8*)
			this area is moved down if any ROMs have numbers greater than eight (6128 only)	
&A700	&A700	1	current drive number (0=A; 1=B)	
&A701	&A701	1	current USER number	
&A702	&A702	1	flag?	
&A703	&A703	2	address?	
&A705	&A705	1	flag?	
&A706	&A706	2	address?	

<> means ‘not the value or bit which follows’
b0 signifies bit 0, etc HB means ‘the most significant byte’ and LB means ‘the least significant byte’
addresses on the right are of System Variables that hold the address of the byte being explained (for the 464 they are in italics)

6128	464	Size	Comments on the memory locations
&A708	&A708	1	OPENIN flag (&FF=closed; <>&FF=opened)
&A709	&A709	&20	**Copy of current or last Disc Directory entry for OPENIN/LOAD:**
&A709	&A709	1	USER number
&A70A	&A70A	8	filename (padded with spaces)
&A712	&A712	3	file extension (BAS, BIN, BAK, etc) including:
&A712	&A712	1	b7 set = Read Only
&A713	&A713	1	b7 set = System (ie not listed by CAT or DIR)
&A715	&A715	1	16K block sequence number for this directory entry (0 for first block; if <>0 part of a larger file)
&A716	&A716	2	unused
&A718	&A718	1	length of this block in 128 byte records
&A719	&A719	16	sequence of Disc Block numbers containing file — &00 as end marker
&A729	&A729	1	number of 128 byte records loaded so far; before loading proper: &00 for ASCII (ie nothing loaded yet); &01 for BIN or BAS files (ie header record loaded)
&A72A	&A72A	1	
&A72B	&A72B	1	

&A72C	&A72C	1	OPENOUT flag (&FF=closed; <>&FF=opened)
&A72D	&A72D	&20	**Copy of current or last Disc Directory entry for OPENOUT/SAVE:**
&A72D	&A72D	1	USER number
&A72E	&A72E	8	filename (padded with spaces)
&A736	&A736	3	file extension (.$$$ while open; correct extension when finished)
&A739	&A739	1	flag (&00=open; &FF=closed, ie finished)
&A73A	&A73A	1	
&A73B	&A73B	1	flag (&00=open; &FF=closed)
&A73C	&A73C	1	number of 128 byte records saved so far
&A73D	&A73D	16	sequence of Disc Block numbers containing file — &00 as end marker
&A74D	&A74D	1	number of 128 byte records saved so far
&A74E	&A74E	1	
&A74F	&A74F	1	

&A750	&A750	1	flag (&00=OPENIN; &01=In Char; &02=In Direct (whole file))
&A751	&A751	2	address of 2K buffer for ASCII, or of start of current/last block if BIN or BAS file
&A753	&A753	2	address of next byte to read for ASCII, or of 2K buffer for BAS or BIN file

&A755	&A755	&45	**first &45 bytes of BAS/BIN file** (extended header) or of extended header made for ASCII file
&A755	&A755	1	USER number
&A756	&A756	8	filename (padded)
&A75E	&A75E	3	extension
&A761	&A761	6	unused
&A767	&A767	1	file type (&00=BASIC; &01=protected BASIC; &02=Binary; &16=ASCII)
&A768	&A768	2	unused
&A76A	&A76A	2	address to load file into (=actual destination), or buffer for an ASCII file
&A76C	&A76C	1	unused for disc
&A76D	&A76D	2	length of file in bytes (&0000 for ASCII files)
&A76F	&A76F	2	execution address for a BIN file
&A770	&A770	&25	unused
&A795	&A795	3	length of actual file in bytes (as &A76D) — BAS and BIN only
&A798	&A798	2	simple checksum of first 67 bytes of header (LB first) — BAS and BIN only

&A79A	&A79A	1	flag (&00=OPENOUT; &01=Out Char; &02=Out Direct (whole file))
&A79B	&A79B	2	address of 2K block if an ASCII file, or of current/last block saved if a BAS or BIN file
&A79D	&A79D	2	address of next byte to write for ASCII files, or of 2K buffer for BAS and BIN files

6128	464	Size	Comments on the memory locations	
&A79F	&A79F	&45	**first &45 bytes of BAS/BIN file** (ie extended header)	
&A79F	&A79F	1	USER number	
&A7A0	&A7A0	8	filename (padded)	
&A7A8	&A7A8	3	extension	
&A7AB	&A7AB	1	flag (&00=Open)	
&A7AC	&A7AC	1		
&A7AD	&A7AD	1	flag (&00=Open)	
&A7AE	&A7AE	3	unused	
&A7B1	&A7B1	1	file type (&00=BASIC; &01=protected BASIC; &02=Binary; &16=ASCII)	
&A7B2	&A7B2	2	unused	
&A7B4	&A7B4	2	address to save file from (for BAS or BIN files), or of buffer for ASCII files	
&A7B6	&A7B6	1	unused for disc	
&A7B7	&A7B7	2	length of file in bytes	
&A7B9	&A7B9	2	execution address for BIN files	
&A7BB	&A7BB	&25	unused	
&A7DF	&A7DF	3	length of actual file in bytes (as at &A7B7) — BAS and BIN only	
&A7E2	&A7E2	2	simple checksum of first 67 bytes of header (LB first) — BAS and BIN only	
&A7E4	&A7E4	&80	buffer area for records sent to or loaded from Disc, or used in forming filename and extension	
&A864	&A864	14*3	Tape Jumpblock is stored here by AMSDOS — is moved to &BC77 etc after ǀTAPE	
&A88B	&A88B	3	far address used by AMSDOS RST 3s at &BC77 etc (&CD30,&07)	
&A890	&A890	&19	**Drive A Extended Disc Parameter Block (XDPB):**	(&BE42,&A91A*)
&A890	&A890	2	number of 128 byte records per track	
&A892	&A892	1	$\log_2$(Block size)–7 (&03=1024 bytes; &04=2048 bytes)	
&A893	&A893	1	(Block size)/128–1 (&07=1024 bytes; &0F=2048 bytes)	
&A894	&A894	1	(Block size)/1024 (if total of blocks <256, else /2048)–1	
&A895	&A895	2	number of blocks per disc side (excluding reserved tracks)	
&A897	&A897	2	number of (directory entries)–1	
&A899	&A899	2	bit significant value of number of blocks for directory (&0080=1; &00C0=2)	
&A89B	&A89B	2	number of bits in checksum = ((&A894)+1)/4	
&A89D	&A89D	2	number of reserved tracks (&00=Data; &01=IBM; &02=System)	
&A89F	&A89F	1	number of first sector (&01=IBM; &41=System; &C1=Data)	
&A8A0	&A8A0	1	number of sectors per track (Data=9; System=9; IBM=8)	
&A8A1	&A8A1	1	gap length (Read/Write)	
&A8A2	&A8A2	1	gap length (Format)	
&A8A3	&A8A3	1	format filler byte (&E5)	
&A8A4	&A8A4	1	$\log_2$(sector size)–7 (&02=512; &03=1024)	
&A8A5	&A8A5	1	records per sector	
&A8A6	&A8A6	1	current track (not for use)	
&A8A7	&A8A7	1	0=not aligned (not for use)	
&A8A8	&A8A8	1	Auto select flag (&00=Auto select; &FF= don't alter)	
&A8A9	&A8A9			(&A91C*)
&A8B9	&A8B9			(&A91E*)
&A8D0	&A8D0	&19	**Drive B Extended Disc Parameter Block** (arranged as at &A890)	(&A92A*)
&A8E9	&A8E9		(&17 bytes of &FF)	(&A92C*)
&A8F9	&A8F9			(&A92E*)
&A900	&A900		(&12 bytes of &00)	
&A910	&A910			(&BE40*)

<> means 'not the value or bit which follows' '*' means this address applies to all machines with a disc drive fitted
b0 means bit 0, b1 means bit 1, etc HB means 'the most significant byte' and LB means 'the least significant byte'
addresses on the right are of System Variables that hold the address of the byte being explained (for the 464 they are in italics)

6128	464	Size	Comments on the memory locations	
&A918	&A918	2	address of area for reading directory entries for Drive A	
&A91A	&A91A	2	address of Drive A XDPB	
&A91C	&A91C	2	address of the byte after the end of Drive A XDPB	
&A91E	&A91E	2		
&A920	&A920		(8 bytes of &00)	
&A928	&A928	2	address of area for reading directory entries for Drive B	
&A92A	&A92A	2	address of Drive B XDPB	
&A92C	&A92C	2	address of the byte after the end of Drive B XDPB	
&A92E	&A92E	2		
&A930	&A930	&80	block of directory entries, including last file loaded	(&A918,&A928*)
&A9B0	&A9B0	&200	buffer for loading; usually contains last sector loaded	(&BE62,&BE76*)
&ABB0	&ABB0		(&50 bytes of &00)	
&AC00	&AC00		**Start of BASIC Operating System reserved area:**	
&AC00	&AC00	1	program line redundant spaces flag (0=keep extra spaces; <>0=remove extra spaces)	
	&AC01	9*3	groups of 3 RET bytes (&C9) called by the Upper ROM	
&AC01	&AC1C	1	AUTO flag (0=off; <>0=on)	
&AC02	&AC1D	2	number of the next line (6128) or of the current line (464) for AUTO	
&AC04	&AC1F	2	step distance for AUTO	
&AC06	&AC21	1		
&AC07	&AC22	1		
&AC08		1		
	&AC23	1		
&AC09	&AC24	1	WIDTH (&84=132)	
&AC0A	&AC25	1		
&AC0B		1		
&AC0C	&AC26	1	FOR/NEXT flag (0=NEXT not yet used; <>0=used)	
&AC0D	&AC27	5	FOR start value (real) — only two bytes are used if % or DEFINT variable	
&AC12	&AC2C	2	address of ':' or of the end of program line byte after a NEXT command	
&AC14	&AC2E	2	address of LB of the line number containing WEND	
&AC16	&AC30	1	WHILE/WEND flag (&41=WEND not yet used; &04=used)	
&AC17	&AC31	1		
&AC18	&AC32	2		
&AC1A	&AC34	2		
&AC1C	&AC36	2	address (&B65B – *&B511*) of location holding ROM routine address for KB event block	
&AC1E	&AC38	&0C	**Event Block for ON SQ(1):**	
&AC1E	&AC38	2	chain address to next event block; &0000 if last in chain, but &FFFF if unused	
&AC20	&AC3A	1	count	
&AC21	&AC3B	1	class: Far address, highest (ON SQ) priority, Normal & Synchronous event	
&AC22	&AC3C	2	routine address (in BASIC ROM, &C926 – *&C879*)	
&AC24	&AC3E	1	ROM Select number (&FD ie ROM 0 enabled, Lower ROM disabled)	
&AC25	&AC3F	1	(first byte of user field)	
&AC26	&AC40	2	address of the end of program line byte or ':' after 'ON SQ(x) GOSUB line number' statement	
&AC28	&AC42	2	address of the end of program line byte of the line before the GOSUB routine	
&AC2A	&AC44	&0C	**Event block for ON SQ(2)**, arranged as (&AC1E – *&AC38*) — second ON SQ priority	
&AC36	&AC50	&0C	**Event block for ON SQ(4)**, arranged as (&AC1E – *&AC38*) — lowest ON SQ priority	
&AC42	&AC5C	&12	**Ticker and Event Block for AFTER/EVERY Timer 0**	
&AC42	&AC5C	2	chain address to next event block (usually to another timer or &00FF)	
&AC44	&AC5E	2	'count down' count	
&AC46	&AC60	2	recharge count (for EVERY only — &0000 if AFTER)	

6128	464	Size	Comments on the memory locations
&AC48	&AC62	2	chain address to next ticker block
&AC4A	&AC64	1	count
&AC4B	&AC65	1	class: Far address, lowest (timer) priority, Normal and Synchronous event
&AC4C	&AC66	2	Routine address (in BASIC ROM, at &C926 – *&C879*)
&AC4E	&AC68	1	ROM Select No (&FD ie ROM 0 enabled, Lower ROM disabled)
&AC4F	&AC69	1	(first byte of user field)
&AC50	&AC6A	2	address of the end of program line byte or ':' after statement in use when the timer 'timed-out'
&AC52	&AC6C	2	address of the end of program line byte of the line before the GOSUB routine
&AC54	&AC6E	&12	**Ticker and Event Block for AFTER/EVERY Timer 1** (3rd Timer priority) arranged as &AC42 – *&AC5C*
&AC66	&AC80	&12	**Ticker and Event Block for AFTER/EVERY Timer 2** (2nd Timer priority) arranged as &AC42 – *&AC5C*
&AC78	&AC92	&12	**Ticker and Event Block for AFTER/EVERY Timer 3** (highest priority) arranged as &AC42 – *&AC5C*

6128	464	Size	Comments on the memory locations
&AC8A	&ACA4	&100	**BASIC input area for lines** (as typed in and not tokenised) or for INPUT
&AD8C	&ADA6	2	address of line number LB in line containing error
&AD8E	&ADA8	2	address of byte before statement containing error — ie of ':' or Line No HB
&AD90	&ADAA	1	ERR (Error No)
&AD91		1	DERR (Disc Error No)
&AD92	&ADAB	2	as (&AD8E – *&ADA8*) if error is in a program (ie not if in Direct Command Mode)
&AD94	&ADAD	2	as (&AD8C – *&ADA6*) if error is in a program (ie not if in Direct Command Mode)
&AD96	&ADAF	2	address of the length LB of line specified by the 'ON ERROR GOTO' command
&AD98	&ADB1	1	

6128	464	Size	Comments on the memory locations
&AD99	&ADB2	&09	**Current SOUND parameter block** (see Firmware Jump &BCAA):
&AD99	&ADB2	1	channel and rendezvous status
&AD9A	&ADB3	1	amplitude envelope (ENV) number
&AD9B	&ADB4	1	tone envelope (ENT) number
&AD9C	&ADB5	2	tone period
&AD9E	&ADB7	1	noise period
&AD9F	&ADB8	1	initial amplitude
&ADA0	&ADB9	2	duration, or envelope repeat count
&ADA2	&ADBB	&10	**Current Amplitude or Tone Envelope parameter block** (see &BCBC or &BCBF)
&ADA2	&ADBB	1	number of sections (+&80 for a negative ENT number, ie the envelope is run until end of sound)
&ADA3	&ADBC	3	first section of the envelope:
&ADA3	&ADBC	1	step count (if <&80) otherwise envelope shape (not tone envelope)
&ADA4	&ADBD	1	step size (if step count<&80) otherwise envelope period (not tone envelope)
&ADA5	&ADBE	1	pause time (if step count<&80) otherwise envelope period (not tone envelope)
&ADA6	&ADBF	3	second section of the envelope (as &ADA3 – *&ADBC*)
&ADA9	&ADC2	3	third section of the envelope (as &ADA3 – *&ADBC*)
&ADAC	&ADC5	3	fourth section of the envelope (as &ADA3 – *&ADBC*)
&ADAF	&ADC8	3	fifth section of the envelope (as &ADA3 – *&ADBC*)
&ADB2	&ADCB	5	
&ADB7	&ADD0	&36	
&ADEB	&AE04	2	
&ADED	&AE06	6	

<> means 'not the value or bit which follows' '*' means this address applies to all machines with a disc drive fitted
b0 means bit 0, b1 means bit 1, etc HB means 'the most significant byte' and LB means 'the least significant byte'
addresses on the right are of System Variables that hold the address of the byte being explained (for the 464 they are in italics)

6128	464	Size	Comments on the memory locations
&ADF3	&AE0C	26*1	table of DEFINT (&02), DEFSTR (&03) or DEFREAL/default (&05), for variables 'a' to 'z'
&AE0D	&AE26	1	
&AE0E	&AE27	2	
&AE10	&AE29	2	
&AE12	&AE2B	2	
&AE14	&AE2D	1	
&AE15	&AE2E	2	address of line number LB of last BASIC line (or &FFFF)
&AE17	&AE30	2	address of byte before next DATA item (eg comma or space)
&AE19	&AE32	2	address of next space on GOSUB etc stack (see also &B06F – *&B08B*)
&AE1B	&AE34	2	address of byte before current statement (&003F if in Direct Command mode)
&AE1D	&AE36	2	address of line number LB of line of current statement (&0000 if in Direct Command mode)
&AE1F	&AE38	1	trace flag (0=TROFF; <>0=TRON)
&AE20	&AE39	1	flag used with Trace (&00 if in Direct Command mode; &01 if in a program)

&AE21	&AE3A	1	
&AE22	&AE3B	2	
&AE24	&AE3D	2	
&AE26	&AE3F	2	address to load cassette file to
&AE28	&AE41	1	
&AE29	&AE42	1	file type from cassette header
&AE2A	&AE43	2	file length from cassette header
&AE2C	&AE45	1	program protection flag (<>0 hides program as if protected)
&AE2D	&AE46	17	buffer used to form binary or hexadecimal numbers before printing etc
&AE3A	&AE53	5	start of buffer used to form hexadecimal numbers before printing etc
&AE3A	&AE53	1	Key Number used with INKEY (providing the Key Number is written as a decimal)
&AE3E	&AE57	1	last byte (usually &00 or &20) of the formed binary or hexadecimal number
&AE43	&AE5D	13	buffer used to form decimal numbers before printing etc
&AE4E	&AE68	1	last byte (usually &00 or &20) of the formed decimal number
	&AE6B	3	
&AE51		1	
&AE52	&AE6E	2	
&AE54		1	
	&AE70	2	temporary store for address after using (&AE68)

&AE55	&AE72	2	address of last used ROM or RSX JUMP instruction in its Jump Block
&AE57	&AE74	1	ROM Select number if address above is in ROM
&AE58	&AE75	2	BASIC Parser position, moved on to ':', or the end of program line byte after a CALL or an RSX
&AE5A	&AE77	2	the resetting address for machine Stack Pointer after a CALL or an RSX
&AE5C	&AE79	2	ZONE value
&AE5D		1	
	&AE7A	1	

&AE5E	&AE7B	2	HIMEM (set by MEMORY)
	&AE7D	2	address of the byte before the UDG area (the end of the user M/C routine area or the Strings area) if the UDG area is still present, otherwise the highest byte of Program etc area
&AE60		2	address of highest byte of free RAM (ie last byte of UDG area)
&AE62	&AE7F	2	address of start of ROM lower reserved area (used for tokenised lines)
&AE64	&AE81	2	address of end of ROM lower reserved area (byte before Program area)
&AE66	&AE83	2	as (&AE68 – *&AE85*)
&AE68	&AE85	2	address of start of Variables and DEF FNs area
&AE6A	&AE87	2	address of start of Arrays area (where next Variable or DEF FN entry is placed)
&AE6C	&AE89	2	address of start of free space (where next Array entry is placed)
&AE6E		1	

6128	464	Size	Comments on the memory locations
&AE70	&AE8C	&1FF	**GOSUB, FOR and WHILE stack.** Entries are added above any existing ones in use (mixed as encountered) at address given by (&B06F — *&B08B*) and must be used up in the opposite order. Completed entries are not deleted, just overwritten by the next new entry:
			GOSUB (84 max capacity):
		1	(byte of &00)
		2	address of end of program line byte or ':' after GOSUB statement (the point to RETURN to)
		2	address of line number HB of line containing GOSUB
		1	byte of &06, ie the length of the GOSUB entry
			FOR (21 max for Real FORs, 31 max for Integer FORs):
		2	address of current value of control variable (in Variables area)
		5	value of limit (ie the TO value) — there are two bytes only for Integer FORs
		5	value of STEP — two bytes for Integer FORs
		1	sign byte (&00 for positive; &01 for negative)
		2	address of the end of program line byte, or ':' after the FOR statement (ie the address for the NEXT loop to restart at)
		2	address of line number LB of line containing FOR
		2	address of byte after NEXT statement (ie the address to continue at when the limit is exceeded)
		2	address of byte after NEXT statement again
		1	length byte (&16 for Real FORs; &10 for Integer FORs)
			WHILE (66 max capacity):
		2	address of line number LB of line containing WHILE
		2	address of the end of program line byte or ':' after WEND statement (ie the address to continue at when the condition is false)
		2	address of condition after the WHILE command
		1	length byte of &07 — end of WHILE entry proper but:
		+5	value of condition (0 or –1 as a floating point value) only while the WHILE entry is the last on the stack

NB: The free space on the stack is also used as a workspace by various routines for values and addresses and for Variable names

&B06F	&B08B	2	address of the next space on the GOSUB etc stack (see also &AE19 – *&AE32*)

&B071	&B08D	2	address of end of free space (the byte before the Strings area)
&B073	&B08F	2	address of end of Strings area (=HIMEM)
&B075		1	
	&B091	1	
	&B092	2	
&B076	&B094	2	
&B078	&B096	2	address of the highest byte of free RAM disregarding UDGs (usually &A6FB – *&ABFB*)
&B07A	&B098	2	
&B07C	&B09A	2	address for the next entry in the String Concatenation area
&B07E	&B09C	10*3	concatenation area holding descriptors of strings being used
&B09C	&B0BA	1	length of last String used
&B09D	&B0BB	2	address of last String used
	&B0BD	2	
	&B0BF	2	
&B09F	&B0C1	1	type byte used with the Virtual Accumulator (&02=Integer; &03=String; &05=Real)
&B0A0	&B0C1	5	Virtual Accumulator used by the maths routines (two bytes for an Integer value; three bytes for a String Descriptor; five bytes for a Real value):

<> means 'not the value or bit which follows' '*' means this address applies to all machines with a disc drive fitted
b0 means bit 0, b1 means bit 1, etc HB means 'the most significant byte' and LB means 'the least significant byte'
addresses on the right are of System Variables that hold the address of the byte being explained (for the 464 they are in italics)

6128	464	Size	Comments on the memory locations
&B0A0	&B0C2	2	
&B0A2	&B0C4	1	
&B0A3	&B0C5	2	
&B0A5	&B0C7		&5B (*&39*) bytes of &FF
&B100	&B8E4	2	&07, &C6
&B102	&B8E6	2	&65, &89
&B104	&B8E8	5	
&B109	&B8ED	5	
&B10E	&B8F2	5	
&B113	&B8F7	1	DEG/RAD flag (&00=RAD; &FF=DEG)
&B114	&B8DC	1	
&B115	&B8DD	1	
&B116	&B8DE	1	
&B117	&B8DF	1	

6128	464	Size	Comments on the memory locations
&B118	&B800	&D2	**Area used for Cassette handling:**
&B118	&B800	1	cassette handling messages flag (0=enabled; <>0=disabled)
&B119	&B801	1	
&B11A	&B802	1	file IN flag (&00=closed; &02=IN file; &03=opened; &05=IN char)
&B11B	&B803	2	address of 2K buffer for directories
&B11D	&B805	2	address of 2K buffer for loading blocks of files — often as &B11B – *&B803*
&B11F	&B807	&40	**IN Channel header:**
&B11F	&B807	&10	filename (padded with NULs)
&B12F	&B817	1	number of block being loaded, or next to be loaded
&B130	&B818	1	last block flag (&FF=last block; &00=not)
&B131	&B819	1	file type (&00=BASIC; &01=Protected BASIC; &02=Binary; &08=Screen; &16=ASCII)
&B132	&B81A	2	length of this block
&B134	&B81C	2	address to load this or the next block at, or the address of the byte after last one loaded
&B136	&B81E	1	first block flag (&FF=first block; &00=not)
&B137	&B81F	2	total length of file (all blocks)
&B139	&B821	2	execution address for BIN files (&0000 if not saved as such)
&B13B	&B823	&24	not allocated
&B15F	&B847	1	file OUT flag (&00=closed; &02=IN file; &03=opened; &05=IN char)
&B160	&B848	2	address to start the next block save from, or the address of the buffer if it is OPENOUT
&B162	&B84A	2	address of start of the last block saved, or the address of the buffer if it is OPENOUT
&B164	&B84C	&40	**OUT Channel Header (details as IN Channel Header):**
&B164	&B84C	&10	filename
&B174	&B85C	1	number of the block being saved, or next to be saved
&B175	&B85D	1	last block flag (&FF=last block; &00=not)
&B176	&B85E	1	file type (as at &B131 – *&B819*)
&B177	&B85F	2	length saved so far
&B179	&B861	2	address of start of area to save, or address of buffer if it is an OPENOUT instruction
&B17B	&B863	1	first block flag (&FF=first block; &00=not)
&B17C	&B864	2	total length of file to be saved
&B17E	&B866	2	execution address for BIN files (&0000 if parameter not supplied)
&B180	&B868	&24	not allocated
&B1A4	&B88C	&40	used to construct IN Channel header:
&B1B5	&B89D	1	
&B1B7	&B89F	2	
&B1BB	&B8A3	1	
&B1BE	&B8A6	1	

6128	464	Size	Comments on the memory locations
&B1B9	&B51D		base address for calculating relevant Sound Channel block
&B1BC	&B520		base address for calculating relevant Sound Channel ?
&B1BE	&B522		base address for calculating relevant Sound Channel ?
&B1D5	&B539		base address for calculating relevant Sound Channel ?
&B1E4	&B8CC	1	

&B1E5	&B8CD	1	synchronisation byte
&B1E6	&B8CE	2	&55, &62
&B1E8	&B8D0	1	
&B1E9	&B8D1	1	cassette precompensation (default &06; SPEED WRITE 1 &0C @ 4uS)
&B1EA	&B8D2	1	cassette 'Half a Zero' duration (default &53; SPEED WRITE 1 &29 @ 4uS)
&B1EB	&B8D3	2	

	&B550	1	used by sound routines
	&B551	1	used by sound routines
&B1ED		1	used by sound routines
&B1EE	&B552	1	used by sound routines
&B1F0	&BB54	1	used by sound routines
	&BB55	7	used by sound and cassette routines

&B1F8	&B55C	&3F	**Sound Channel A (1) data:**
&B212	&B576	1	number of sounds still queuing
&B213	&B577	1	number of sounds originally queuing
&B217	&B57B	8	first or fifth sound in Channel 1 (A) queue:
&B217	&B57B	1	status: b0 to b2 = rendezvous with channel 1, 2 or 4; b3 = Hold; b4 = Flush
&B218	&B57C	1	b0 to b3 = tone envelope number; b4 to b7 = volume envelope number (ie ENV number*16)
&B219	&B57D	2	pitch
&B21B	&B57F	1	noise
&B21C	&B580	1	volume
&B21D	&B581	2	duration (in 0.01 seconds)
&B21F	&B583	8	second sound in Channel 1 queue (as &B217 – *&B57B*)
&B227	&B58B	8	third sound in Channel 1 queue (as &B217 – *&B57B*)
&B22F	&B593	8	fourth sound in Channel 1 queue (as &B217 – *&B57B*)

&B237	&B59B	&3F	**Sound Channel B (2) data:** — as described at &B1F8 – *&B55C*
&B256	&B5BA	8	first or fifth sound in Channel 2 queue (as &B217 – *&B57B*)
&B25E	&B5C2	8	second sound in Channel 2 queue (as &B217 – *&B57B*)
&B266	&B5CA	8	third sound in Channel 2 queue (as &B217 – *&B57B*)
&B26E	&B5D2	8	fourth sound in Channel 2 queue (as &B217 – *&B57B*)

&B276	&B5DA	&3F	**Sound Channel C (4) data** — as described at &B1F8 – *&B55C*
&B295	&B5F9	8	1st/5th sound in Channel 4 queue (as &B217 – *&B57B*)
&B29D	&B601	8	2nd sound in Channel 4 queue (as &B217 – *&B57B*)
&B2A5	&B609	8	3rd sound in Channel 4 queue (as &B217 – *&B57B*)
&B2AD	&B611	8	4th sound in Channel 4 queue (as &B217 – *&B57B*)
&B2A6	&B60A		base address for calculating relevant ENV parameter block
&B2B5	&B619	1	&3F

<> means 'not the value or bit which follows' '*' means this address applies to all machines with a disc drive fitted
b0 means bit 0, b1 means bit 1, etc HB means 'the most significant byte' and LB means 'the least significant byte'
addresses on the right are of System Variables that hold the address of the byte being explained (for the 464 they are in italics)

6128	464	Size	Comments on the memory locations
&B2B6	&B61A	15*&10	**ENV parameter block area** (each arranged as &ADA2(&ADBB)):
&B2B6	&B61A	&10	ENV 1
&B2C6	&B62A	&10	ENV 2
&B2D6	&B63A	&10	ENV 3
&B2E6	&B64A	&10	ENV 4
&B2F6	&B65A	&10	ENV 5
&B306	&B66A	&10	ENV 6
&B316	&B67A	&10	ENV 7
&B326	&B68A	&10	ENV 8
&B336	&B69A	&10	ENV 9
&B346	&B6AA	&10	ENV 10
&B356	&B6BA	&10	ENV 11
&B366	&B6CA	&10	ENV 12
&B376	&B6DA	&10	ENV 13
&B386	&B6EA	&10	ENV 14
&B396	&B6FA	&10	ENV 15
&B396	&B6FA		base address for calculating relevant ENT parameter block
&B3A6	&B70A	15*&10	**ENT parameter block area** (each arranged as &ADA2(&ADBB)):
&B3A6	&B70A	&10	ENT 1
&B3B6	&B71A	&10	ENT 2
&B3C6	&B72A	&10	ENT 3
&B3D6	&B73A	&10	ENT 4
&B3E6	&B74A	&10	ENT 5
&B3F6	&B75A	&10	ENT 6
&B406	&B76A	&10	ENT 7
&B416	&B77A	&10	ENT 8
&B426	&B78A	&10	ENT 9
&B436	&B79A	&10	ENT 10
&B446	&B7AA	&10	ENT 11
&B456	&B7BA	&10	ENT 12
&B466	&B7CA	&10	ENT 13
&B476	&B7DA	&10	ENT 14
&B486	&B7EA	&10	ENT 15

&B496 &B34C &50 **Normal Key Table:** (&B68B – *&B541*)

Cur U	Cur R	Cur D	f9	f6	f3	Enter	f.
Cur L	Copy	f7	f8	f5	f1	f2	f0
Clr	[	Return	]	f4		\	
^	-	@	p	;	:	/	.
0	9	o	i	l	k	m	j
8	7	u	y	h	j	n	space
6	5	r	t	g	f	b	v
4	3	e	w	s	d	c	x
1	2	Esc	q	Tab	a	Caps-lock	z
[VT]	[LF]	[BS]	[TAB]	Fire 2	Fire 1		Del

&B4E6 &B39C &50 **Shifted Key Table:** (&B68D – *&B543*)

Cur U	Cur R	Cur D	f9	f6	f3	Enter	f.
Cur L	Copy	f7	f8	f5	f1	f2	f0
Clr	{	Return	}	f4		‘	
£	=	|	P	+	*	?	>
_	)	O	I	L	K	M	<
(	’	U	Y	H	J	N	space
&	%	R	T	G	F	B	V
$	#	E	W	S	D	C	X
!	“	Esc	Q	->	A	Caps-lock	Z
	[VT]	[LF]	[BS]	[TAB]	Fire 2	Fire 1	Del

6128	464	Size	Comments on the memory locations								
&B536	&B3EC	&50	**Control Key Table:**								(&B68F – *&B545*)
			Cur U	Cur R	Cur D	f9	f6	f3	Enter	f.	
			Cur L	Copy	f7	f8	f5	f1	f2	f0	
			Clr	(ESC)	Return	(GS)	f4		(FS)		
			(RS)		(NUL)	(DLE)					
			(US)		(SI)	(HT)	(FF)	(VT)	(CR)		
					(NAK)	(EM)	(BS)	(LF)	(SO)		
					(DC2)	(DC4)	(BEL)	(ACK)	(STX)	(SYN)	
					(ENQ)	(ETB)	(DC3)	(EOT)	(ETX)	(CAN)	
				~	Esc	(DC1)	Ins/Ovrt	(SOH)	Shift-lock	(SUB)	
									Del		
&B586	&B43C	10*1	KB repeats table (each byte/bit applies to all three key tables): 1 byte is used per line of the tables; b0 to b7 give the columns (left to right), repeat if set								(&B691 – *&B547*)
&B590	&B446	&98	DEF KEY's definition area (for Keys &80 to &9F in sequence): each definition has either a single byte of &00 if it is unused/unaltered, or: byte 1: length of definition; bytes 2 to x: definition, either a single key or a string of keys								(&B62B – *&B4E1*)
&B628	&B4DE	1									(&B62D – *&B4E3*)
&B629	&B4DF	1									
&B62A	&B4E0	1									
&B62B	&B4E1	2	address of DEF KEY area								
&B62D	&B4E3	2	address of byte after end of DEF KEY area								
&B62F	&B4E5	1									
&B630	&B4E6	1									
&B631	&B4E7	1	Shift lock flag (&00=off; &FF=on)								
&B632	&B4E8	1	Caps lock flag (&00=off; &FF=on)								
&B633	&B4E9	1	KB repeat period (SPEED KEY — default &02 @ 0.02 seconds)								
&B634	&B4EA	1	KB delay period (SPEED KEY — default &1E @ 0.02 seconds)								
&B635	&B4EB	2*10	**Tables used for key scanning;** bits 0 to 7 give the table columns (from left to right):								
&B635	&B4EB	1	Cur U	Cur R	Cur D	f9	f6	f3	Enter	f.	
&B636	&B4EC	1	Cur L	Copy	f7	f8	f5	f1	f2	f0	
&B637	&B4ED	1	Clr	[	Return	]	f4	Shift	\	Control	
&B638	&B4EE	1	^	-	@	P	;	:	/	.	
&B639	&B4EF	1	0	9	O	I	L	K	M	,	
&B63A	&B4F0	1	8	7	U	Y	H	J	N	space	
&B63B	&B4F1	1	Down	Up	Left	Right	Fire 2	Fire 1	*(joystick 1)*		
&B63B	&B4F1	1	6	5	R	T	G	F	B	V	
&B63C	&B4F2	1	4	3	E	W	S	D	C	X	
&B63D	&B4F3	1	1	2	Esc	Q	Tab	A	Caps-lock	Z	
&B63E	&B4F4	1	Down	Up	Left	Right	Fire 2	Fire 1	*(joystick 2)*		
&B63E	&B4F4	1								Del	
&B63F	&B4F5	1	complement of &B635 – *&B4EB*								
&B640	&B4F6	1	complement of &B636 – *&B4EC*								
&B641	&B4F7	1	complement of &B637 – *&B4ED*								
&B642	&B4F8	1	complement of &B638 – *&B4EE*								
&B643	&B4F9	1	complement of &B639 – *&B4EF*								
&B644	&B4FA	1	complement of &B63A – *&B4F0*								
&B645	&B4FB	1	complement of &B63B – *&B4F1*								
&B646	&B4FC	1	complement of &B63C – *&B4F2*								
&B647	&B4FD	1	complement of &B63D – *&B4F3*								
&B648	&B4FE	1	complement of &B63E – *&B4F4*								

<> means 'not the value or bit which follows' '*' means this address applies to all machines with a disc drive fitted
b0 means bit 0, b1 means bit 1, etc HB means 'the most significant byte' and LB means 'the least significant byte'
addresses on the right are of System Variables that hold the address of the byte being explained (for the 464 they are in italics)

6128	464	Size	Comments on the memory locations
&B64B	&B501		
&B653	&B509	1	
&B654	&B50A	1	
&B655	&B50B	1	
&B656	&B50C	1	
&B657	&B50D	7	event block for KB handling, comprises:
&B657	&B50D	2	chain address
&B659	&B50F	1	count
&B65A	&B510	1	class: express event
&B65B	&B511	2	ROM routine address: &C492 – *&C45E* (&AC1C – *&AC36*)
&B65D	&B513	1	ROM select number: &FD
&B65E	&B514	20*2	store for last keys pressed and each entry is as follows:
			byte 1: +0 to +10=key tables' line number; if bit 5 is set then Shift is pressed; bit 7=Control
			byte 2: b0 to b7=key tables' column number — see &B496 – *&B34C* etc
&B67F	&B67F	2	vestige from the 464?
&B686	&B53C	1	
&B687	&B53D	1	accumulated count of the number of keys pressed (MOD 20)
&B688	&B53E	1	number of keys left in key buffer
&B689	&B53F	1	accumulated count of the number of keys removed from the buffer (MOD 20)
&B68A	&B540	1	
&B68B	&B541	2	address of the normal key table
&B68D	&B543	2	address of the shifted key table
&B68F	&B545	2	address of the control key table
&B691	&B547	2	address of the KB repeats table

&B692		1	
&B693	&B328	2	ORIGIN x
&B695	&B32A	2	ORIGIN y
&B697	&B32C	2	graphics text x position (pixel)
&B699	&B32E	2	graphics text y position (pixel)
&B69B	&B330	2	graphics window x of one edge (pixel)
&B69D	&B332	2	graphics window x of other edge (pixel)
&B69F	&B334	2	graphics window y of one side (pixel)
&B6A1	&B336	2	graphics window y of other side (pixel)
&B6A3	&B338	1	GRAPHICS PEN
&B6A4	&B339	1	GRAPHICS PAPER
&B6A5	&B33A	4*2(*14*)	used by line drawing (and other) routines, as follows:
&B6A7	&B33A	2	x+1( )
&B6A9	&B33C	2	y/2+1( )
&B6AB	&B33E	2	y/2-x( )
&B6AD	&B340	2	
	&B342	2	
&B6AF	&B344	1	
&B6B0	&B345	1	
&B6B1	&B346	1	
&B6B2		1	first point on drawn line flag (<>0=to be plotted; 0=don't plot)
&B6B3		1	line MASK
&B6B4		1	
	&B207	2	
&B6B5	&B20C	1	current stream number

6128	464	Size	Comments on the memory locations	
&B6B6	&B20D	14(*15*)	stream (window) 0 parameter block — arranged as &B726 – *&B285*	
&B6C4	&B21C	14(*15*)	stream (window) 1 parameter block — arranged as &B726 – *&B285*	
&B6D2	&B22B	14(*15*)	stream (window) 2 parameter block — arranged as &B726 – *&B285*	
&B6E0	&B23A	14(*15*)	stream (window) 3 parameter block — arranged as &B726 – *&B285*	
&B6EE	&B249	14(*15*)	stream (window) 4 parameter block — arranged as &B726 – *&B285*	
&B6FC	&B258	14(*15*)	stream (window) 5 parameter block — arranged as &B726 – *&B285*	
&B70A	&B267	14(*15*)	stream (window) 6 parameter block — arranged as &B726 – *&B285*	
&B718	&B276	14(*15*)	stream (window) 7 parameter block — arranged as &B726 – *&B285*	
&B726	&B285	14(*15*)	**Current Stream (Window) parameter block:**	
&B726	&B285	1	cursor y position (line)	with respect to the whole screen (starting from 0)
&B727	&B286	1	cursor x position (column)	with respect to the whole screen (starting from 0)
&B728	&B287	1		
&B729	&B288	1	window top line (y)	with respect to the whole screen (starting from 0)
&B72A	&B289	1	window left column (x)	with respect to the whole screen (starting from 0)
&B72B	&B28A	1	window bottom line (y)	with respect to the whole screen (starting from 0)
&B72C	&B28B	1	window right column (x)	with respect to the whole screen (starting from 0)
&B72D	&B28C	1	scroll count	
&B72E	&B28D	1	cursor flag (&01=disable; &02=off; &FD=on; &FE=enable)	
	&B28E	1		
&B72F	&B28F	1	current PEN number (encoded, not its INK number)	
&B730	&B290	1	current PAPER number (encoded, not its INK number)	
&B731	&B291	2	address of text background routine: opaque=&1392 – *&1391*; transparent=&13A0 – *&139F*	
&B733	&B293	1	graphics character writing flag (0=off; <>0=on)	
&B734	&B294	1	ASCII number of the first character in User Defined Graphic (UDG) matrix table	
&B735	&B295	1	UDG matrix table flag (&00=non-existent; &FF=present)	
&B736	&B296	2	address of UDG matrix table	
&B738	&B298	2		
&B758	&B2B8	1		
&B759	&B2B9	1		

6128	464	Size	Comments		
&B763	&B2C3	32*3	**Control Code handling routine table** — each code's entry comprises: (&B8A2 – *&B175*)		
			byte 1: +0 to +9=number of parameters; +&80=re-run routine at a System Reset		
			bytes 2 and 3: address of the control code's handling routine		
&B763	&B2C3	3	ASC 0:	&80,&1513(*&14E2*):	NUL
&B766	&B2C6	3	ASC 1:	&81,&1335(*&1334*):	Print control code chararacter [,char]
&B769	&B2C9	3	ASC 2:	&80,&1297(*&129A*):	Disable cursor
&B76C	&B2CC	3	ASC 3:	&80,&1286(*&1289*):	Enable cursor
&B76F	&B2CF	3	ASC 4:	&81,&0AE9(*&0ACA*):	Set mode [,mode]
&B772	&B2D2	3	ASC 5:	&81,&1940(*&1945*):	Print character using graphics mode [,char]
&B775	&B2D5	3	ASC 6:	&00,&1459(*&1451*):	Enable VDU
&B778	&B2D8	3	ASC 7:	&80,&14E1(*&14D8*):	Beep
&B77B	&B2DB	3	ASC 8:	&80,&1519(*&150A*):	Back-space
&B77E	&B2DE	3	ASC 9:	&80,&151E(*&150F*):	Step-right
&B781	&B2E1	3	ASC 10:	&80,&1523(*&1514*):	Line feed
&B784	&B2E4	3	ASC 11:	&80,&1528(*&1519*):	Previous line
&B787	&B2E7	3	ASC 12:	&80,&154F(*&1540*):	Clear window and locate the text cursor at position 1,1
&B78A	&B2EA	3	ASC 13:	&80,&153F(*&1530*):	RETURN
&B78D	&B2ED	3	ASC 14:	&81,&12AB(*&12AE*):	Set paper [,pen]
&B790	&B2F0	3	ASC 15:	&81,&12A6(*&12A9*):	Set pen [,pen]
&B793	&B2F3	3	ASC 16:	&80,&155E(*&154F*):	Delete the character at the cursor position

<> means 'not the value or bit which follows' '*' means this address applies to all machines with a disc drive fitted
b0 means bit 0, b1 means bit 1, etc HB means 'the most significant byte' and LB means 'the least significant byte'
addresses on the right are of System Variables that hold the address of the byte being explained (for the 464 they are in italics)

6128	464	Size	Comments on the memory locations		
&B796	&B2F6	3	ASC 17:	&80,&1599(*&158E*):	Clear the line up to the current cursor position
&B799	&B2F9	3	ASC 18:	&80,&158F(*&1584*):	Clear from the cursor position to the end of the line
&B79C	&B2FC	3	ASC 19:	&80,&1578(*&156D*):	Clear from start of the window to the cursor position
&B79F	&B2FF	3	ASC 20:	&80,&1565(*&1556*):	Clear from the cursor position to the end of a window
&B7A2	&B302	3	ASC 21:	&80,&1452(*&144B*):	Disable VDU
&B7A5	&B305	3	ASC 22:	&81,&14EC(*&14E3*):	Set text write mode [,mode]
&B7A8	&B308	3	ASC 23:	&81,&0C55(*&0C49*):	Set graphics draw mode [,mode]
&B7AB	&B30B	3	ASC 24:	&80,&12C6(*&12C9*):	Exchange pen and paper
&B7AE	&B30E	3	ASC 25:	&89,&150D(*&1504*):	Define user defined character [,char,8 rows of char]
&B7B1	&B311	3	ASC 26:	&84,&1501(*&15F8*):	Define window [,left,right,top,bottom]
&B7B4	&B314	3	ASC 27:	&00,&14EB(*&14E2*):	ESC (=user)
&B7B7	&B317	3	ASC 28:	&83,&14F1(*&14E8*):	Set the pen inks [,pen,ink 1,ink 2]
&B7BA	&B31A	3	ASC 29:	&82,&14FA(*&14F1*):	Set border colours [,ink 1,ink 2]
&B7BD	&B31D	3	ASC 30:	&80,&1539(*&152A*):	Locate the text cursor at position 1,1
&B7C0	&B320	3	ASC 31:	&82,&1547(*&1538*):	Locate the text cursor at [,column,line]

&B7C3	&B1C8	1	MODE number
&B7C4	&B1C9	2	screen offset
&B7C6	&B1CB	1	screen base HB (LB taken as &00)
&B7C7	&B1CC	3	graphics VDU write mode indirection — JP &0C74 – *JP &0C6B*
	&B1CF	8	list of bytes having only one bit set, from b7 down to b0
&B7D2	&B1D7	1	first flash period (SPEED INK - default &0A @ 0.02 seconds)
&B7D3	&B1D8	1	second flash period (SPEED INK - default &0A @ 0.02 seconds)

&B7D4	&B1D9	1+16*1	**Border and Pens' First Inks** (as hardware numbers):		
&B7D4	&B1D9	1	hw &04 = sw 1	(blue)	border
&B7D5	&B1DA	1	hw &04 = sw 1	(blue)	pen 0
&B7D6	&B1DB	1	hw &0A = sw 24	(bright yellow)	pen 1
&B7D7	&B1DC	1	hw &13 = sw 20	(bright cyan)	pen 2
&B7D8	&B1DD	1	hw &0C = sw 6	(bright red)	pen 3
&B7D9	&B1DE	1	hw &0B = sw 26	(bright white)	pen 4
&B7DA	&B1DF	1	hw &14 = sw 0	(black)	pen 5
&B7DB	&B1E0	1	hw &15 = sw 2	(bright blue)	pen 6
&B7DC	&B1E1	1	hw &0D = sw 8	(bright magenta)	pen 7
&B7DD	&B1E2	1	hw &06 = sw 10	(cyan)	pen 8
&B7DE	&B1E3	1	hw &1E = sw 12	(yellow)	pen 9
&B7DF	&B1E4	1	hw &1F = sw 14	(pale blue)	pen 10
&B7E0	&B1E5	1	hw &07 = sw 16	(pink)	pen 11
&B7E1	&B1E6	1	hw &12 = sw 18	(bright green)	pen 12
&B7E2	&B1E7	1	hw &19 = sw 22	(pale green)	pen 13
&B7E3	&B1E8	1	hw &04 = sw 1	(blue)	pen 14
&B7E4	&B1E9	1	hw &17 = sw 11	(sky blue)	pen 15

&B7E5	&B1EA	1+16*1	**Border and Pens' Second Inks** (as hardware numbers):		
&B7E5	&B1EA	1	hw &04 = sw 1	(blue)	border
&B7E6	&B1EB	1	hw &04 = sw 1	(blue)	pen 0
&B7E7	&B1EC	1	hw &0A = sw 24	(bright yellow)	pen 1
&B7E8	&B1ED	1	hw &13 = sw 20	(bright cyan)	pen 2
&B7E9	&B1EE	1	hw &0C = sw 6	(bright red)	pen 3
&B7EA	&B1EF	1	hw &0B = sw 26	(bright white)	pen 4
&B7EB	&B1F0	1	hw &14 = sw 0	(black)	pen 5
&B7EC	&B1F1	1	hw &15 = sw 2	(bright blue)	pen 6
&B7ED	&B1F2	1	hw &0D = sw 8	(bright magenta)	pen 7
&B7EE	&B1F3	1	hw &06 = sw 10	(cyan)	pen 8
&B7EF	&B1F4	1	hw &1E = sw 12	(yellow)	pen 9

6128	464	Size	Comments on the memory locations			
&B7F0	&B1F5	1	hw &1F	= sw 14	(pale blue)	pen 10
&B7F1	&B1F6	1	hw &07	= sw 16	(pink)	pen 11
&B7F2	&B1F7	1	hw &12	= sw 18	(bright green)	pen 12
&B7F3	&B1F8	1	hw &19	= sw 22	(pale green)	pen 13
&B7F4	&B1F9	1	hw &0A	= sw 24	(bright yellow)	pen 14
&B7F5	&B1FA	1	hw &07	= sw 16	(pink)	pen 15

&B7F6	&B1FB	1	
&B7F7	&B1FC	1	
&B7F8	&B1FD	1	
&B7F9	&B1FE	2	(&B8B9 – *&B18C*)
&B7FB	&B200	2	(&B82F – *&B102*)
&B7FD			
&B802		1+1	
&B804		1	number of entries in the Printer Translation table (normally 10)
&B805		20*2	**Printer Translation Table;** each entry comprises: byte 1: screen code byte 2: printer code
&B805		2	screen &A0 printer &5E (acute accent)
&B807		2	screen &A1 printer &5C (\)
&B809		2	screen &A2 printer &7B ({)
&B80B		2	screen &A3 printer &23 (#)
&B80D		2	screen &A6 printer &40 (@)
&B80F		2	screen &AB printer &7C (|)
&B811		2	screen &AC printer &7D (})
&B813		2	screen &AD printer &7E (~)
&B815		2	screen &AE printer &5D (])
&B817		2	screen &AF printer &5E ([)
&B819		20	room for ten more translations

&B82D	&B100	1	
&B82E	&B101	1	
&B82F	&B102	2	
&B831	&B104	1	
&B832	&B105	2	temporary store for stack pointer (SP) during interrupt handling
&B834	&B107	&70	temporary machine stack (from &B8B3 – *&B186* downwards) during interrupt handling
&B8B4	&B187	4	TIME (stored with the LB first — four bytes give >166 days; three bytes give >15 hours)
&B8B8	&B18B	1	
&B8B9	&B18C	2	
&B8BB	&B18E	2	
&B8BD	&B190	2	address of the first ticker block in chain (if any)
&B8BF	&B192	1	KB scan flag (&00=scan not needed; &01=scan needed)
&B8C0	&B193	2	address of the first event block in chain (if any)
&B8C2	&B195	1	
&B8C3	&B196	&10	buffer for last RSX or RSX command name (last character has bit 7 set)
&B8D3	&B1A6	2	address of first ROM or RSX chaining block in chain (eg &ACFC – *&ABFC*)
&B8D5		1	RAM bank number
&B8D6	&B1A8	1	Upper ROM status (eg select number)
&B8D7	&B1A9	2	entry point of foreground ROM in use (eg &C006 for BASIC ROM)
&B8D9	&B1AB	1	foreground ROM select address (0 for the BASIC ROM)

<> means 'not the value or bit which follows' '*' means this address applies to all machines with a disc drive fitted
b0 means bit 0, b1 means bit 1, etc HB means 'the most significant byte' and LB means 'the least significant byte'
addresses on the right are of System Variables that hold the address of the byte being explained (for the 464 they are in italics)

6128	464	Size	Comments on the memory locations
&B8DA		16*2	ROM entry IY value (ie address table) — the 6128 has ROMs numbered from 0 to 15:
	&B1AC	*7*2*	*ROM entry IY value (ie the address table) — the 464 has ROMs 1 to 7 only:*
&B8DA		2	ROM 0 IY (not for the 464)
&B8DC	&B1AC	2	ROM 1 IY
&B8DE	&B1AE	2	ROM 2 IY
&B8E0	&B1B0	2	ROM 3 IY
&B8E2	&B1B2	2	ROM 4 IY
&B8E4	&B1B4	2	ROM 5 IY
&B8E6	&B1B6	2	ROM 6 IY
&B8E8	&B1B8	2	ROM 7 IY (usually &A700 for AMSDOS/CPM ROM)
&B8EA		2	ROM 8 IY (not 464)
&B8EC		2	ROM 9 IY (not 464)
&B8EE		2	ROM 10 IY (not 464)
&B8F0		2	ROM 11 IY (not 464)
&B8F2		2	ROM 12 IY (not 464)
&B8F4		2	ROM 13 IY (not 464)
&B8F6		2	ROM 14 IY (not 464)
&B8F8		2	ROM 15 IY (not 464)
&B8FA			(6 bytes of &FF)
	&B1BA		*(14 bytes of &00)*

&B900	&B900	12*3	**High Kernel Jumpblock** — *on the 464 this block is (11*3) bytes in size*
&B924	&B921	&1C0	routines used by the High Kernel Jumpblock — *on the 464 this is &1C8 bytes in size*
&BAE4	&BAE9		&1C(*&17*) bytes of &FF
&BB00	&BB00	26*3	**Key Manager Jumpblock**
&BB4E	&BB4E	36*3	**Text VDU Jumpblock**
&BBBA	&BBBA	23*3	**Graphics VDU Jumpblock**
&BBFF	&BBFF	34*3	**Screen Pack Jumpblock**
&BC65	&BC65	22*3	**Cassette (and Disc if fitted) Manager Jumpblock**
&BCA7	&BCA7	11*3	**Sound Manager Jumpblock**
&BCC8	&BCC8	25*3	**Kernel Jumpblock**
&BD13	&BD13	26*3	**Machine Pack Jumpblock** — *on the 464 this block is (14*3) bytes in size*
&BD61	&BD3D	32*3	**Maths Jumpblock** — *on the 464 this block is (48*3) bytes in size*
&BDCD	&BDCD	14*3	**Firmware Indirections** — *on the 464 this block is (13*3) bytes in size*
&BDF7	&BDF4		(&09(*&0C*) bytes of &00 — the lower limit of Machine Stack if no Disc Drive
&BE00	&BE00		(&40 bytes of &FF)

&BE40	&BE40	&4x	used by the AMSDOS ROM if a disc drive is fitted (otherwise &4x bytes of &FF)
&BE40	&BE40	2	(address &A910)
&BE42	&BE42	2	address of drive A XDPB
&BE44	&BE44	9	**Disc Set Up timing block:**
&BE44	&BE44	2	motor on period (default &0032; fastest &0023 @ 20mS)
&BE46	&BE46	2	motor off period (default &00FA; fastest &00C8 @ 20mS)
&BE48	&BE48	1	write current off period (default &AF @ 10uS)
&BE49	&BE49	1	head settle time (default &0F @ 1mS)
&BE4A	&BE4A	1	step rate period (default &0C; fastest &0A @ 1mS)
&BE4B	&BE4B	1	head unload delay (default &01)
&BE4C	&BE4C	1	b0=non DMA mode; b1 to b7=head load delay (default &03)
&BE4D	&BE4D	2	
&BE4F	&BE4F	4	**Drive Header Information Block:**
&BE4F	&BE4F	1	last track used
&BE50	&BE50	1	head number (&00)
&BE51	&BE51	1	last sector used
&BE52	&BE52	1	log_2(sector size)–7

6128	464	Size	Comments on the memory locations
&BE53	&BE53	1	
&BE54	&BE54	1	
&BE55	&BE55	1	
&BE56	&BE56	1	
&BE58	&BE58	1	
&BE59	&BE59	1	
&BE5D	&BE5D	1	
&BE5E	&BE5E	1	
&BE5F	&BE5F	1	disc motor flag (&00=off; &01=on — strangely reversed)
&BE60	&BE60	2	address of buffer for directory entries block (&A930)
&BE62	&BE62	2	as &BE76 (ie &A9B0)
&BE64	&BE64	2	
&BE66	&BE66	1	disc retries (default &10)
&BE67	&BE67	&11	**AMSDOS Ticker and Event Block:**
&BE67	BE67	2	ticker chaining address
&BE69	BE69	2	tick count
&BE6B	BE6B	2	recharge count
&BE6D	BE6D	2	event chaining address
&BE6F	BE6F	1	count
&BE70	BE70	1	class (asynchronous event)
&BE71	BE71	2	ROM routine address (&C9D6)
&BE73	BE73	1	ROM select number (&07 ie the AMSDOS/CPM ROM)
&BE74	BE74	1	last sector number used
&BE75	BE75	1	
&BE76	BE76	2	address of 1/2K buffer, or of header info block (for WRITE SECTOR etc)
&BE78	&BE78	1	disc error message flag (&00=on; &FF=off — reversed again)
&BE7D	&BE7D	2	address of AMSDOS reserved area (&A700)
&BE7F	&BE7F	x	area used by AMSDOS to copy routines into RAM for running
&BE80	&BE80		(&80 bytes of &FF) limit of machine stack if disc drive fitted
&BF00	&BF00		(&xy bytes of &00)
&BFxy			machine stack (in theory this stack could extend down much further)
&BFFF	&BFFF		upper limit of machine stack

The area from &C000 to &FFFF is taken up by the screen memory — the layout of which is illustrated on the next page. Printed below are diagrams which show how the CPC uses the bytes of screen memory in the different MODEs. For each byte:

in MODE 2 (where there are two colours only, so each pixel needs only one bit — either on or off)

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
p0	p1	p2	p3	p4	p5	p6	p7	(the pixels are arranged with p0 being the leftmost one, etc)

in MODE 1 (where four colours are available and so two bits are needed for each pixel — 1 byte represents 4 pixels)

bit 7	bit 6	bi 5	bit 4	bit 3	bit 2	bit 1	bit 0	
p0(1)	p1(1)	p2(1)	p3(1)	p0(0)	p1(0)	p2(0)	p3(0)	(each pixel is twice as wide as in MODE 2)

in MODE 0 (where sixteen colours are possible and four bits are needed for each pixel — 1 byte represents 2 pixels)

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	
p0(0)	p1(0)	p0(2)	p1(2)	p0(1)	p1(1)	p0(3)	p1(3)	(each pixel is four times as wide as in MODE 2)

NB: the numbers in brackets show which bit of the pixel's pen number the screen byte bit refers to. For example in MODE 1, the 4 most significant bits of the byte hold bit 1 of the pixel's pen value and the 4 least signifcant bits hold bit 0 of the pen value.

<> means 'not the value or bit which follows' '*' means this address applies to all machines with a disc drive fitted
b0 means bit 0, b1 means bit 1, etc HB means 'the most significant byte' and LB means 'the least significant byte'
addresses on the right are of System Variables that hold the address of the byte being explained (for the 464 they are in italics)

6128	464	Size	Comments on the memory locations
&C000	&C000	&4000	normal (upper) screen area. The alternative (lower) screen area is from &4000 to &7FFF. The addresses of the starts of lines and rows in the normal screen area after a MODE instruction are:

LINE	ROW 0	ROW 1	ROW 2	ROW 3	ROW 4	ROW 5	ROW 6	ROW 7
1	C000	C800	D000	D800	E000	E800	F000	F800
2	C050	C850	D050	D850	E050	E850	F050	F850
3	C0A0	C8A0	D0A0	D8A0	E0A0	E8A0	F0A0	F8A0
4	C0F0	C8F0	D0F0	D8F0	E0F0	E8F0	F0F0	F8F0
5	C140	C940	D140	D940	E140	E940	F140	F940
6	C190	C990	D190	D990	E190	E990	F190	F990
7	C1E0	C9E0	D1E0	D9E0	E1E0	E9E0	F1E0	F9E0
8	C230	CA30	D230	DA30	E230	EA30	F230	FA30
9	C280	CA80	D280	DA80	E280	EA80	F280	FA80
10	C2D0	CAD0	D2D0	DAD0	E2D0	EAD0	F2D0	FAD0
11	C320	CB20	D320	DB20	E320	EB20	F320	FB20
12	C370	CB70	D370	DB70	E370	EB70	F370	FB70
13	C3C0	CBC0	D3C0	DBC0	E3C0	EBC0	F3C0	FBC0
14	C410	CC10	D410	DC10	E410	EC10	F410	FC10
15	C460	CC60	D460	DC60	E460	EC60	F460	FC60
16	C4B0	CCB0	D4B0	DCB0	E4B0	ECB0	F4B0	FCB0
17	C500	CD00	D500	DD00	E500	ED00	F500	FD00
18	C550	CD50	D550	DD50	E550	ED50	F550	FD50
19	C5A0	CDA0	D5A0	DDA0	E5A0	EDA0	F5A0	FDA0
20	C5F0	CDF0	D5F0	DDF0	E5F0	ED50	F550	FD50
21	C640	CE40	D640	DE40	E640	EE40	F640	FE40
22	C690	CE90	D690	DE90	E690	EE90	F690	FE90
23	C6E0	CEE0	D6E0	DEE0	E6E0	EEE0	F6E0	FEE0
24	C730	CF30	D730	DF30	E730	EF30	F730	FF30
25	C780	CF80	D780	DF80	E780	EF80	F780	FF80
spare start	C7D0	CFD0	D7D0	DFD0	E7D0	EFD0	F7D0	FFD0
spare end	C7FF	CFFF	D7FF	DFFF	E7FF	EFFF	F7FF	FFFF

Once the whole screen has been scrolled in any direction, the above table will become incorrect. On scrolling, all the above addresses will have an offset (MOD &800) added, derived as follows:

+&02 per scroll to the left (=2, 1 or 1/2 character in MODE 2, MODE 1 or MODE 0 respectively)
–&02 per scroll to the right (=2, 1 or 1/2 character in MODE 2, MODE 1 or MODE 0 respectively)
+&50 per scroll up one line
–&50 per scroll down one line

If scrolled far enough, a screen row may sit across the boundaries of the screen memory area, whose bottom end will then wrap around to join up with the top (ie byte &FFFF will be followed by byte &C000 assuming the normal screen area). If before scrolling however, a window had been set up smaller than the whole screen then the table will remain accurate despite any scrolling.

The 'spare' areas of screen memory are filled with bytes of the relevant PAPER value each time there is a full screen CLS, and are not really available for other uses. After scrolling the spare areas may be used as screen with other bytes becoming spare.

Please note that this section of the guide has been set out with all the addresses in the leftmost column in the correct order for the 6128. However, a Conversion Chart specifically for the 464 is printed in Appendix C.

The Firmware Jumpblocks

The Firmware Jumpblock is the recommended method of communicating with the routines in the lower ROM — it is used by BASIC, and it should also be used by other programs.

The reason for using the jumpblock is that the routines in the lower ROM are located at different positions on the different machines. The entries in the jumpblock, however, are all in the same place — the instructions in the jumpblock redirect the computer to the correct place in the lower ROM. Thus, providing a program uses the jumpblock, it should work on any CPC or Plus computer.

By altering the firmware jumpblock it is possible to make the computer run a different routine from normal. This could either be a different routine in the lower or upper ROM, or a routine written by the user — this is known as 'patching the jumpblock'. It is worth noting that because BASIC uses the firmware jumpblock quite heavily, it is possible to alter the effect of BASIC commands.

The following example will change the effect of calling SCR SET MODE (&BC0E) — instead of changing the mode, any calls to this location will print the letter 'A'. The first thing to do is to assemble the piece of code that will be used to print the letter — this is printed below and starts at &4000.

```
ORG &4000
LD A,65
CALL &BB5A
RET
```

The jumpblock entry for SCR SET MODE is now patched so that it re-routes all calls to &BC0E away from the lower ROM and to our custom routine at &4000. This is done by changing the bytes at &BC0E, &BC0F and &BC10 to &C3, &00, &40 respectively (ie JP &4000). Any calls to &BC0E or MODE commands will now print the letter A instead of changing mode.

The indirections jumpblock contains a small number of routines which are called by the rest of the firmware. By altering this jumpblock, it is possible to alter the way in which the firmware operates on a large scale — thus it is not always necessary to patch large numbers of entries in the firmware jumpblock.

There are two jumpblocks which are to do with the Kernel (ie the high and low Kernel jumpblocks). The high jumpblock allows ROM states and interrupts to be altered, and also controls the introduction of RSXs. The low jumpblock contains general routines and restart instructions which are used by the computer for its own purposes.

The CPC Firmware

The Low Kernel Jumpblock

000	&0000	RESET ENTRY (RST 0)
001	&0008	LOW JUMP (RST 1)
002	&000B	KL LOW PCHL
003	&000E	PCBC INSTRUCTION
004	&0010	SIDE CALL (RST 2)
005	&0013	KL SIDE PCHL
006	&0016	PCDE INSTRUCTION
007	&0018	FAR CALL (RST 3)
008	&001B	KL FAR PCHL
009	&001E	PCHL INSTRUCTION
010	&0020	RAM LAM (RST 4)
011	&0023	KL FAR CALL
012	&0028	FIRM JUMP (RST 5)
013	&0030	USER RESTART (RST 6)
014	&0038	INTERRUPT ENTRY (RST 7)
015	&003B	EXT INTERRUPT

The High Kernel Jumpblock

000	&B900	KL U ROM ENABLE
001	&B903	KL U ROM DISABLE
002	&B906	KL L ROM ENABLE
003	&B909	KL L ROM DISABLE
004	&B90C	KL ROM RESTORE
005	&B90F	KL ROM SELECT
006	&B912	KL CURR SELECTION
007	&B915	KL PROBE ROM
008	&B918	KL ROM DESELECT
009	&B91B	KL LDIR
010	&B91E	KL LDDR
011	&B921	KL POLL SYNCHRONOUS
014	&B92A	KL SCAN NEEDED

The Key Manager

000	&BB00	KM INITIALISE
001	&BB03	KM RESET
002	&BB06	KM WAIT CHAR
003	&BB09	KM READ CHAR
004	&BB0C	KM CHAR RETURN
005	&BB0F	KM SET EXPAND
006	&BB12	KM GET EXPAND
007	&BB15	KM EXP BUFFER
008	&BB18	KM WAIT KEY
009	&BB1B	KM READ KEY
010	&BB1E	KM TEST KEY
011	&BB21	KM GET STATE
012	&BB24	KM GET JOYSTICK
013	&BB27	KM SET TRANSLATE
014	&BB2A	KM GET TRANSLATE
015	&BB2D	KM SET SHIFT
016	&BB30	KM GET SHIFT
017	&BB33	KM SET CONTROL
018	&BB36	KM GET CONTROL
019	&BB39	KM SET REPEAT
020	&BB3C	KM GET REPEAT
021	&BB3F	KM SET DELAY
022	&BB42	KM GET DELAY
023	&BB45	KM ARM BREAK
024	&BB48	KM DISARM BREAK
025	&BB4B	KM BREAK EVENT

The Text VDU

026	&BB4E	TXT INITIALISE
027	&BB51	TXT RESET
028	&BB54	TXT VDU ENABLE
029	&BB57	TXT VDU DISABLE
030	&BB5A	TXT OUTPUT
031	&BB5D	TXT WR CHAR
032	&BB60	TXT RD CHAR
033	&BB63	TXT SET GRAPHIC
034	&BB66	TXT WIN ENABLE
035	&BB69	TXT GET WINDOW
036	&BB6C	TXT CLEAR WINDOW
037	&BB6F	TXT SET COLUMN
038	&BB72	TXT SET ROW
039	&BB75	TXT SET CURSOR
040	&BB78	TXT GET CURSOR
041	&BB7B	TXT CUR ENABLE
042	&BB7E	TXT CUR DISABLE
043	&BB81	TXT CUR ON
044	&BB84	TXT CUR OFF
045	&BB87	TXT INVALIDATE
046	&BB8A	TXT PLACE CURSOR
047	&BB8D	TXT REMOVE CURSOR
048	&BB90	TXT SET PEN
049	&BB93	TXT GET PEN
050	&BB96	TXT SET PAPER
051	&BB99	TXT GET PAPER
052	&BB9C	TXT INVERSE
053	&BB9F	TXT SET BACK
054	&BBA2	TXT GET BACK
055	&BBA5	TXT GET MATRIX
056	&BBA8	TXT SET MATRIX
057	&BBAB	TXT SET M TABLE
058	&BBAE	TXT GET M TABLE
059	&BBB1	TXT GET CONTROLS
060	&BBB4	TXT STR SELECT
061	&BBB7	TXT SWAP STREAMS

The Graphics VDU

062	&BBBA	GRA INITIALISE
063	&BBBD	GRA RESET
064	&BBC0	GRA MOVE ABSOLUTE
065	&BBC3	GRA MOVE RELATIVE
066	&BBC6	GRA ASK CURSOR
067	&BBC9	GRA SET ORIGIN
068	&BBCC	GRA GET ORIGIN
069	&BBCF	GRA WIN WIDTH
070	&BBD2	GRA WIN HEIGHT
071	&BBD5	GRA GET W WIDTH
072	&BBD8	GRA GET W HEIGHT
073	&BBDB	GRA CLEAR WINDOW
074	&BBDE	GRA SET PEN
075	&BBE1	GRA GET PEN
076	&BBE4	GRA SET PAPER
077	&BBE7	GRA GET PAPER
078	&BBEA	GRA PLOT ABSOLUTE
079	&BBED	GRA PLOT RELATIVE
080	&BBF0	GRA TEST ABSOLUTE
081	&BBF3	GRA TEST RELATIVE
082	&BBF6	GRA LINE ABSOLUTE
083	&BBF9	GRA LINE RELATIVE
084	&BBFC	GRA WR CHAR

The Screen Pack

085	&BBFF	SCR INITIALISE
086	&BC02	SCR RESET
087	&BC05	SCR SET OFFSET
088	&BC08	SCR SET BASE
089	&BC0B	SCR GET LOCATION
090	&BC0E	SCR SET MODE
091	&BC11	SCR GET MODE
092	&BC14	SCR CLEAR
093	&BC17	SCR CHAR LIMITS
094	&BC1A	SCR CHAR POSITION
095	&BC1D	SCR DOT POSITION
096	&BC20	SCR NEXT BYTE
097	&BC23	SCR PREV BYTE
098	&BC26	SCR NEXT LINE
099	&BC29	SCR PREV LINE
100	&BC2C	SCR INK ENCODE
101	&BC2F	SCR INK DECODE
102	&BC32	SCR SET INK
103	&BC35	SCR GET INK
104	&BC38	SCR SET BORDER

105	&BC3B	SCR GET BORDER
106	&BC3E	SCR SET FLASHING
107	&BC41	SCR GET FLASHING
108	&BC44	SCR FILL BOX
109	&BC47	SCR FLOOD BOX
110	&BC4A	SCR CHAR INVERT
111	&BC4D	SCR HW ROLL
112	&BC50	SCR SW ROLL
113	&BC53	SCR UNPACK
114	&BC56	SCR REPACK
115	&BC59	SCR ACCESS
116	&BC5C	SCR PIXELS
117	&BC5F	SCR HORIZONTAL
118	&BC62	SCR VERTICAL
119	&BC65	CAS INITIALISE
120	&BC68	CAS SET SPEED
121	&BC6B	CAS NOISY
122	&BC6E	CAS START MOTOR
123	&BC71	CAS STOP MOTOR
124	&BC74	CAS RESTORE MOTOR
125	&BC77	CAS IN OPEN
126	&BC7A	CAS IN CLOSE
127	&BC7D	CAS IN ABANDON
128	&BC80	CAS IN CHAR
129	&BC83	CAS IN DIRECT
130	&BC86	CAS RETURN
131	&BC89	CAS TEST EOF
132	&BC8C	CAS OUT OPEN
133	&BC8F	CAS OUT CLOSE
134	&BC92	CAS OUT ABANDON
135	&BC95	CAS OUT CHAR
136	&BC98	CAS OUT DIRECT
137	&BC9B	CAS CATALOG
138	&BC9E	CAS WRITE
139	&BCA1	CAS READ
140	&BCA4	CAS CHECK
141	&BCA7	SOUND RESET
142	&BCAA	SOUND QUEUE
143	&BCAD	SOUND CHECK
144	&BCB0	SOUND ARM EVENT
145	&BCB3	SOUND RELEASE
146	&BCB6	SOUND HOLD
147	&BCB9	SOUND CONTINUE
148	&BCBC	SOUND AMPL ENVELOPE
149	&BCBF	SOUND TONE ENVELOPE
150	&BCC2	SOUND A ADDRESS
151	&BCC5	SOUND T ADDRESS
152	&BCC8	KL CHOKE OFF
153	&BCCB	KL ROM WALK
154	&BCCE	KL INIT BACK
155	&BCD1	KL LOG EXT
156	&BCD4	KL FIND COMMAND
157	&BCD7	KL NEW FRAME FLY
158	&BCDA	KL ADD FRAME FLY
159	&BCDD	KL DEL FRAME FLY
160	&BCE0	KL NEW FAST TICKER
161	&BCE3	KL ADD FAST TICKER
162	&BCE6	KL DEL FAST TICKER
163	&BCE9	KL ADD TICKER
164	&BCEC	KL DEL TICKER
165	&BCEF	KL INIT EVENT
166	&BCF2	KL EVENT
167	&BCF5	KL SYNC REST
168	&BCF8	KL DEL SYNCHRONOUS
169	&BCFB	KL NEXT SYNC
170	&BCFE	KL DO SYNC
171	&BD01	KL DONE SYNC
172	&BD04	KL EVENT DISABLE
173	&BD07	KL EVENT DISABLE
174	&BD0A	KL DISARM EVENT
175	&BD0D	KL TIME PLEASE
176	&BD10	KL TIME SET
177	&BD13	MC BOOT PROGRAM
178	&BD16	MC START PROGRAM
179	&BD19	MC WAIT FLYBACK
180	&BD1C	MC SET MODE
181	&BD1F	MC SCREEN OFFSET
182	&BD22	MC CLEAR INKS
183	&BD25	MC SET INKS
184	&BD28	MC RESET PRINTER
185	&BD2B	MC PRINT CHAR
186	&BD2E	MC BUSY PRINTER
187	&BD31	MC SEND PRINTER
188	&BD34	MC SOUND REGISTER
189	&BD37	JUMP RESTORE

664 or 6128 only

190	&BD3A	KM SET LOCKS
191	&BD3D	KM FLUSH
192	&BD40	TXT ASK STATE
193	&BD43	GRA DEFAULT
194	&BD46	GRA SET BACK
195	&BD49	GRA SET FIRST
196	&BD4C	GRA SET LINE MASK
197	&BD4F	GRA FROM USER
198	&BD52	GRA FILL
199	&BD55	SCR SET POSITION
200	&BD58	MC PRINT TRANSLATION
201	&BD5B	KL BANK SWITCH

Firmware Indirections

000	&BDCD	TXT DRAW CURSOR
001	&BDD0	TXT UNDRAW CURSOR
002	&BDD3	TXT WRITE CHAR
003	&BDD6	TXT UNWRITE
004	&BDD9	TXT OUT ACTION
005	&BDDC	GRA PLOT
006	&BDDF	GRA TEST
007	&BDE2	GRA LINE
008	&BDE5	SCR READ
009	&BDE8	SCR WRITE
010	&BDEB	SCR MODE CLEAR
011	&BDEE	KM TEST BREAK
012	&BDF1	MC WAIT PRINTER
013	&BDF4	KM SCAN KEYS

The Firmware in Detail

Low Kernel Jumpblock

000 &0000 RESET ENTRY (RST 0)
Action: Resets the computer as if it has just been switched on
Entry: No entry conditions
Exit: This routine is never returned from
Notes: After initialisation of the hardware and firmware, control is handed over to ROM 0 (usually BASIC)

001 &0008 LOW JUMP (RST 1)
Action: Jumps to a routine in either the lower ROM or low RAM
Entry: No entry conditions — all the registers are passed to the destination routine unchanged
Exit: The registers are as set by the routine in the lower ROM or RAM or are returned unaltered
Notes: The RST 1 instruction is followed by a two byte low address, which is defined as follows:
- if bit 15 is set, then the upper ROM is disabled
- if bit 14 is set, then the lower ROM is disabled
- bits 13 to 0 contain the address of the routine to jump to

This command is used by the majority of entries in the main firmware jumpblock

002 &000B KL LOW PCHL
Action: Jumps to a routine in either the lower ROM or low RAM
Entry: HL contains the low address — all the registers are passed to the destination routine unchanged
Exit: The registers are as set by the routine in the lower ROM or RAM or are returned unaltered
Notes: The two byte low address in the HL register pair is defined as follows:
- if bit 15 is set, then the upper ROM is disabled
- if bit 14 is set, then the lower ROM is disabled
- bits 13 to 0 contain the address of the routine to jump to

003 &000E PCBC INSTRUCTION
Action: Jumps to the specified address
Entry: BC contains the address to jump to — all the registers are passed to the destination routine unaltered
Exit: The registers are as set by the destination routine or are returned unchanged

004 &0010 SIDE CALL (RST 2)
Action: Calls a routine in ROM, in a group of upto four foreground ROMs
Entry: No entry conditions — all the registers apart from IY are passed to the destination routine unaltered
Exit: IY is corrupt, and the other registers are as set by the destination routine or are returned unchanged
Notes: The RST 2 instruction is followed by a two byte side address, which is defined as follows:
- bits 14 and 15 give a number between 0 and 3, which is added to the main foreground ROM select address — this is then used as the ROM select address
- bits 0 to 13 contain the address to which is added &C000 — this gives the address of the routine to be called

005 &0013 KL SIDE PCHL
Action: Calls a routine in another ROM
Entry: HL contains the side address — all the registers apart from IY are passed to the destination routine unaltered
Exit: IY is corrupt, and the other registers are as set by the destination routine or are returned unchanged
Notes: The two byte side address is defined as follows:
- bits 14 and 15 give a number between 0 and 3, which is added to the main foreground ROM select address — this is then used as the ROM select address
- bits 0 to 13 contain the address to which is added &C000 — this gives the address of the routine to be called

006 &0016 PCDE INSTRUCTION
Action: Jumps to the specified address
Entry: DE contains the address to jump to — all the registers are passed to the destination routine unaltered
Exit: The registers are as set by the destination routine or are returned unchanged

007 &0018 FAR CALL (RST 3)
Action: Calls a routine anywhere in RAM or ROM
Entry: No entry conditions — all the registers apart from IY are passed to the destination routine unaltered
Exit: IY is preserved, and the other registers are as set by the destination routine or are returned unchanged
Notes: The RST 3 instruction is followed by a two byte in-line address. At this address, there is a three byte far address, which is defined as follows:
bytes 0 and 1 give the address of the routine to be called
byte 2 is the ROM select byte which has values as follows:
&00 to &FB — select the given upper ROM, enable the upper ROM and disable the lower ROM
&FC — no change to the ROM selection, enable the upper and lower ROMs
&FD — no change to the ROM selection, enable the upper ROM and disable the lower ROM
&FE — no change to the ROM selection, disable the upper ROM and enable the lower ROM
&FF — no change to the ROM selection, disable the upper and lower ROMs
When it is returned from, the ROM selection and state are restored to their settings before the RST 3 command

008 &001B KL FAR PCHL
Action: Calls a routine, given by the far address in HL & C, anywhere in RAM or ROM
Entry: HL holds the address of the routine to be called, and C holds the ROM select byte — all the registers apart from IY are passed to the destination routine unaltered
Exit: IY is preserved, and the other registers are as set by the destination routine or are returned unchanged
Notes: See FAR CALL above for more details on the ROM select byte

009 &001E PCHL INSTRUCTION
Action: Jumps to the specified address
Entry: HL contains the address to jump to — all the registers are passed to the destination routine unaltered
Exit: The registers are as set by the destination routine or are returned unchanged

010 &0020 RAM LAM
Action: Puts the contents of a RAM memory location into the A register
Entry: HL contains the address of the memory location
Exit: A holds the contents of the memory location, and all other registers are preserved
Notes: This routine always reads from RAM, even if the upper or lower ROM is enabled

011 &0023 KL FAR CALL
Action: Calls a routine anywhere in RAM or ROM
Entry: HL holds the address of the three byte far address that is to be used — all the registers apart from IY are passed to the destination routine unaltered
Exit: IY is preserved, and the other registers are as set by the destination routine or are returned unchanged
Notes: See FAR CALL above for more details on the three byte far address

012 &0028 FIRM JUMP (RST 5)
Action: Jumps to a routine in either the lower ROM or the central 32K of RAM
Entry: No entry conditions — all the registers are passed to the destination routine unchanged
Exit: The registers are as set by the routine in the lower ROM or RAM or are returned unaltered
Notes: The RST 5 instruction is followed by a two byte address, which is the address to jump to; before the jump is made, the lower ROM is enabled, and is disabled when the destination routine is returned from

013 &0030 USER RESTART (RST 6)
Action: This is an RST instruction that may be set aside by the user for any purpose
Entry: Defined by the user
Exit: Defined by the user
Notes: The bytes from &0030 to &0037 are available for the user to put their own code in if they wish

014 &0038 INTERRUPT ENTRY (RST 7)
Action: Deals with normal interrupts
Entry: No entry conditions
Exit: All registers are preserved
Notes: The RST 7 instruction must not be used by the user; any external interrupts that are generated by hardware on the expansion port will be dealt with by the EXT INTERRUPT routine (see over)

015 &003B EXT INTERRUPT
Action: This area is set aside for dealing with external interrupts that are generated by any extra hardware
Entry: No entry conditions
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved
Notes: If any external hardware is going to generate interrupts, then the user must patch the area from &003B to &003F so that the computer can deal with the external interrupt; when an external interrupt occurs, the lower ROM is disabled and the code at &003B is called; the default external interrupt routine at &003B simply returns, and this will cause the computer to hang because the interrupt will continue to exist

High Kernel Jumpblock

000 &B900 KL U ROM ENABLE
Action: Enables the current upper ROM
Entry: No entry conditions
Exit: A contains the previous state of the ROM, the flags are corrupt, and all other registers are preserved
Notes: After this routine has been called, all reading from addresses between &C000 and &FFFF refers to the upper ROM, and not the top 16K of RAM which is usually the screen memory; any writing to these addresses still affects the RAM as, by its nature, ROM cannot be written to

001 &B903 KL U ROM DISABLE
Action: Disables the upper ROM
Entry: No entry conditions
Exit: A contains the previous state of the ROM, the flags are corrupt, and all other registers are preserved
Notes: After this routine has been called, all reading from addresses between &C000 and &FFFF refers to the top 16K of RAM which is usually the screen memory

002 &B906 KL L ROM ENABLE
Action: Enables the lower ROM
Entry: No entry conditions
Exit: A contains the previous state of the ROM, the flags are corrupt, and all other registers are preserved
Notes: After this routine has been called, all reading from addresses between &0000 and &4000 refers to the lower ROM, and not the bottom 16K of RAM; any writing to these addresses still affects the RAM as a ROM cannot be written to; the lower ROM is automatically enabled when a firmware routine is called, and is then disabled when the routine returns

003 &B909 KL L ROM DISABLE
Action: Disables the lower ROM
Entry: No entry conditions
Exit: A contains the previous state of the ROM, the flags are corrupt, and all other registers are preserved
Notes: After this routine has been called, all reading from addresses between &0000 and &4000 refers to the bottom 16K of RAM; the lower ROM is automatically enabled when a firmware routine is called, and is then disabled when the routine returns

004 &B90C KL ROM RESTORE
Action: Restores the ROM to its previous state
Entry: A contains the previous state of the ROM
Exit: AF is corrupt, and all other registers are preserved
Notes: The previous four routines all return values in the A register which are suitable for use by KL ROM RESTORE

005 &B90F KL ROM SELECT
Action: Selects an upper ROM and also enables it
Entry: C contains the ROM select address of the required ROM
Exit: C contains the ROM select address of the previous ROM, and B contains the state of the previous ROM

006 &B912 KL CURR SELECTION
Action: Gets the ROM select address of the current ROM
Entry: No entry conditions
Exit: A contains the ROM select address of the current ROM, and all other registers are preserved

007 &B915 KL PROBE ROM

Action: Gets the class and version of a specified ROM
Entry: C contains the ROM select address of the required ROM
Exit: A contains the class of the ROM, H holds the version number, L holds the mark number, B and the flags are corrupt, and all other registers are preserved
Notes: The ROM class may be one of the following:
&00 — a foreground ROM
&01 — a background ROM
&02 — an extension foreground ROM
&80 — the built in ROM (ie the BASIC ROM)

008 &B918 KL ROM DESELECT

Action: Selects the previous upper ROM and sets its state
Entry: C contains the ROM select address of the ROM to be reselected, and B contains the state of the required ROM
Exit: C contains the ROM select address of the current ROM, B is corrupt, and all others are preserved
Notes: This routine reverses the action of KL ROM SELECT, and uses the values that it returns in B and C

009 &B91B KL LDIR

Action: Switches off the upper and lower ROMs, and moves a block of memory
Entry: As for a standard LDIR instruction (ie DE holds the destination location, HL points to the first byte to be moved, and BC holds the length of the block to be moved)
Exit: F, BC, DE and HL are set as for a normal LDIR instruction, and all other registers are preserved

010 &B91E KL LDDR

Action: Switches off the upper and lower ROMs, and moves a block of memory
Entry: As for a standard LDDR instruction (ie DE holds the first destination location, HL points to the highest byte in memory to be moved, and BC holds the number of bytes to be moved)
Exit: F, BC, DE and HL are set as for a normal LDDR instruction, and all other registers are preserved

011 &B921 KL POLL SYNCHRONOUS

Action: Tests whether an event with a higher priority than the current event is waiting to be dealt with
Entry: No entry conditions
Exit: If there is a higher priority event, then Carry is false;
if there is no higher priority event, then Carry is true;
in either case, A and the other flags are corrupt, and all other registers are preserved

014 &B92A KL SCAN NEEDED

Action: Ensures that the keyboard is scanned when the next ticker interrupt occurs
Entry: No entry conditions
Exit: AF and HL are corrupt, and all other registers are preserved
Notes: This routine is useful for scanning the keyboard when the interrupts are disabled and normal key scanning is not occurring

The Key Manager

000 &BB00 KM INITIALISE

Action: Initialises the Key Manager and sets up everything as it is when the computer is first switched on; the key buffer is emptied, Shift and Caps lock are turned off and all the expansion and translation tables are reset to normal; also see the routine KM RESET below
Entry: No entry conditions
Exit: AF, BC, DE and HL corrupt, and all other registers are preserved

001 &BB03 KM RESET

Action: Resets the Key Manager; the key buffer is emptied and all current keys/characters are ignored
Entry: No entry conditions
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved
Notes: See also KM INITIALISE above; on the 664 or 6128, the key buffer can also be cleared separately by calling the KM FLUSH routine

002 &BB06 KM WAIT CHAR
Action: Waits for the next character from the keyboard buffer
Entry: No entry conditions
Exit: Carry is true, A holds the character value, the other flags are corrupt, and all other registers are preserved

003 &BB09 KM READ CHAR
Action: Tests to see if a character is available from the keyboard buffer, but doesn't wait for one to become available
Entry: No entry conditions
Exit: If a character was available, then Carry is true, and A contains the character;
otherwise Carry is false, and A is corrupt;
in both cases, the other registers are preserved

004 &BB0C KM CHAR RETURN
Action: Saves a character for the next use of KM WAIT CHAR or KM READ CHAR
Entry: A contains the ASCII code of the character to be put back
Exit: All registers are preserved

005 &BB0F KM SET EXPAND
Action: Assigns a string to a key code
Entry: B holds the key code; C holds the length of the string; HL contains the address of the string (must be in RAM)
Exit: If it is OK, then Carry is true;
otherwise Carry is false;
in either case, A, BC, DE and HL are corrupt, and all other registers are preserved

006 &BB12 KM GET EXPAND
Action: Reads a character from an expanded string of characters
Entry: A holds an expansion token (ie a key code) and L holds the character position number (starts from 0)
Exit: If it is OK, then Carry is true, and A holds the character;
otherwise Carry is false, and A is corrupt;
in either case, DE and flags are corrupt, and the other registers are preserved

007 &BB15 KM EXP BUFFER
Action: Sets aside a buffer area for character expansion strings
Entry: DE holds the address of the buffer and HL holds the length of the buffer
Exit: If it is OK, then Carry is true;
otherwise Carry is false;
in either case, A, BC, DE and HL are corrupt
Notes: The buffer must be in the central 32K of RAM and must be at least 49 bytes long

008 &BB18 KM WAIT KEY
Action: Waits for a key to be pressed — this routine does not expand any expansion tokens
Entry: No entry conditions
Exit: Carry is true, A holds the character or expansion token, and all other registers are preserved

009 &BB1B KM READ KEY
Action: Tests whether a key is available from the keyboard
Entry: No entry conditions
Exit: If a key is available, then Carry is true, and A contains the character;
otherwise Carry is false, and A is corrupt;
in either case, the other registers are preserved
Notes: Any expansion tokens are not expanded

010 &BB1E KM TEST KEY
Action: Tests if a particular key (or joystick direction or button) is pressed
Entry: A contains the key/joystick number
Exit: If the requested key is pressed, then Zero is false;
otherwise Zero is true;
for both, Carry is false, A and HL are corrupt, C holds the Shift and Control status, and others are preserved
Notes: After calling this, C will hold the state of shift and control — if bit 7 is set then Control was pressed, and if bit 5 is set then Shift was pressed

011 &BB21 KM GET STATE
Action: Gets the state of the Shift and Caps locks
Entry: No entry conditions
Exit: If L holds &FF then the shift lock is on, but if L holds &00 then the Shift lock is off; if H holds &FF then the caps lock is on, and if H holds &00 then the Caps lock is off; whatever the outcome, all the other registers are preserved

012 &BB24 KM GET JOYSTICK
Action: Reads the present state of any joysticks attached
Entry: No entry conditions
Exit: H and A contains the state of joystick 0, L holds that state of joystick 1, and all others are preserved
Notes: The joystick states are bit significant and are as follows:

Bit 0 — Up	Bit 1 — Down	Bit 2 — Left	Bit 3 — Right
Bit 4 — Fire 2	Bit 5 — Fire 1	Bit 6 — Spare	Bit 7 — Always zero

The bits are set when the corresponding buttons or directions are operated

013 &BB27 KM SET TRANSLATE
Action: Sets the token or character that is assigned to a key when neither Shift nor Control are pressed
Entry: A contains the key number and B contains the new token or character
Exit: AF and HL are corrupt, and all other registers are preserved
Notes: Special values for B are as follows:

&80 to &9F — these values correspond to the expansion tokens
&FD — this causes the caps lock to toggle on and off
&FE — this causes the shift lock to toggle on and off
&FF — causes this key to be ignored

014 &BB2A KM GET TRANSLATE
Action: Finds out what token or character will be assigned to a key when neither Shift nor Control are pressed
Entry: A contains the key number
Exit: A contains the token/character that is assigned, HL and flags are corrupt, and all others are preserved
Notes: See KM SET TRANSLATE for special values that can be returned

015 &BB2D KM SET SHIFT
Action: Sets the token or character that will be assigned to a key when Shift is pressed as well
Entry: A contains the key number and B contains the new token or character
Exit: AF and HL are corrupt, and all others are preserved
Notes: See KM SET TRANSLATE for special values that can be set

016 &BB30 KM GET SHIFT
Action: Finds out what token/character will be assigned to a key when Shift is pressed as well
Entry: A contains the key number
Exit: A contains the token/character that is assigned, HL and flags are corrupt, and all others are preserved
Notes: See KM SET TRANSLATE for special values that can be returned

017 &BB33 KM SET CONTROL
Action: Sets the token or character that will be assigned to a key when Control is pressed as well
Entry: A contains the key number and B contains the new token/character
Exit: AF and HL are corrupt, and all others are preserved
Notes: See KM SET TRANSLATE for special values that can be set

018 &BB36 KM GET CONTROL
Action: Finds out what token or character will be assigned to a key when Control is pressed as well
Entry: A contains the key number
Exit: A contains the token/character that is assigned, HL and flags are corrupt and all others are preserved
Notes: See KM SET TRANSLATE for special values that can be set

019 &BB39 KM SET REPEAT
Action: Sets whether a key may repeat or not
Entry: A contains the key number, B contains &00 if there is no repeat and &FF is it is to repeat
Exit: AF, BC and HL are corrupt, and all others are preserved

020 &BB3C KM GET REPEAT

Action: Finds out whether a key is set to repeat or not
Entry: A contains a key number
Exit: If the key repeats, then Zero is false;
if the key does not repeat, then Zero is true;
in either case, A, HL and flags are corrupt, Carry is false, and all other registers are preserved

021 &BB3F KM SET DELAY

Action: Sets the time that elapses before the first repeat, and also set the repeat speed
Entry: H contains the time before the first repeat, and L holds the time between repeats (repeat speed)
Exit: AF is corrupt, and all others are preserved
Notes: The values for the times are given in 1/50th seconds, and a value of 0 counts as 256

022 &BB42 KM GET DELAY

Action: Finds out the time that elapses before the first repeat and also the repeat speed
Entry: No entry conditions
Exit: H contains the time before the first repeat, and L holds the time between repeats, and all others are preserved

023 &BB45 KM ARM BREAK

Action: Arms the Break mechanism
Entry: DE holds the address of the Break handling routine, C holds the ROM select address for this routine
Exit: AF, BC, DE and HL are corrupt, and all the other registers are preserved

024 &BB48 KM DISARM BREAK

Action: Disables the Break mechanism
Entry: No entry conditions
Exit: AF and HL are corrupt, and all the other registers are preserved

025 &BB4B KM BREAK EVENT

Action: Generates a Break interrupt if a Break routine has been specified by KM ARM BREAK
Entry: No entry conditions
Exit: AF and HL are corrupt, and all other registers are preserved

The Text VDU

026 &BB4E TXT INITIALISE

Action: Initialise the text VDU to its settings when the computer is switched on, includes resetting all the text VDU indirections, selecting Stream 0, resetting the text paper to pen 0 and the text pen to pen 1, moving the cursor to the top left corner of the screen and setting the writing mode to be opaque
Entry: No entry conditions
Exit: AF, BC, DE and HL are corrupt, and all others are preserved

CHARACTER COORDINATES AND POSITIONS

The CPC handbook's and the original Firmware Manual ambiguously used the term 'row' to indicate both a character line, and any one of the rows of pixels which go to make up a displayed character. In this guide, the term 'line' is used to indicate a character line (as in everyday usage), and a 'row' to indicate any of the eight horizontal rows of pixels which make up a character line.

Character positions are expressed using three systems of coordinates:

- logical coordinates are those related to 1,1 at the top left of the screen; used in BASIC only to set a window's size, or in the Firmware
- physical coordinates again relate to the top left of the screen, but the character position in the top left is here expressed as being 0,0; used only by the Firmware
- window (normal) coordinates are those used in BASIC and are relative to 1,1 at the top left of the current window; they only coincide with logical coordinates when the window extends into the top left corner of the screen

027 &BB51 TXT RESET
Action: Resets the text VDU indirections and the control code table
Entry: No entry conditions
Exit: AF, BC, DE and HL are corrupt, and all the other registers are preserved

028 &BB54 TXT VDU ENABLE
Action: Allows characters to be printed on the screen in the current stream
Entry: No entry conditions
Exit: AF is corrupt, and all other registers are preserved

029 &BB57 TXT VDU DISABLE
Action: Prevents characters from being printed to the current stream
Entry: No entry conditions
Exit: AF is corrupt, and all the other registers are preserved

030 &BB5A TXT OUTPUT
Action: Output a character or control code (&00 to &1F) to the screen
Entry: A contains the character to output
Exit: All registers are preserved
Notes: Any control codes are obeyed and nothing is printed if the VDU is disabled; characters are printed using the TXT OUT ACTION routine; if using graphics printing mode, then control codes are printed and not obeyed

031 &BB5D TXT WR CHAR
Action: Print a character at the current cursor position — control codes are printed and not obeyed
Entry: A contains the character to be printed
Exit: AF, BC, DE and HL are corrupt, and all others are preserved
Notes: This routine uses the TXT WRITE CHAR indirection to put the character on the screen

032 &BB60 TXT RD CHAR
Action: Read a character from the screen at the current cursor position
Entry: No entry conditions
Exit: If it was successful then A contains the character that was read from the screen and Carry is true;
otherwise Carry is false, and A holds 0;
in either case, the other flags are corrupt, and all registers are preserved
Notes: This routine uses the TXT UNWRITE indirection

033 &BB63 TXT SET GRAPHIC
Action: Enables or disables graphics print character mode
Entry: To switch graphics printing mode on, A must be non-zero; to turn it off, A must contain zero
Exit: AF corrupt, and all other registers are preserved
Notes: When turned on, control codes are printed and not obeyed; characters are printed by GRA WR CHAR

034 &BB66 TXT WIN ENABLE
Action: Sets the boundaries of the current text window — uses physcial coordinates
Entry: H holds the column number of one edge, D holds the column number of the other edge, L holds the line number of one edge, and E holds the line number of the other edge
Exit: AF, BC, DE and HL are corrupt
Notes: The window is not cleared but the cursor is moved to the top left corner of the window

035 &BB69 TXT GET WINDOW
Action: Returns the size of the current window — returns physical coordinates
Entry: No entry conditions
Exit: H holds the column number of the left edge, D holds the column number of the right edge, L holds the line number of the top edge, E holds the line number of the bottom edge, A is corrupt, Carry is false if the window covers the entire screen, and the other registers are always preserved

036 &BB6C TXT CLEAR WINDOW
Action: Clears the window (of the current stream) and moves the cursor to the top left corner of the window
Entry: No entry conditions
Exit: AF, BC, DE and HL are corrupt, and all others are preserved

037 &BB6F TXT SET COLUMN

Action: Sets the cursor's horizontal position
Entry: A contains the logical column number to move the cursor to
Exit: AF and HL are corrupt, and all the other registers are preserved
Notes: See also TXT SET CURSOR

038 &BB72 TXT SET ROW

Action: Sets the cursor's vertical position
Entry: A contains the logical line number to move the cursor to
Exit: AF and HL are corrupt, and all others are preserved
Notes: See also TXT SET CURSOR

039 &BB75 TXT SET CURSOR

Action: Sets the cursor's vertical and horizontal position
Entry: H contains the logical column number and L contains the logical line number
Exit: AF and HL are corrupt, and all the others are preserved
Notes: See also TXT SET COLUMN and TXT SET ROW

040 &BB78 TXT GET CURSOR

Action: Gets the cursor's current position
Entry: No entry conditions
Exit: H holds the logical column number, L holds the logical line number, and A contains the roll count, the flags are corrupt, and all the other registers are preserved
Notes: The roll count is increased when the screen is scrolled down, and is decreased when it is scrolled up

041 &BB7B TXT CUR ENABLE

Action: Allows the text cursor to be displayed (if it is allowed by TXT CUR ON) — intended for use by the user
Entry: No entry conditions
Exit: AF is corrupt, and all other registers are preserved

042 &BB7E TXT CUR DISABLE

Action: Prevents the text cursor from being displayed — intended for use by the user
Entry: No entry conditions
Exit: AF is corrupt, and all others are preserved

043 &BB81 TXT CUR ON

Action: Allows the text cursor to be displayed — intended for use by the operating system
Entry: No entry conditions
Exit: All registers and flags are preserved

044 &BB84 TXT CUR OFF

Action: Prevents the text cursor from being displayed — intended for use by the operating system
Entry: No entry conditions
Exit: All registers and flags are preserved

045 &BB87 TXT VALIDATE

Action: Checks whether a cursor position is within the current window
Entry: H contains the logical column number to check, and L holds the logical line number
Exit: H holds the logical column number where the next character will be printed, L holds the logical line number;
if printing at this position would make the window scroll up, then Carry is false and B holds &FF;
if printing at this position would make the window scroll down, then Carry is false and B contains &00;
if printing at the specified cursor position would not scroll the window, then Carry is true and B is corrupt;
always, A and the other flags are corrupt, and all others are preserved

046 &BB8A TXT PLACE CURSOR

Action: Puts a 'cursor blob' on the screen at the current cursor position
Entry: No entry conditions
Exit: AF is corrupt, and all other registers are preserved
Notes: It is possible to have more than one cursor in a window (see also TXT DRAW CURSOR); do not use this routine twice without using TXT REMOVE CURSOR between

047 &BB8D TXT REMOVE CURSOR
Action: Removes a 'cursor blob' from the current cursor position
Entry: No entry conditions
Exit: AF is corrupt, and all the others are preserved
Notes: This should be used only to remove cursors created by TXT PLACE CURSOR, but see also TXT UNDRAW CURSOR

048 &BB90 TXT SET PEN
Action: Sets the foreground PEN for the current stream
Entry: A contains the PEN number to use
Exit: AF and HL are corrupt, and all other registers are preserved

049 &BB93 TXT GET PEN
Action: Gets the foreground PEN for the current stream
Entry: No entry conditions
Exit: A contains the PEN number, the flags are corrupt, and all other registers are preserved

050 &BB96 TXT SET PAPER
Action: Sets the background PAPER for the current stream
Entry: A contains the PEN number to use
Exit: AF and HL are corrupt, and all other registers are preserved

051 &BB99 TXT GET PAPER
Action: Gets the background PAPER for the current stream
Entry: No entry conditions
Exit: A contains the PEN number, the flags are corrupt, and all other registers are preserved

052 &BB9C TXT INVERSE
Action: Swaps the current PEN and PAPER colours over for the current stream
Entry: No entry conditions
Exit: AF and HL are corrupt, and all others are preserved

053 &BB9F TXT SET BACK
Action: Sets the character write mode to either opaque or transparent
Entry: For transparent mode, A must be non-zero; for opaque mode, A has to hold zero
Exit: AF and HL are corrupt, and all other registers are preserved
Notes: Setting the character write mode has no effects on the graphics VDU

054 &BBA2 TXT GET BACK
Action: Gets the character write mode for the current stream
Entry: No entry conditions
Exit: If in transparent mode, A is non-zero;
in opaque mode, A is zero;
in either case DE, HL and flags are corrupt, and the other registers are preserved

PEN AND INK

The term 'ink' is used by the handbooks and guides to designate both pen and the colour it contains and writes with. To differentiate in this guide, the following system will be used:

- 'Pen' is one of the 16, 4 or 2 PENs available in Modes 0, 1 or 2 respectively
- 'Foreground pen' is one of these pens selected to print the next character with; when the foreground pen is changed, only future printing is affected
- 'Paper' is one of the pens selected as background for the next character
- 'Colour' is one of 26 colours (inks) which a pen writes with, or a paper shows as background to a character; when a pen's colour is changed, all existing characters in that pen are changed as well as all future ones
- 'Encoded pen' is a special value accorded to a pen for use by the Firmware; it can be used as a mask to set all of a byte's pixels to that pen
- 'Hardware colour' is a special value accorded to a colour for use by the Hardware (colour is created by the hardware only, although it is selected by the software)

055 **&BBA5 TXT GET MATRIX**
Action: Gets the address of a character matrix
Entry: A contains the character whose matrix is to be found
Exit: If it is a user-defined matrix, then Carry is true;
if it is in the lower ROM then Carry is false;
in either event, HL contains the address of the matrix, A and other flags are corrupt, and others are preserved
Notes: The character matrix is stored in 8 bytes; the first byte is for the top row of the character, and the last byte refers to the bottom row of the character; bit 7 of a byte refers to the leftmost pixel of a line, and bit 0 refers to the rightmost pixel in Mode 2. For Modes 0 and 1, see page 21

056 **&BBA8 TXT SET MATRIX**
Action: Installs a matrix for a user-defined character
Entry: A contains the character which is being defined and HL contains the address of the matrix to be used
Exit: If the character is user-definable then Carry is true;
otherwise Carry is false, and no action is taken;
in both cases AF, BC, DE and HL are corrupt, and all other registers are preserved

057 **&BBAB TXT SET M TABLE**
Action: Sets the address of a user-defined matrix table
Entry: DE is the first character in the table and HL is the table's address (in the central 32K of RAM)
Exit: If there are no existing tables then Carry is false, and A and HL are both corrupt;
otherwise Carry is true, A is the first character and HL is the table's address;
in both cases BC, DE and the other flags are corrupt

058 **&BBAE TXT GET M TABLE**
Action: Gets the address of a user-defined matrix table
Entry: No entry conditions
Exit: See TXT SET M TABLE above for details of the values that can be returned

059 **&BBB1 TXT GET CONTROLS**
Action: Gets the address of the control code table
Entry: No entry conditions
Exit: HL contains the address of the table, and all others are preserved
Notes: The table has 32 entries, and each entry has three bytes:
byte 1 is the number of parameters needed by the control code
bytes 2 and 3 are the address of the routine, in the Lower ROM, to execute the control code

060 **&BBB4 TXT STR SELECT**
Action: Selects a new VDU text stream
Entry: A contains the value of the stream to change to
Exit: A contains the previously selected stream, HL and the flags are corrupt, and all others are preserved

061 **&BBB7 TXT SWAP STREAMS**
Action: Swaps the states of two stream attribute tables
Entry: B contains a stream number, and C contains the other stream number
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved
Notes: The foreground pen and paper, the window size, the cursor position, the character write mode and graphic character mode are all exchanged between the two streams

The Graphics VDU

062 **&BBBA GRA INITIALISE**
Action: Initialises the graphics VDU to its default set-up (ie its set-up when the computer is switched on)
Entry: No entry conditions
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved
Notes: Sets the graphics indirections to their defaults, sets the graphic paper to text pen 0 and the graphic pen to text pen 1, reset the graphics origin and move the graphics cursor to the bottom left of the screen, reset the graphics window and write mode to their defaults

063 &BBBD GRA RESET
Action: Resets the graphics VDU
Entry: No entry conditions
Exit: AF, BC, DE and HL are corrupt, and all others are preserved
Notes: Resets the graphics indirections and the graphics write mode to their defaults

064 &BBC0 GRA MOVE ABSOLUTE
Action: Moves the graphics cursor to an absolute screen position
Entry: DE contains the user X-coordinate and HL holds the user Y-coordinate
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved

065 &BBC3 GRA MOVE RELATIVE
Action: Moves the graphics cursor to a point relative to its present screen position
Entry: DE contains the X-distance to move and HL holds the Y-distance
Exit: AF, BC, DE and HL are corrupt, and all others are preserved

066 &BBC6 GRA ASK CURSOR
Action: Gets the graphics cursor's current position
Entry: No entry conditions
Exit: DE holds the user X-coordinate, HL holds the user Y-coordinate, AF is corrupt, and all others are preserved

067 &BBC9 GRA SET ORIGIN
Action: Sets the graphics user origin's screen position
Entry: DE contains the standard X-coordinate and HL holds the standard Y-coordinate
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved

068 &BBCC GRA GET ORIGIN
Action: Gets the graphics user origin's screen position
Entry: No entry conditions
Exit: DE contains the standard X-coordinate and HL holds the standard Y-coordinate, and all others are preserved

069 &BBCF GRA WIN WIDTH
Action: Sets the left and right edges of the graphics window
Entry: DE contains the standard X-coordinate of one edge and HL holds the standard X-coordinate of the other side
Exit: AF, BC, DE and HL are corrupt, and all the other registers are preserved
Notes: The default window covers the entire screen and is restored to its default when the mode is changed; used in conjunction with GRA WIN HEIGHT

070 &BBD2 GRA WIN HEIGHT
Action: Sets the top and bottom edges of the graphics window
Entry: DE contains the standard Y-coordinate of one side and HL holds the standard Y-coordinate of the other side
Exit: AF, BC, DE and HL are corrupt, and all others are preserved
Notes: See GRA WIN WIDTH for further details

GRAPHICS COORDINATES

Graphics position coordinates are expressed using three systems (each of which defines a 'point'):

- User coordinates are those relative to the user Origin (as set by BASIC's ORIGIN command or by the Firmware's GRA SET ORIGIN routine) which becomes 0,0; this system accords with 'absolute' coordinates used in BASIC, but is also used by the Firmware
- Relative coordinates are relative to the current graphics position
- Standard coordinates relate to 0,0 at the bottom left of the screen, and are used only for setting the Origin or the size of the graphics window, and are independent of both of these

In addition, pixel coordinates are defined using the following system:

- Base coordinates relate to 0,0 at the bottom left of the screen, but they relate exclusively to pixels, and so they are calculated as follows:
  x base coordinate = (x standard coordinate)/8 or /4 or /2 (for Modes 0, 1 or 2 respectively)
  y base coordinate = (y standard coordinate)/2 (for all Modes)

071 &BBD5 GRA GET W WIDTH
Action: Gets the left and right edges of the graphics window
Entry: No entry conditions
Exit: DE contains the standard X-coordinate of the left edge and HL contains the standard X-coordinate of the right edge, AF is corrupt, and all other registers are preserved

072 &BBD8 GRA GET W HEIGHT
Action: Gets the top and bottom edges of the graphics window
Entry: No entry conditions
Exit: DE contains the standard Y-coordinate of the top edge and HL contains the standard Y-coordinate of the bottom edge, AF is corrupt, and all other registers are preserved

073 &BBDB GRA CLEAR WINDOW
Action: Clears the graphics window to the graphics paper colour and moves the cursor back to the user origin
Entry: No entry conditions
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved

074 &BBDE GRA SET PEN
Action: Sets the graphics PEN
Entry: A contains the required text PEN number
Exit: AF is corrupt, and all other registers are preserved

075 &BBE1 GRA GET PEN
Action: Gets the graphics PEN
Entry: No entry conditions
Exit: A contains the text PEN number, the flags are corrupt, and all other registers are preserved

076 &BBE4 GRA SET PAPER
Action: Sets the graphics PAPER
Entry: A contains the required text PEN number
Exit: AF corrupt, and all others are preserved

077 &BBE7 GRA GET PAPER
Action: Gets the graphics PAPER
Entry: No entry conditions
Exit: A contains the text PEN number, the flags are corrupt, and all others are preserved

078 &BBEA GRA PLOT ABSOLUTE
Action: Plots a point at an absolute user coordinate, using the GRA PLOT indirection
Entry: DE contains the user X-coordinate and HL holds the user Y-coordinate
Exit: AF, BC, DE and HL are corrupt, and all others are preserved

079 &BBED GRA PLOT RELATIVE
Action: Plots a point at a position relative to the current graphics cursor, using the GRA PLOT indirection
Entry: DE contains the relative X-coordinate and HL contains the relative Y-coordinate
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved

080 &BBF0 GRA TEST ABSOLUTE
Action: Moves to an absolute position, and tests the point there using the GRA TEST indirection
Entry: DE contains the user X-coordinate and HL holds the user Y-coordinate for the point you wish to test
Exit: A contains the pen at the point, and BC, DE, HL and flags are corrupt, and all others are preserved

GRAPHICS PEN AND PAPER

- Graphics Pen is one of the available text pens, selected for drawing lines with; it can be different from the current foreground (text) pen
- Graphics Paper is one of the available text pens, selected to act as a background when printing characters in Graphics write mode; it can be different from the current text pen
- The colours these produce are those set for the text pen in use, using BASIC's INK command or SCR SET INK

081 &BBF3 GRA TEST RELATIVE
Action: Moves to a position relative to the current position, and tests the point there using the GRA TEST indirection
Entry: DE contains the relative X-coordinate and HL contains the relative Y-coordinate
Exit: A contains the pen at the point, and BC, DE, HL and flags are corrupt, and all others are preserved

082 &BBF6 GRA LINE ABSOLUTE
Action: Draws a line from the current graphics position to an absolute position, using GRA LINE
Entry: DE contains the user X-coordinate and HL holds the user Y-coordinate of the end point
Exit: AF, BC, DE and HL are corrupt, and all others are preserved
Notes: The line will be plotted in the current graphics pen colour (may be masked to produce a dotted line on a 6128)

083 &BBF9 GRA LINE RELATIVE
Action: Draws a line from the current graphics position to a relative screen position, using GRA LINE
Entry: DE contains the relative X-coordinate and HL contains the relative Y-coordinate
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved
Notes: See GRA LINE ABSOLUTE above for details of how the line is plotted

084 &BBFC GRA WR CHAR
Action: Writes a character onto the screen at the current graphics position
Entry: A contains the character to be put onto the screen
Exit: AF, BC, DE and HL are corrupt, and all the other registers are preserved
Notes: As in BASIC, all characters including control codes are printed; the character is printed with its top left corner at the current graphics position; the graphics position is moved one character width to the right so that it is ready for another character to be printed

The Screen Pack

085 &BBFF SCR INITIALISE
Action: Initialises the Screen Pack to the default values used when the computer is first switched on
Entry: No entry conditions
Exit: AF, BC, DE and HL are corrupt, and all others are preserved
Notes: All screen indirections are restored to their default settings, as are inks and flashing speeds; the mode is switched to MODE 1 and the screen is cleared with PEN 0; the screen address is moved to &C000 and the screen offset is set to zero

086 &BC02 SCR RESET
Action: Resets the Screen Pack's indirections, flashing speeds and inks to their default values
Entry: No entry conditions
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved

087 &BC05 SCR SET OFFSET
Action: Sets the screen offset to the specified values — this can cause the screen to scroll
Entry: HL contains the required offset, which should be even
Exit: AF and HL are corrupt, and all others are preserved
Notes: The screen offset is reset to 0 whenever its mode is set, or it is cleared by SCR CLEAR (but not BASIC's CLS)

088 &BC08 SCR SET BASE
Action: Sets the location in memory of the screen — effectively can only be &C000 or &4000
Entry: A contains the most significant byte of the screen address required
Exit: AF and HL are corrupt, and all other registers are preserved
Notes: The screen memory can only be set at 16K intervals (ie &0000, &4000, &8000, &C000) and when the computer is first switched on the 16K of screen memory is located at &C000

089 &BC0B SCR GET LOCATION
Action: Gets the location of the screen memory and also the screen offset
Entry: No entry conditions
Exit: A holds the most significant byte of the screen address, HL holds the current offset, and all others are preserved

090 **&BC0E SCR SET MODE**
Action: Sets the screen mode
Entry: A contains the mode number — it has the same value and characteristics as in BASIC
Exit: AF, BC, DE and HL are corrupt, and all others are preserved
Notes: The windows are set to cover the whole screen and the graphics origin is set to the bottom left corner of the screen; in addition, the current stream is set to zero, and the screen offset is zeroed

091 **&BC11 SCR GET MODE**
Action: Gets the current screen mode
Entry: No entry conditions
Exit: If the mode is 0, then Carry is true, Zero is false, and A contains 0;
if the mode is 1, then Carry is false, Zero is true, and A contains 1;
if the mode is 2, then Carry is false, Zero is false, and A contains 2;
in all cases the other flags are corrupt and all the other registers are preserved

092 **&BC14 SCR CLEAR**
Action: Clears the whole of the screen
Entry: No entry conditions
Exit: AF, BC, DE and HL are corrupt, and all others are preserved

093 **&BC17 SCR CHAR LIMITS**
Action: Gets the size of the whole screen in terms of the numbers of characters that can be displayed
Entry: No entry conditions
Exit: B contains the number of characters across the screen, C contains the number of characters down the screen, AF is corrupt, and all other registers are preserved

094 **&BC1A SCR CHAR POSITION**
Action: Gets the memory address of the top left corner of a specified character position
Entry: H contains the character physical column and L contains the character physical row
Exit: HL contains the memory address of the top left corner of the character, B holds the width in bytes of a character in the present mode, AF is corrupt, and all other registers are preserved

095 **&BC1D SCR DOT POSITION**
Action: Gets the memory address of a pixel at a specified screen position
Entry: DE contains the base X-coordinate of the pixel, and HL contains the base Y-coordinate
Exit: HL contains the memory address of the pixel, C contains the bit mask for this pixel, B contains the number of pixels stored in a byte minus 1, AF and DE are corrupt, and all others are preserved

096 **&BC20 SCR NEXT BYTE**
Action: Calculates the screen address of the byte to the right of the specified screen address (may be on the next line)
Entry: HL contains the screen address
Exit: HL holds the screen address of the byte to the right of the original screen address, AF is corrupt, all others are preserved

097 **&BC23 SCR PREV BYTE**
Action: Calculates the screen address of the byte to the left of the specified screen address (this address may actually be on the previous line)
Entry: HL contains the screen address
Exit: HL holds the screen address of the byte to the left of the original address, AF is corrupt, all others are preserved

098 **&BC26 SCR NEXT LINE**
Action: Calculates the screen address of the byte below the specified screen address
Entry: HL contains the screen address
Exit: HL contains the screen address of the byte below the original screen address, AF is corrupt, and all the other registers are preserved

099 **&BC29 SCR PREV LINE**
Action: Calculates the screen address of the byte above the specified screen address
Entry: HL contains the screen address
Exit: HL holds the screen address of the byte above the original address, AF is corrupt, and all others are preserved

100 &BC2C SCR INK ENCODE

Action: Converts a PEN to provide a mask which, if applied to a screen byte, will convert all of the pixels in the byte to the appropriate PEN

Entry: A contains a PEN number

Exit: A contains the encoded value of the PEN, the flags are corrupt, and all other registers are preserved

Notes: The mask returned is different in each of the screen modes (see page 21)

101 &BC2F SCR INK DECODE

Action: Converts a PEN mask into the PEN number (see SCR INK ENCODE for the reverse process)

Entry: A contains the encoded value of the PEN

Exit: A contains the PEN number, the flags are corrupt, and all others are preserved

102 &BC32 SCR SET INK

Action: Sets the colours of a PEN — if the two values supplied are different then the colours will alternate (flash)

Entry: A contains the PEN number, B contains the first colour, and C holds the second colour

Exit: AF, BC, DE and HL are corrupt, and all others are preserved

103 &BC35 SCR GET INK

Action: Gets the colours of a PEN

Entry: A contains the PEN number

Exit: B contains the first colour, C holds the second colour, and AF, DE and HL are corrupt, and all others are preserved

104 &BC38 SCR SET BORDER

Action: Sets the colours of the border — again if two different values are supplied, the border will flash

Entry: B contains the first colour, and C contains the second colour

Exit: AF, BC, DE and HL are corrupt, and all others are preserved

105 &BC3B SCR GET BORDER

Action: Gets the colours of the border

Entry: No entry conditions

Exit: B contains the first colour, C holds the second colour, and AF, DE and HL are corrupt, and all others are preserved

106 &BC3E SCR SET FLASHING

Action: Sets the speed with which the border's and PENs' colours flash

Entry: H holds the time that the first colour is displayed, L holds the time the second colour is displayed for

Exit: AF and HL are corrupt, and all other registers are preserved

Notes: The length of time that each colour is shown is measured in 1/50ths of a second, and a value of 0 is taken to mean 256 * 1/50 seconds — the default value is 10 * 1/50 seconds

107 &BC41 SCR GET FLASHING

Action: Gets the periods with which the colours of the border and PENs flash

Entry: No entry conditions

Exit: H holds the duration of the first colour, L holds the duration of the second colour, AF is corrupt, and all other registers are preserved — see SCR SET FLASHING for the units of time used

108 &BC44 SCR FILL BOX

Action: Fills an area of the screen with an ink — this only works for 'character-sized' blocks of screen

Entry: A contains the mask for the ink that is to be used, H contains the left hand column of the area to fill, D contains the right hand column, L holds the top line, and E holds the bottom line of the area (using physical coordinates)

Exit: AF, BC, DE and HL are corrupt, and all others are preserved

109 &BC47 SCR FLOOD BOX

Action: Fills an area of the screen with an ink — this only works for 'byte-sized' blocks of screen

Entry: C contains the encoded PEN that is to be used, HL contains the screen address of the top left hand corner of the area to fill, D contains the width of the area to be filled in bytes, and E contains the height of the area to be filled in screen lines

Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved

Notes: The whole of the area to be filled must lie on the screen otherwise unpredictable results may occur

110 &BC4A SCR CHAR INVERT

Action: Inverts a character's colours; all pixels in one PEN's colour are printed in another PEN's colour, and vice versa
Entry: B contains one encoded PEN, C contains the other encoded PEN, H contains the physical column number, and L contains the physical line number of the character that is to be inverted
Exit: AF, BC, DE and HL are corrupt, and all the other registers are preserved

111 &BC4D SCR HW ROLL

Action: Scrolls the entire screen up or down by eight pixel rows (ie one character line)
Entry: B holds the direction that the screen will roll, A holds the encoded PAPER which the new line will appear in
Exit: AF, BC, DE and HL are corrupt, and all others are preserved
Notes: This alters the screen offset; to roll down, B must hold zero, and to roll upwards B must be non-zero

112 &BC50 SCR SW ROLL

Action: Scrolls part of the screen up or down by eight pixel lines — only for 'character-sized' blocks of the screen
Entry: B holds the direction to roll the screen, A holds the encoded PAPER which the new line will appear in, H holds the left column of the area to scroll, D holds the right column, L holds the top line, E holds the bottom line
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved
Notes: The area of the screen is moved by copying it; to roll down, B must hold zero, and to roll upwards B must be non-zero; this routine uses physical coordinates

113 &BC53 SCR UNPACK

Action: Changes a character matrix from its eight byte standard form into a set of pixel masks which are suitable for the current mode — four *8 bytes are needed in mode 0, two *8 bytes in mode 1, and 8 bytes in mode 2
Entry: HL contains the address of the matrix, and DE contains the address where the masks are to be stored
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved

114 &BC56 SCR REPACK

Action: Changes a set of pixel masks (for the current mode) into a standard eight byte character matrix
Entry: A contains the encoded foreground PEN to be matched against (ie the PEN that is to be regarded as being set in the character), H holds the physical column of the character to be 'repacked', L holds the physical line of the character, and DE contains the address of the area where the character matrix will be built
Exit: AF, BC, DE and HL are corrupt, and all the others are preserved

115 &BC59 SCR ACCESS

Action: Sets the screen write mode for graphics
Entry: A contains the write mode (0=Fill, 1=XOR, 2=AND, 3=OR)
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved
Notes: The fill mode means that the ink that printing was requested in is the ink that appears on the screen; in XOR mode, the specified ink is XORed with ink that is at that point on the screen already before printing; a similar situation occurs with the AND and OR modes

116 &BC5C SCR PIXELS

Action: Puts a pixel or pixels on the screen regardless of the write mode specified by SCR ACCESS above
Entry: B contains the mask of the PEN to be drawn with, C contains the pixel mask, and HL holds the screen address of the pixel
Exit: AF is corrupt, and all others are preserved

117 &BC5F SCR HORIZONTAL

Action: Draws a horizontal line on the screen using the current graphics write mode
Entry: A contains the encoded PEN to be drawn with, DE contains the base X-coordinate of the start of the line, BC contains the end base X-coordinate, and HL contains the base Y-coordinate
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved
Notes: The start X-coordinate must be less than the end X-coordinate

118 &BC62 SCR VERTICAL

Action: Draws a vertical line on the screen using the current graphics write mode
Entry: A contains the encoded PEN to be drawn with, DE contains the base X-coordinate of the line, HL holds the start base Y-coordinate, and BC contains the end base Y-coordinate — the start coordinate must be less than the end coordinate
Exit: AF, BC, DE and HL are corrupt, and all the other registers are preserved

The Cassette/AMSDOS Manager

NOTE: **Some of these routines are only applicable to the cassette manager; where a disc version exists it is indicated by an asterisk (*) next to the command name. These disc version jumpblocks are automatically installed by the Operating System on switch on.**

119 &BC65 CAS INITIALISE
Action: Initialises the cassette manager
Entry: No entry conditions
Exit: AF, BC, DE and HL are corrupt, and all the other registers are preserved
Notes: Both read and write streams are closed; tape messages are switched on; the default speed is reselected

120 &BC68 CAS SET SPEED
Action: Sets the speed at which the cassette manager saves programs
Entry: HL holds the length of 'half a zero' bit, and A contains the amount of precompensation
Exit: AF and HL are corrupt
Notes: The value in HL is the length of time that half a zero bit is written as; a one bit is twice the length of a zero bit; the default values (ie SPEED WRITE 0) are 333 microseconds (HL) and 25 microseconds (A)
for SPEED WRITE 1, the values are given as 107 microseconds and 50 microseconds respectively

121 &BC6B CAS NOISY
Action: Enables or disables the display of cassette handling messages
Entry: To enable the messages then A must be 0, otherwise the messages are disabled
Exit: AF is corrupt, and all other registers are preserved

122 &BC6E CAS START MOTOR
Action: Switches on the tape motor
Entry: No entry conditions
Exit: If the motor operates properly then Carry is true;
if ESC was pressed then Carry is false;
in either case, A contains the motor's previous state, the flags are corrupt, and all others are preserved

123 &BC71 CAS STOP MOTOR
Action: Switches off the tape motor
Entry: No entry conditions
Exit: If the motor turns off then Carry is true;
if ESC was pressed then Carry is false;
in both cases, A holds the motor's previous state, the other flags are corrupt, all others are preserved

124 &BC74 CAS RESTORE MOTOR
Action: Resets the tape motor to its previous state
Entry: A contains the previous state of the motor (eg from CAS START MOTOR or CAS STOP MOTOR)
Exit: If the motor operates properly then Carry is true;
if ESC was pressed then Carry is false;
in all cases, A and the other flags are corrupt and all others are preserved

Format used for recording on tape

LEADER	DATA BLOCK 1	DATA BLOCK 2	DATA BLOCK...	TRAILER

Each data block consists of 256 bytes and a two byte checksum; in addition there is also a header which is the equivalent of a data block; each tape block contains a maximum of eight data blocks, although the last tape block may contain fewer data blocks; the leader consists of 2048 bits set to 1, followed by 1 bit set to 0, and then a synchronising byte; the trailer consists of 32 bits set to 1; for an example of cassette headers, see &B11F (&B807 for the 464) in the memory map section

For an example of disc headers, see &A755 (or &A755 for the 464) again in the memory map section of the guide

125 &BC77 CAS IN OPEN *

Action: Opens an input buffer and reads the first block of the file

Entry: B contains the length of the filename, HL contains the filename's address, and DE contains the address of the 2K buffer to use for reading the file

Exit: If the file was opened successfully, then Carry is true, Zero is false, HL holds the address of a buffer containing the file header data, DE holds the address of the destination for the file, BC holds the file length, and A holds the file type;
if the read stream is already open then Carry and Zero are false, A contains an error number (664/6128 only) and BC, DE and HL are corrupt;
if ESC was pressed by the user, then Carry is false, Zero is true, A holds an error number (664/6128 only) and BC, DE and HL are corrupt;
in all cases, IX and the other flags are corrupt, and the others are preserved

Notes: A filename of zero length means 'read the next file on the tape'; the stream remains open until it is closed by either CAS IN CLOSE or CAS IN ABANDON

Disc: Similar to tape except that if there is no header on the file, then a fake header is put into memory by this routine

126 &BC7A CAS IN CLOSE *

Action: Closes an input file

Entry: No entry conditions

Exit: If the file was closed successfully, then Carry is true and A is corrupt;
if the read stream was not open, then Carry is false, and A holds an error code (664/6128 only);
in both cases, BC, DE, HL and the other flags are all corrupt

Disc: All the above applies, but also if the file failed to close for any other reason, then Carry is false, Zero is true and A contains an error number; in all cases the drive motor is turned off immediately

127 &BC7D CAS IN ABANDON *

Action: Abandons an input file

Entry: No entry conditions

Exit: AF, BC, DE and HL are corrupt, and all others are preserved

Disc: All the above applies for the disc routine

128 &BC80 CAS IN CHAR *

Action: Reads in a single byte from a file

Entry: No entry conditions

Exit: If a byte was read, then Carry is true, Zero is false, and A contains the byte read from the file;
if the end of file was reached, then Carry and Zero are false, A contains an error number (664/6128 only) or is corrupt (for the 464);
if ESC was pressed, then Carry is false, Zero is true, and A holds an error number (664/6128 only) or is corrupt (for the 464);
in all cases, IX and the other flags are corrupt, and all others are preserved

Disc: All the above applies for the disc routine

129 &BC83 CAS IN DIRECT *

Action: Reads an entire file directly into memory

Entry: HL contains the address where the file is to be placed in RAM

Exit: If the operation was successful, then Carry is true, Zero is false, HL contains the entry address and A is corrupt;
if it was not open, then Carry and Zero are both false, HL is corrupt, and A holds an error code (664/6128) or is corrupt (464);
if ESC was pressed, Carry is false, Zero is true, HL is corrupt, and A holds an error code (664/6128 only);
in all cases, BC, DE and IX and the other flags are corrupt, and the others are preserved

Notes: This routine cannot be used once CAS IN CHAR has been used

Disc: All the above applies to the disc routine

130 &BC86 CAS RETURN *

Action: Puts the last byte read back into the input buffer so that it can be read again at a later time

Entry: No entry conditions

Exit: All registers are preserved

Notes: The routine can only return the last byte read and at least one byte must have been read

Disc: All the above applies to the disc routine

131 &BC89 CAS TEST EOF *

Action: Tests whether the end of file has been encountered

Entry: No entry conditions

Exit: If the end of file has been reached, then Carry and Zero are false, and A is corrupt;
if the end of file has not been encountered, then Carry is true, Zero is false, and A is corrupt;
if ESC was pressed then Carry is false, Zero is true and A contains an error number (664/6128 only);
in all cases, IX and the other flags are corrupt, and all others are preserved

Disc: All the above applies to the disc routine

132 &BC8C CAS OUT OPEN *

Action: Opens an output file

Entry: B contains the length of the filename, HL contains the address of the filename, and DE holds the address of the 2K buffer to be used

Exit: If the file was opened correctly, then Carry is true, Zero is false, HL holds the address of the buffer containing the file header data that will be written to each block, and A is corrupt;
if the write stream is already open, then Carry and Zero are false, A holds an error number (664/6128) and HL is corrupt;
if ESC was pressed then Carry is false, Zero is true, A holds an error number (664/6128) and HL is corrupt;
in all cases, BC, DE, IX and the other flags are corrupt, and the others are preserved

Notes: The buffer is used to store the contents of a file block before it is actually written to tape

Disc: The same as for tape except that the filename must be present in its usual AMSDOS format

133 &BC8F CAS OUT CLOSE *

Action: Closes an output file

Entry: No entry conditions

Exit: If the file was closed successfully, then Carry is true, Zero is false, and A is corrupt;
if the write stream was not open, then Carry and Zero are false and A holds an error code (664/6128 only);
if ESC was pressed then Carry is false, Zero is true, and A contains an error code (664/6128 only);
in all cases, BC, DE, HL, IX and the other flags are all corrupt

Notes: The last block of a file is written only when this routine is called; if writing the file is to be abandoned, then CAS OUT ABANDON should be used instead

Disc: All the above applies to the disc routine

134 &BC92 CAS OUT ABANDON *

Action: Abandons an output file

Entry: No entry conditions

Exit: AF, BC, DE and HL are corrupt, and all others are preserved

Notes: When using this routine, the current last block of the file is not written to the tape

Disc: Similar to the tape routine; if more than 16K of a file has been written to the disc, then the first 16K of the file will exist on the disc with a file extension of .$$$ because each 16K section of the file requires a separate directory entry

135 &BC95 CAS OUT CHAR *

Action: Writes a single byte to a file

Entry: A contains the byte to be written to the file output buffer

Exit: If a byte was written to the buffer, then Carry is true, Zero is false, and A is corrupt;
if the file was not open, then Carry and Zero are false, and A contains an error number (664/6128 only) or is corrupt (on the 464);
if ESC was pressed, then Carry is false, Zero is true, and A contains an error number (664/6128 only) or it is corrupt (on the 464);
in all cases, IX and the other flags are corrupt, and all others are preserved

Notes: If the 2K buffer is full of data then it is written to the tape before the new character is placed in the buffer; it is important to call CAS OUT CLOSE when all the data has been sent to the file so that the last block is written to the tape

Disc: All the above applies to the disc routine

For a full list of error codes that may be returned from both tape and disc routines, please see page 46

136 &BC98 CAS OUT DIRECT *

Action: Writes an entire file directly to tape

Entry: HL contains the address of the data which is to be written to tape, DE contains the length of this data, BC contains the execution address, and A contains the file type

Exit: If the operation was successful, then Carry is true, Zero is false, and A is corrupt;
if the file was not open, Carry and Zero are false, A holds an error number (664/6128) or is corrupt (464);
if ESC was pressed, then Carry is false, Zero is true, and A holds an error code (664/6128 only);
in all cases BC, DE, HL, IX and the other flags are corrupt, and the others are preserved

Notes: This routine cannot be used once CAS OUT CHAR has been used

Disc: All the above applies to the disc routine

137 &BC9B CAS CATALOG *

Action: Creates a catalogue of all the files on the tape

Entry: DE contains the address of the 2K buffer to be used to store the information

Exit: If the operation was successful, then Carry is true, Zero is false, and A is corrupt;
if the read stream is already being used, then Carry and Zero are false, and A holds an error code (664/6128) or is corrupt (for the 464);
in all cases, BC, DE, HL, IX and the other flags are corrupt and all others are preserved

Notes: This routine is only left when the ESC key is pressed (cassette only) and is identical to BASIC's CAT command

Disc: All the above applies, except that a sorted list of files is displayed; system files are not listed by this routine

138 &BC9E CAS WRITE

Action: Writes data to the tape in one long file (ie not in 2K blocks)

Entry: HL contains the address of the data to be written to tape, DE contains the length of the data to be written, and A contains the sync character

Exit: If the operation was successful, then Carry is true and A is corrupt;
if an error occurred then Carry is false and A contains an error code;
in both cases, BC, DE, HL and IX are corrupt, and all other registers are preserved

Notes: For header records the sync character is &2C, and for data it is &16; this routine starts and stops the cassette motor and also turns off interrupts whilst writing data

139 &BCA1 CAS READ

Action: Reads data from the tape in one long file (ie as originally written by CAS WRITE only)

Entry: HL holds the address to place the file, DE holds the length of the data, and A holds the expected sync character

Exit: If the operation was successful, then Carry is true and A is corrupt;
if an error occurred then Carry is false and A contains an error code;
in both cases, BC, DE, HL and IX are corrupt, and all other registers are preserved

Notes: For header records the sync character is &2C, and for data it is &16; this routine starts and stops the cassette motor and turns off interrupts whilst reading data

The full list of error codes that can be returned from the tape and disc routines are as follows:

TAPE:

&00	The ESC key was pressed	&03	The file data did not match that in memory
&01	A bit was too long to read or write (overrun)	&0E	The stream is not open for reading or writing
&02	A failure of the cassette deck was detected	&0F	The hard end of file marker was met

DISC: if bit 6 of the error isn't set, then it was detected by AMSDOS; if bit 7 is set, then it has been reported to the user

&00	The ESC key was pressed	&13	Directory is full
&0E	The stream is not open for reading or writing	&14	Disc is full
&0F	The hard end of file marker was met	&15	Disc has been changed while files were open
&10	Bad command — incorrect filename	&16	File is read-only
&11	File already exists	&1A	The CPM soft end of file marker was met
&12	File doesn't exist		

if bit 6 is set, then the error was detected by the disc controller, and the error is bit sensitive as follows:

bit 5	Data error	bit 4	Overrun error	bit 3	Drive is not ready
bit 2	No data — can't find sector	bit 1	Disc is write protected	bit 0	Address mark missing

140 &BCA4 CAS CHECK

Action: Compares the contents of memory with a file record (ie header or data) on tape

Entry: HL contains the address of the data to check, DE contains the length of the data, and A holds the sync character that was used when the file was originally written to the tape

Exit: If the two are identical, then Carry is true and A is corrupt;
if an error occurred then Carry is false and A holds an error code;
in all cases, BC, DE, HL, IX and other flags are corrupt, and all other registers are preserved

Notes: For header records the sync character is &2C, and for data it is &16; this routine starts and stops the cassette motor and turns off interrupts whilst reading data; does not have to read the whole of a record, but must start at the beginning

The Sound Manager

141 &BCA7 SOUND RESET

Action: Resets the sound manager by clearing the sound queues and abandoning any current sounds

Entry: No entry conditions

Exit: AF, BC, DE and HL are corrupt, and all others are preserved

142 &BCAA SOUND QUEUE

Action: Adds a sound to the sound queue of a channel

Entry: HL contains the address of a series of bytes which define the sound and are stored in the central 32K of RAM

Exit: If the sound was successfully added to the queue, then Carry is true and HL is corrupt;
if one of the sound queues was full, then Carry is false and HL is preserved;
in either case, A, BC, DE, IX and the other flags are corrupt, and all others are preserved

Notes: The bytes required to define the sound are as follows:

byte 0 — channel status byte (see the box below)
byte 1 — volume envelope to use
byte 2 — tone envelope to use
bytes 3 & 4 — tone period
byte 5 — noise period
byte 6 — start volume
bytes 7 & 8 — duration of the sound, or envelope repeat count

143 &BCAD SOUND CHECK

Action: Gets the status of a sound channel

Entry: A contains the channel to test — for channel A, bit 0 set; for channel B, bit 1 set; for channel C, bit 2 set

Exit: A contains the channel status, BC, DE, HL and flags are corrupt, and all others are preserved

Notes: The channel status returned is bit significant, as follows:

bits 0 to 2 — the number of free spaces in the sound queue
bit 3 — trying to rendezvous with channel A
bit 4 — trying to rendezvous with channel B
bit 5 — trying to rendezvous with channel C
bit 6 — holding the channel
bit 7 — producing a sound

144 &BCB0 SOUND ARM EVENT

Action: Sets up an event which will be activated when a space occurs in a sound queue

Entry: A contains the channel to set the event up for (see SOUND CHECK for the bit values this can take), and HL holds the address of the event block

Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved

Notes: The event block must be initialised by KL INIT EVENT and is disarmed when the event itself is run

The channel status byte is bit significant, as follows:

bit 0 — play on channel A
bit 1 — play on channel B
bit 2 — play on channel C
bit 3 — rendezvous with channel A
bit 4 — rendezvous with channel B
bit 5 — rendezvous with channel C
bit 6 — hold the channel
bit 7 — flush (empty) the sound queue

145 &BCB3 SOUND RELEASE

Action: Allows the playing of sounds on specific channels that had been stopped by SOUND HOLD
Entry: A contains the sound channels to be released (see SOUND CHECK for the bit values this can take)
Exit: AF, BC, DE, HL and IX are corrupt, and all others are preserved

146 &BCB6 SOUND HOLD

Action: Immediately stops all sound output (on all channels)
Entry: No entry conditions
Exit: If a sound was being made, then Carry is true;
if no sound was being made, then Carry is false;
in all cases, A, BC, HL and other flags are corrupt, and all others are preserved
Notes: When the sounds are restarted, they will begin from exactly the same place that they were stopped

147 &BCB9 SOUND CONTINUE

Action: Restarts all sound output (on all channels)
Entry: No entry conditions
Exit: AF, BC, DE and IX are corrupt, and all others are preserved

148 &BCBC SOUND AMPL ENVELOPE

Action: Sets up a volume envelope
Entry: A holds an envelope number (from 1 to 15), HL holds the address of a block of data for the envelope
Exit: If it was set up properly, Carry is true, HL holds the data block address +16, A and BC are corrupt;
if the envelope number is invalid, then Carry is false, and A, B and HL are preserved;
in either case, DE and the other flags are corrupt, and all other registers are preserved
Notes: All the rules of enevelopes in BASIC also apply; the block of the data for the envelope is set up as follows:

byte 0 — number of sections in the envelope
bytes 1 to 3 — first section of the envelope
bytes 4 to 6 — second section of the envelope
bytes 7 to 9 — third section of the envelope
bytes 10 to 12 — fourth section of the envelope
bytes 13 to 15 — fifth section of the envelope

Each section of the envelope has three bytes set out as follows:

byte 0 — step count (with bit 7 set)
byte 1 — step size
byte 2 — pause time

or if it is a hardware envelope, then each section takes the following form:

byte 0 — envelope shape (with bit 7 not set)
bytes 1 and 2 — envelope period

See also SOUND TONE ENVELOPE below

149 &BCBF SOUND TONE ENVELOPE

Action: Sets up a tone envelope
Entry: A holds an envelope number (from 1 to 15), HL holds the address of a block of data for the envelope
Exit: If it was set up properly, Carry is true, HL holds the data block address +16, A and BC are corrupt;
if the envelope number is invalid, then Carry is false, and A, B and HL are preserved;
in either case, DE and the other flags are corrupt, and all other registers are preserved
Notes: All the rules of envelopes in BASIC also apply; the block of the data for the envelope is set up as follows:

byte 0 — number of sections in the envelope
bytes 1 to 3 — first section of the envelope
bytes 4 to 6 — second section of the envelope
bytes 7 to 9 — third section of the envelope
bytes 10 to 12 — fourth section of the envelope
bytes 13 to 15 — fifth section of the envelope

Each section of the envelope has three bytes set out as follows:

byte 0 — step count
byte 1 — step size
byte 2 — pause time

See also SOUND AMPL ENVELOPE above

150 &BCC2 SOUND A ADDRESS
Action: Gets the address of the data block associated with a volume envelope
Entry: A contains an envelope number (from 1 to 15)
Exit: If it was found, then Carry is true, HL holds the data block's address, and BC holds its length;
if the envelope number is invalid, then Carry is false, HL is corrupt and BC is preserved;
in both cases, A and the other flags are corrupt, and all others are preserved

151 &BCC5 SOUND T ADDRESS
Action: Gets the address of the data block associated with a tone envelope
Entry: A contains an envelope number (from 1 to 15)
Exit: If it was found, then Carry is true, HL holds the data block's address, and BC holds its length;
if the envelope number is invalid, then Carry is false, HL is corrupt and BC is preserved;
in both cases, A and the other flags are corrupt, and all others are preserved

The Kernel

152 &BCC8 KL CHOKE OFF
Action: Clears all event queues and timer lists, with the exception of keyboard scanning and sound routines
Entry: No entry conditions
Exit: B contains the foreground ROM select address (if any), DE contains the ROM entry address, C holds the ROM select address for a RAM foreground program, AF and HL are corrupt, and all others are preserved

153 &BCCB KL ROM WALK
Action: Finds and initialises all background ROMs
Entry: DE holds the address of the first usable byte of memory, HL holds the address of the last usable byte
Exit: DE holds the address of the new first usable byte of memory, HL holds the address of the new last usable byte, AF and BC are corrupt, and all other registers are preserved
Notes: This routine looks at the ROM select addresses from 0 to 15 (1 to 7 for the 464) and calls the initialisation routine of any ROMs present; these routines may reserve memory by adjusting DE and HL before returning control to KL ROM WALK, and the ROM is then added to the list of command handling routines

154 &BCCE KL INIT BACK
Action: Finds and initialises a specific background ROM
Entry: C contains the ROM select address of the ROM, DE holds the address of the first usable byte of memory, HL holds the address of the last usable byte of memory
Exit: DE holds the address of the new first usable byte of memory, HL holds the address of the new last usable byte, AF and B are corrupt, and all other registers are preserved
Notes: The ROM select address must be in the range of 0 to 15 (or 1 to 7 for the 464) although address 7 is for the AMSDOS/CPM ROM if present. The ROM's initialisation routine is then called and some memory may be reserved for the ROM by adjusting the values of DE and HL before returning control to KL INIT BACK

155 &BCD1 KL LOG EXT
Action: Logs on a new RSX to the firmware — see the following page for more detail
Entry: BC contains the address of the RSX's command table, HL contains the address of four bytes exclusively for use by the firmware
Exit: DE is corrupt, and all other registers are preserved

156 &BCD4 KL FIND COMMAND
Action: Searches an RSX, background ROM or foreground ROM, to find a command in its table
Entry: HL contains the address of the command name (in RAM only) which is being searched for
Exit: If the name was found in a RSX or background ROM then Carry is true, C contains the ROM select address, and HL contains the address of the routine;
if the command was not found, then Carry is false, C and HL are corrupt;
in either case, A, B and DE are corrupt, and all others are preserved
Notes: The command names should be in upper case and the last character should have &80 added to it; the sequence of searching is RSXs, then ROMs with lower numbers before ROMs with higher numbers

RESIDENT SYSTEM EXTENSION (RSX) COMMAND TABLE:

2 byte address of the Command Name Table
JP xxxx — to the first command's handling routine
JP xxxx — to the second command's handling routine...
JP xxxx — to the last command's handling routine

Command Name Table:

name of first command	names may be up to 16 characters long, and may be any characters
name of second command...	except lower case, space or comma; the last character of each name
name of last command	must have bit 7 set (ie &80 added to it)
DEFB &00	this is the end of table byte

On entering an RSX or called routine:

- A holds the number of parameters passed
- IX points to the low byte of the last parameter (if any)
- IY points to ':' or 'end of line byte' after the RSX name or call in the BASIC line
- B holds 32 minus the number of parameters passed
- C holds the ROM select number (or &FF if the RSX or called routine is in RAM)
- DE holds the last parameter entered (if any), or the address of the Command Table entry for an RSX, or the called routine entry point

LAYOUT OF EVENT AND TICKER BLOCKS SET UP BY THE FIRMWARE:

Event block — see &AC1E (&AC38 for the 464) in the Memory Maps section for an example:

bytes 0 and 1	—	Chaining address	
byte 2	—	Count of events awaiting processing (the event is disarmed if bit 7 is set)	
byte 3	—	Event class (see KL INIT EVENT for more details)	from B
bytes 4 and 5	—	Address of routine to service the event	from DE
byte 6	—	ROM select address	from C
bytes 7+	—	area for use by the servicing routine if required	

On entry to the event servicing routine, if the event routine is at a far address then HL will hold the address of the event block +5; if the event is at a near address then DE will hold the address of the event block +6

Ticker block — see &AC42 (&AC5C for the 464) in the Memory Maps section for an example:

bytes 0 and 1	—	Ticker chaining address	
bytes 2 and 3	—	Tick count	from DE
bytes 4 and 5	—	Recharge count	from BC
bytes 6 to 12	—	Event block (as described above)	
bytes 13+	—	area for use by the servicing routine (if required)	

Frame flyback block:

bytes 0 and 1	—	Frame chaining address
bytes 2 to 8	—	Event block (as described above)
bytes 9+	—	area for use by the servicing routine (if required)

Fast ticker block:

bytes 0 and 1	—	Fast chaining address
bytes 2 to 8	—	Event block (as described above)
bytes 9+	—	area for use by the serving routine (if required)

157 &BCD7 KL NEW FRAME FLY

Action: Sets up a frame flyback event block which will be acted on whenever a frame flyback occurs
Entry: HL contains the address of the event block in the central 32K of RAM, B contains the event class, C contains the ROM select address (if any), and DE contains the address if the event routine
Exit: AF, DE and HL are corrupt, and all other registers are preserved

158 &BCDA KL ADD FRAME FLY

Action: Adds an existing but deleted frame flyback event block to the list of routines run when a frame flyback occurs
Entry: HL contains the address of the event block (in the central 32K of RAM)
Exit: AF, DE and HL are corrupt, and all others are preserved

159 &BCDD KL DEL FRAME FLY
Action: Removes a frame flyback event block from the list of routines which are run when a frame flyback occurs
Entry: HL contains the address of the event block
Exit: AF, DE and HL are corrupt, and all others are preserved

160 &BCE0 KL NEW FAST TICKER
Action: Sets up a fast ticker event block which will be run whenever the 1/300th second ticker interrupt occurs
Entry: HL contains the address of the event block (in the central 32K of RAM), B contains the event class, C contains the ROM select address (if any), and DE contains the address of the event routine
Exit: AF, DE and HL are corrupt, and all other registers are preserved

161 &BCE3 KL ADD FAST TICKER
Action: Adds an existing but deleted fast ticker event block to the list of routines which are run when the 1/300th sec ticker interrupt occurs
Entry: HL contains the address of the event block
Exit: AF, DE and HL are corrupt, and all the other registers are preserved

162 &BCE6 KL DEL FAST TICKER
Action: Removes a fast ticker event block from the list of routines run when the 1/300th sec ticker interrupt occurs
Entry: HL contains the address of the event block
Exit: AF, DE and HL are corrupt, and all others are preserved

163 &BCE9 KL ADD TICKER
Action: Sets up a ticker event block which will be run whenever a 1/50th second ticker interrupt occurs
Entry: HL contains the address of the event block (in the central 32K of RAM), DE contains the initial value for the counter, and BC holds the value that the counter will be given whenever it reaches zero
Exit: AF, BC, DE and HL are corrupt, and all the other registers are preserved
Notes: Every 1/50th of a second all the tick blocks are looked at and their counter is decreased by 1; when the counter reaches zero, the event is 'kicked' and the counter is loaded with the value in BC; any tick block with a counter of 0 is ignored, and therefore if the value in BC is 0, the event will be kicked only once and ignored after that

164 &BCEC KL DEL TICKER
Action: Removes a ticker event block from the list of routines that are run when a 1/50th sec ticker interrupt occurs
Entry: HL contains the address of the event block
Exit: If the event block was found, then Carry is true, and DE holds the value remaining of the counter;
if the event block was not found, then Carry is false, and DE is corrupt;
in both cases, A, HL and the other flags are corrupt, and all other registers are preserved

165 &BCEF KL INIT EVENT
Action: Initialises an event block
Entry: HL contains the address of the event block (in the central 32K of RAM), B contains the class of event, and C contains the ROM select address, and DE holds the address of the event routine
Exit: HL holds the address of the event block+7, and all other registers are preserved
Notes: The event class is derived as follows:
- bit 0 — indicates a near address
- bits 1 to 4 — hold the synchronous event priority
- bit 5 — always zero
- bit 6 — if bit 6 is set, then it is an express event
- bit 7 — if bit 7 is set, then it is an asynchronous event

Asynchronous events do not have priorities; if it is an express asynchronous event, then its event routine is called from the interrupt path; if it is a normal asynchronous event, then its event routine is called just before returning from the interrupt; if it is an express synchronous event, then it has a higher priority than normal synchronous events, and it may not be disabled through use of KL EVENT DISABLE; if the near address bit is set, then the routine is located in the central 32K of RAM and is called directly, so saving time; no event may have a priority of zero

166 &BCF2 KL EVENT
Action: Kicks an event block
Entry: HL contains the address of the event block
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved

167 &BCF5 KL SYNC RESET

Action: Clears the synchronous event queue
Entry: No entry conditions
Exit: AF and HL are corrupt, and all other registers are preserved
Notes: When using this routine, all events that are waiting to be dealt with are simply discarded

168 &BCF8 KL DEL SYNCHRONOUS

Action: Removes a synchronous event from the event queue
Entry: HL contains the address of the event block
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved

169 &BCFB KL NEXT SYNC

Action: Finds out if there is a synchronous event with a higher priority
Entry: No entry conditions
Exit: If there is an event to be processed, then Carry is true, HL contains the address of the event block, and A contains the priority of the previous event;
if there is no event to be processed, then Carry is false, and A and HL are corrupt;
in either case, DE is corrupt, and all other registers are preserved

170 &BCFE KL DO SYNC

Action: Runs a synchronous event routine
Entry: HL contains the address of the event block
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved
Notes: See KL DONE SYNC below

171 &BD01 KL DONE SYNC

Action: Finishes running a synchronous event routine
Entry: A contains the priority of the previous event, and HL contains the address of the event block
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved
Notes: When an event that is waiting to be processed has been found by KL NEXT SYNC, the event routine should be run by KL DO SYNC; after this KL DONE SYNC should be called so that the event counter can be decreased — if the counter is greater than zero then the event is placed back on the synchronous event queue

172 &BD04 KL EVENT DISABLE

Action: Disables normal synchronous events
Entry: No entry conditions
Exit: HL is corrupt, and all other registers are preserved

173 &BD07 KL EVENT ENABLE

Action: Enables normal synchronous events
Entry: No entry conditions
Exit: HL is corrupt, and all other registers are preserved

174 &BD0A KL DISARM EVENT

Action: Disarms a specific event and stops it from occurring
Entry: HL contains the address of the event block
Exit: AF is corrupt, and all other registers are preserved
Notes: This routine should be used to disarm only *asynchronous* events; see also KL DEL SYNCHRONOUS

175 &BD0D KL TIME PLEASE

Action: Returns the time that has elapsed since the computer was switched on or reset (in 1/300ths of a second)
Entry: No entry conditions
Exit: DEHL contains the four byte count of the time elapsed, and all other registers are preserved
Notes: D holds the most significant byte of the time elapsed, and L holds the least significant ; the four byte count overflows after approximately 166 days have elapsed

176 &BD10 KL TIME SET

Action: Sets the elapsed time (in 1/300ths of a second)
Entry: DEHL contains the four byte count of the time to set
Exit: AF is corrupt, and all other registers are preserved

The Machine Pack

177 &BD13 MC BOOT PROGRAM

Action: Loads a program into RAM and then executes it

Entry: HL contains the address of the routine which is used to load the program

Exit: Control is handed over to the program and so the routine is not returned from

Notes: All events, sounds and interrupts are turned off, the firmware indirections are returned to their default settings, and the stack is reset; the routine to run the program should be in the central block of memory, and should obey the following exit conditions:

if the program was loaded successfully, then Carry is true, and HL contains the program entry point;
if the program failed to load, then Carry is false, and HL is corrupt;
in either case, A, BC, DE, IX, IY and the other flags are all corrupt

Should the program fail to load, control is returned to the previous foreground program

178 &BD16 MC START PROGRAM

Action: Runs a foreground program

Entry: HL contains the entry point for the program, and C contains the ROM selection number

Exit: Control is handed over to the program and so the routine is not returned from

179 &BD19 MC WAIT FLYBACK

Action: Waits until a frame flyback occurs

Entry: No entry conditions

Exit: All registers are preserved

Notes: When the frame flyback occurs the screen is not being written to and so the screen can be manipulated during this period without any flickering or ghosting on the screen

180 &BD1C MC SET MODE

Action: Sets the screen mode

Entry: A contains the required mode

Exit: AF is corrupt, and all other registers are preserved

Notes: Although this routine changes the screen mode it does not inform the routines which write to the screen that the mode has been changed; therefore these routines will write to the screen as if the mode had not been changed; however as the hardware is now interpreting these signals differently, unusual effects may occur

181 &BD1F MC SCREEN OFFSET

Action: Sets the screen offset

Entry: A contains the screen base, and HL contains the screen offset

Exit: AF is corrupt, and all other registers are preserved

Notes: As with MC SET MODE, this routine changes the hardware setting without telling the routines that write to the screen; therefore these routines may cause unpredictable effects if called; the default screen base is &C0

182 &BD22 MC CLEAR INKS

Action: Sets all the PENs and the border to one colour, so making it seem as if the screen has been cleared

Entry: DE contains the address of the ink vector

Exit: AF is corrupt, and all other registers are preserved

Notes: The ink vector takes the following form:

byte 0 — holds the colour for the border
byte 1 — holds the colour for all of the PENs

The values for the colours are all given as hardware values

183 &BD25 MC SET INKS

Action: Sets the colours of all the PENs and the border

Entry: DE contains the address of the ink vector

Exit: AF is corrupt, and all other registers are preserved

Notes: The ink vector takes the following form:

byte 0 — holds the colour for the border
byte 1 — holds the colour for PEN 0...
... byte 16 — holds the colour for PEN 15

The values for the colours are all given as hardware values; the routine sets all sixteen PENs

184 &BD28 MC RESET PRINTER
Action: Sets the MC WAIT PRINTER indirection to its original routine
Entry: No entry conditions
Exit: AF, BC, DE and HL are corrupt, and all others are preserved

185 &BD2B MC PRINT CHAR
Action: Sends a character to the printer and detects if it is busy for too long (more than 0.4 seconds)
Entry: A contains the character to be printed — only characters upto ASCII 127 can be printed
Exit: If the character was sent properly, then Carry is true;
if the printer was busy, then Carry is false;
in either case, A and the other flags are corrupt, and all other registers are preserved
Notes: This routine uses the MC WAIT PRINTER indirection

186 &BD2E MC BUSY PRINTER
Action: Tests to see if the printer is busy
Entry: No entry conditions
Exit: If the printer is busy, then Carry is true;
if the printer is not busy, then Carry is false;
in both cases, the other flags are corrupt, and all other registers are preserved

187 &BD31 MC SEND PRINTER
Action: Sends a character to the printer, which must *not* be busy
Entry: A contains the character to be printed — only characters upto ASCII 127 can be printed
Exit: Carry is true, A and the other flags are corrupt, and all other registers are preserved

188 &BD34 MC SOUND REGISTER
Action: Sends data to a sound chip register
Entry: A contains the register number, and C contains the data to be sent
Exit: AF and BC are corrupt, and all other registers are preserved

189 &BD37 JUMP RESTORE
Action: Restores the jumpblock to its default state
Entry: No entry conditions
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved
Notes: This routine does not affect the indirections jumpblock, but restores all entries in the main jumpblock

664 and 6128 only

190 &BD3A KM SET LOCKS
Action: Turns the shift and caps locks on and off
Entry: H contains the caps lock state, and L contains the shift lock state
Exit: AF is corrupt, and all others are preserved
Notes: In this routine, &00 means turned off, and &FF means turned on

191 &BD3D KM FLUSH
Action: Empties the key buffer
Entry: No entry conditions
Exit: AF is corrupt, and all other registers are preserved
Notes: This routine also discards any current expansion string

192 &BD40 TXT ASK STATE
Action: Gets the VDU and cursor state
Entry: No entry conditions
Exit: A contains the VDU and cursor state, the flags are corrupt, and all others are preserved
Notes: The value in the A register is bit significant, as follows:
if bit 0 is set, then the cursor is disabled, otherwise it is enabled
if bit 1 is set, then the cursor is turned off, otherwise it is on
if bit 7 is set, then the VDU is enabled, otherwise it is disabled

193 &BD43 GRA DEFAULT
Action: Sets the graphics VDU to its default mode
Entry: No entry conditions
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved
Notes: Sets the background to opaque, the first point of line is plotted, lines aren't dotted, and the write mode is force

194 &BD46 GRA SET BACK
Action: Sets the graphics background mode to either opaque or transparent
Entry: A holds zero if opaque mode is wanted, or holds &FF to select transparent mode
Exit: All registers are preserved

195 &BD49 GRA SET FIRST
Action: Sets whether the first point of a line is plotted or not
Entry: A holds zero if the first point is not to be plotted, or holds &FF if it is to be plotted
Exit: All registers are preserved

196 &BD4C GRA SET LINE MASK
Action: Sets how the points in a line are plotted — ie defines whether a line is dotted or not
Entry: A contains the line mask that will be used when drawing lines
Exit: All registers are preserved
Notes: The first point in the line corresponds to bit 7 of the line mask and after bit 0 the mask repeats; if a bit is set then that point will be plotted; the mask is always applied from left to right, or from bottom to top

197 &BD4F GRA FROM USER
Action: Converts user coordinates into base coordinates
Entry: DE contains the user X coordinate, and HL contains the user Y coordinate
Exit: DE holds the base X coordinate, and HL holds the base Y coordinate, AF is corrupt, and all others are preserved

198 &BD52 GRA FILL
Action: Fills an area of the screen starting from the current graphics position and extending until it reaches either the edge of the window or a pixel set to the PEN
Entry: A holds a PEN to fill with, HL holds the address of the buffer, and DE holds the length of the buffer
Exit: If the area was filled properly, then Carry is true;
if the area was not filled, then Carry is false;
in either case, A, BC, DE, HL and the other flags are corrupt, and all others are preserved
Notes: The buffer is used to store complex areas to fill, which are remembered and filled when the basic shape has been done; each entry in the buffer uses seven bytes and so the more complex the shape the larger the buffer; if it runs out of space to store these complex areas, it will fill what it can and then return with Carry false

199 &BD55 SCR SET POSITION
Action: Sets the screen base and offset without telling the hardware
Entry: A contains the screen base, and HL contains the screen offset
Exit: A contains the masked screen base, and HL contains the masked screen offset, the flags are corrupt, and all other registers are preserved

200 &BD58 MC PRINT TRANSLATION
Action: Sets how ASCII characters will be translated before being sent to the printer
Entry: HL contains the address of the table
Exit: If the table is too long, then Carry is false (ie more than 20 entries);
if the table is correctly set out, then Carry is true;
in either case, A, BC, DE, HL and the other flags are corrupt, and all others are preserved
Notes: The first byte in the table is the number of entries; each entry requires two bytes, as follows:
byte 0 — the character to be translated
byte 1 — the character that is to be sent to the printer
If the character to be sent to the printer is &FF, then the character is ignored and nothing is sent

201 &BD5B KL BANK SWITCH (6128 only)
Action: Sets which RAM banks are being accessed by the Z80
Entry: A contains the organisation that is to be used
Exit: A contains the previous organisation, the flags are corrupt, and all other registers are preserved

The Firmware Indirections

000 &BDCD TXT DRAW CURSOR
Action: Places the cursor on the screen, if the cursor is enabled
Entry: No entry conditions
Exit: AF is corrupt, and all other registers are preserved
Notes: The cursor is an inverse blob which appears at the current text position

001 &BDD0 TXT UNDRAW CURSOR
Action: Removes the cursor from the screen, if the cursor is enabled
Entry: No entry conditions
Exit: AF is corrupt, and all the other registers are preserved

002 &BDD3 TXT WRITE CHAR
Action: Writes a character onto the screen
Entry: A holds the character to be written, H holds the physical column number, and L holds the physical line number
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved

003 &BDD6 TXT UNWRITE
Action: Reads a character from the screen
Entry: H contains the physical column number, and L contains the physical line number to read from
Exit: If a character was found, then Carry is true, and A contains the character;
if no character was found, then Carry is false, and A contains zero;
in either case, BC, DE, HL and the other flags are corrupt, and all other registers are preserved
Notes: This routine works by comparing the image on the screen with the character matrices; therefore if the character matrices have been altered the routine may not find a readable a character

004 &BDD9 TXT OUT ACTION
Action: Writes a character to the screen or obeys a control code (&00 to &1F)
Entry: A contains the character or code
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved
Notes: Control codes may take a maximum of nine parameters; when a control code is found, the required number of parameters is read into the control code buffer, and then the control code is acted upon; if the graphics character write mode is enabled, then characters and codes are printed using the graphics VDU; when using the graphics VDU control codes are printed and not obeyed

005 &BDDC GRA PLOT
Action: Plots a point in the current graphics PEN
Entry: DE contains the user X coordinate, and HL contains the user Y coordinate of the point
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved
Notes: This routine uses the SCR WRITE indirection to write the point to the screen

006 &BDDF GRA TEST
Action: Tests a point and finds out what PEN it is set to
Entry: DE contains the user X coordinate, and HL contains the user Y coordinate of the point
Exit: A contains the PEN that the point is written in, BC, DE and HL are corrupt, and all others are preserved
Notes: This routine uses the SCR READ indirection to test a point on the screen

007 &BDE2 GRA LINE
Action: Draws a line in the current graphics PEN, from the current graphics position to the specified point
Entry: DE contains the user X coordinate, and HL contains the user Y coordinate for the endpoint
Exit: AF, BC, DE and HL are corrupt, and all others are preserved
Notes: This routine uses the SCR WRITE indirection to write the points of the line on the screen

008 &BDE5 SCR READ
Action: Reads a pixel from the screen and returns its decode a PEN
Entry: HL contains the screen address of the pixel, and C contains the mask for the pixel
Exit: A contains the decoded PEN of the pixel, the flags are corrupt, and all others are preserved
Notes: The mask should be for a single pixel, and is dependent on the screen mode

009 &BDE8 SCR WRITE
Action: Writes one or more pixels to the screen
Entry: HL contains the screen address of the pixel, C contains the mask, and B contains the encoded PEN
Exit: AF is corrupt, and all other registers are preserved
Notes: The mask should determine which pixels in the screen byte are to be plotted

010 &BDEB SCR MODE CLEAR
Action: Fills the entire screen memory with &00, which clears the screen to PEN 0
Entry: No entry conditions
Exit: AF, BC, DE and HL are corrupt, and all the other registers are preserved

011 &BDEE KM TEST BREAK
Action: Tests if the ESC key has been pressed, and acts accordingly
Entry: C contains the Shift and Control key states, and interrupts must be disabled
Exit: AF and HL are corrupt, and all other registers are preserved
Notes: If bit 7 of C is set, then the Control key is pressed; if bit 5 of C is set, then the Shift key is pressed; if ESC, Shift and Control are pressed at the same time, then it initiates a system reset; otherwise it reports a break event

012 &BDF1 MC WAIT PRINTER
Action: Sends a character to the printer if it is not busy
Entry: A contains the character to be sent to the printer
Exit: If the character was printed successfully, then Carry is true;
if the printer was busy for too long (more than 0.4 seconds), then Carry is false;
in either case, A and BC are corrupt, and all other registers are preserved

013 &BDF4 KM SCAN KEYS
Action: Scans the keyboard every 1/50th of a second, and updates the status of all keys
Entry: All interrupts must be disabled
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved

The Maths Firmware

000 &BD61 MOVE REAL (&BD3D for the 464)
Action: Copies the five bytes that are pointed to by DE to the location held in HL
Entry: DE points to the source real value, and HL points to the destination
Exit: HL points to the real value in the destination, Carry is true if the move went properly, F is corrupt, and all other registers are preserved
Notes: For the 464 only, A holds the exponent byte of the real value when the routine is exited

001 &BD64 INTEGER TO REAL (&BD40 for the 464)
Action: Converts an integer value into a real value
Entry: HL holds the integer value, DE points to the destination for the real value, bit 7 of A holds the sign of the integer value — it is taken to be negative if bit 7 is set
Exit: HL points to the real value in the destination, AF and DE are corrupt, and all others are preserved

002 &BD67 BINARY TO REAL (&BD43 for the 464)
Action: Converts a four byte binary value into a real value at the same location
Entry: HL points to the binary value, bit 7 of A holds the sign of the binary value — negative if it is set
Exit: HL points to the real value in lieu of the four byte binary value, AF is corrupt, and all others are preserved
Notes: A four byte binary value is an unsigned integer up to &FFFFFFFF and is stored with the least significant byte first, and with the most significant byte last

003 &BD6A REAL TO INTEGER (&BD46 for the 464)
Action: Converts a real value, rounding it into an unsigned integer value held in HL
Entry: HL points to the real value
Exit: HL holds the integer value, Carry is true if the conversion worked successfully, the Sign flag holds the sign of the integer (negative if it is set), A, IX and the other flags are corrupt, and all other registers are preserved
Notes: This rounds the decimal part down if it is less than 0.5, but rounds up if it is greater than, or equal to 0.5

004 &BD6D REAL TO BINARY (&BD49 for the 464)

Action: Converts a real value, rounding it into a four byte binary value at the same location
Entry: HL points to the real value
Exit: HL points to the binary value in lieu of the real value, bit 7 of B holds the sign for the binary value (it is negative if bit 7 is set), AF, B and IX are corrupt, and all other registers are preserved
Notes: See REAL TO INTEGER for details of how the values are rounded up or down

005 &BD70 REAL FIX (&BD4C for the 464)

Action: Performs an equivalent of BASIC's FIX function on a real value, leaving the result as a four bye binary value at the same location
Entry: HL points to the real value
Exit: HL points to the binary value in lieu of the real value, bit 7 of B has the sign of the binary value (it is negative if bit 7 is set), AF, B and IX are corrupt, and all others are preserved
Notes: FIX removes any decimal part of the value, rounding down whether positive or negative — see the BASIC handbook for more details on the FIX command

006 &BD73 REAL INT (&BD4F for the 464)

Action: Performs an equivalent of BASIC's INT function on a real value, leaving the result as a four byte binary value at the same location
Entry: HL points to the real value
Exit: HL points to the binary value in lieu of the real value, bit 7 of B has the sign of the binary value (it is negative if bit 7 is set), AF, B and IX are corrupt, and all others are preserved
Notes: INT removes any decimal part of the value, rounding down if the number is positive, but rounding up if it is negative

007 &BD76 INTERNAL SUBROUTINE — not useful (&BD52 for the 464)

008 &BD79 REAL *10^A (&BD55 for the 464)

Action: Multiplies a real value by '10 to the power of the value in the A register', leaving the result at the same location
Entry: HL points to the real value, and A holds the power of 10
Exit: HL points to the result, AF, BC, DE, IX and IY are corrupt

009 &BD7C REAL ADDITION (&BD58 for the 464)

Action: Adds two real values, and leaves the result in lieu of the first real number
Entry: HL points to the first real value, and DE points to the second real value
Exit: HL points to the result, AF, BC, DE, IX and IY are corrupt

010 &BD82 REAL REVERSE SUBTRACTION (&BD5E for the 464)

Action: Subtracts the first real value from the second real value, and leaves the result in lieu of the first number
Entry: HL points to the first real value, and DE points to the second real value
Exit: HL points to the result in place of the first real value, AF, BC, DE, IX and IY are corrupt

011 &BD85 REAL MULTIPLICATION (&BD61 for the 464)

Action: Multiplies two real values together, and leaves the result in lieu of the first number
Entry: HL points to the first real value, and DE points to the second real value
Exit: HL points to the result in place of the first real value, AF, BC, DE, IX and IY are corrupt

012 &BD88 REAL DIVISION (&BD64 for the 464)

Action: Divides the first real value by the second real value, and leaves the result in lieu of the first number
Entry: HL points to the first real value, and DE points to the second real value
Exit: HL points to the result in place of the first real value, AF, BC, DE, IX and IY are corrupt

013 &BD8E REAL COMPARISON (&BD6A for the 464)

Action: Compares two real values
Entry: HL points to the first real value, and DE points to the second real value
Exit: A holds the result of the comparison process, IX, IY and the other flags are corrupt, and all others are preserved
Notes: After this routine has been called, the value in A depends on the result of the comparison as follows:
if the first real number is greater than the second real number, then A holds &01
if the first real number is the same as the second real number, then A holds &00
if the second real number is greater than the first real number, then A holds &FF

014 &BD91 REAL UNARY MINUS (&BD6D for the 464)
Action: Reverses the sign of a real value
Entry: HL points to the real value
Exit: HL points to the new value of the real number (which is stored in place of the original number), bit 7 of A holds the sign of the result (it is negative if bit 7 is set), AF and IX are corrupt, and all other registers are preserved

015 &BD94 REAL SIGNUM/SGN (&BD70 for the 464)
Action: Tests a real value, and compares it with zero
Entry: HL points to the real value
Exit: A holds the result of this comparison process, IX and the other flags are corrupt, and all others are preserved
Notes: After this routine has been called, the value in A depends on the result of the comparison as follows:
if the real number is greater than 0, then A holds &01, Carry is false, and Zero is false
if the real number is the same as 0, then A holds &00, Carry is false, and Zero is true
if the real number is smaller than 0, then A holds &FF, Carry is true, and Zero is false

016 &BD97 SET ANGLE MODE (&BD73 for the 464)
Action: Sets the angular calculation mode to either degrees (DEG) or radians (RAD)
Entry: A holds the mode setting — 0 for RAD, and any other value for DEG
Exit: All registers are preserved

017 &BD9A REAL PI (&BD76 for the 464)
Action: Places the real value of pi at a given memory location
Entry: HL holds the address at which the value of pi is to be placed
Exit: AF and DE are corrupt, and all other registers are preserved

018 &BD9D REAL SQR (&BD79 for the 464)
Action: Calculates the square root of a real value, leaving the result in lieu of the real value
Entry: HL points to the real value
Exit: HL points to the result of the calculation, AF, BC, DE, IX and IY are corrupt

019 &BDA0 REAL POWER (&BD7C for the 464)
Action: Raises the first real value to the power of the second real value, leaving the result in lieu of the first real value
Entry: HL points to the first real value, and DE points to the second real value
Exit: HL points to the result of the calculation, AF, BC, DE, IX and IY are corrupt

020 &BDA3 REAL LOG (&BD7F for the 464)
Action: Returns the naperian logarithm (to base e) of a real value, leaving the result in lieu of the real value
Entry: HL points to the real value
Exit: HL points to the logarithm that has been calculated, AF, BC, DE, IX and IY are corrupt

021 &BDA6 REAL LOG 10 (&BD82 for the 464)
Action: Returns the logarithm (to base 10) of a real value, leaving the result in lieu of the real value
Entry: HL points to the real value
Exit: HL points to the logarithm that has been calculated, AF, BC, DE, IX and IY are corrupt

022 &BDA9 REAL EXP (&BD85 for the 464)
Action: Returns the antilogarithm (base e) of a real value, leaving the result in lieu of the real value
Entry: HL points to the real value
Exit: HL points to the antilogarithm that has been calculated, AF, BC, DE, IX and IY are corrupt
Notes: See the BASIC handbook for details of EXP

023 &BDAC REAL SINE (&BD88 for the 464)
Action: Returns the sine of a real value, leaving the result in lieu of the real value
Entry: HL points to the real value (ie an angle)
Exit: HL points to the sine value that has been calculated, AF, BC, DE, IX and IY are corrupt

024 &BDAF REAL COSINE (&BD8B for the 464)
Action: Returns the cosine of a real value, leaving the result in lieu of the real value
Entry: HL points to the real value (ie an angle)
Exit: HL points to the cosine value that has been calculated, AF, BC, DE, IX and IY are corrupt

025 **&BDB2 REAL TANGENT (&BD8E for the 464)**
Action: Returns the tangent of a real value, leaving the result in lieu of the real value
Entry: HL points to the real value (ie an angle)
Exit: HL points to the tangent value that has been calculated, AF, BC, DE, IX and IY are corrupt

026 **&BDB5 REAL ARCTANGENT (&BD91 for the 464)**
Action: Returns the arctangent of a real value, leaving the result in lieu of the real value
Entry: HL points to the real value (ie an angle)
Exit: HL points to the arctangent value that has been calculated, AF, BC, DE, IX and IY are corrupt

All of the above routines to calculate sine, cosine, tangent and arctangent are slightly inaccuarate

027 **&BDB8 INTERNAL SUBROUTINE** — not useful (&BD94 for the 464)

028 **&BDBB INTERNAL SUBROUTINE** — not useful (&BD97 for the 464)

029 **&BDBE INTERNAL SUBROUTINE** — not useful (&BD9A for the 464)

Maths Routines for the 464 only

&BD5B REAL SUBTRACTION
Action: Subtracts the second real value from the first real value, and leaves the result in lieu of the first number
Entry: HL points to the first real value, and DE points to the second real value
Exit: HL points to the result in place of the first real value, AF, BC, DE, IX and IY are corrupt

&BD67 REAL EXPONENT ADDITION
Action: Adds the value of the A register to the exponent byte of a real number
Entry: HL points to the real value, and A holds the value to be added
Exit: HL points to the result in place of the first real value, AF and IX are corrupt, and all others are preserved

&BD9D INTERNAL SUBROUTINE — not useful

&BDA0 INTERNAL SUBROUTINE — not useful

&BDA3 INTERNAL SUBROUTINE — not useful

&BDA6 INTERNAL SUBROUTINE — not useful

&BDA9 INTERNAL SUBROUTINE — not useful

&BDAC INTEGER ADDITION
Action: Adds two signed integer values
Entry: HL holds the first integer value, and DE holds the second integer value
Exit: HL holds the result of the addition, A holds &FF if there is an overflow but is preserved otherwise, the flags are corrupt, and all other registers are preserved

&BDAF INTEGER SUBTRACTION
Action: Subtracts the second signed integer value from the first signed integer value
Entry: HL holds the first integer value, and DE holds the second integer value
Exit: HL holds the result of the subtraction, A holds &FF if there is an overflow but is preserved otherwise, the flags are corrupt, and all the other registers are preserved

&BDB2 INTEGER REVERSE SUBTRACTION
Action: Subtracts the first signed integer value from the second signed integer value
Entry: HL holds the first integer value, and DE holds the second integer value
Exit: HL holds the result of the subtraction, AF and DE are corrupt, and all others are preserved

&BDB5 INTEGER MULTIPLICATION
Action: Multiplies two signed integer values together, and leaves the result in lieu of the first number
Entry: HL holds the first integer value, and DE holds the second integer value
Exit: HL holds the result of the multiplication, A holds &FF if there is an overflow but is corrupted otherwise, the flags, BC and DE are corrupt, and the other registers are preserved
Notes: Multiplication of signed integers does not produce the same result as with unsigned integers

&BDB8 INTEGER DIVISION
Action: Divides the first signed integer value by the second signed integer value
Entry: HL holds the first integer value, and DE holds the second integer value
Exit: HL holds the result of the division, DE holds the remainder, AF and BC are corrupt, and all others are preserved
Notes: Division of signed integers does not produce the same result as with unsigned integers

&BDBB INTEGER DIVISION 2
Action: Divides the first signed integer value by the second signed integer value
Entry: HL holds the first integer value, and DE holds the second integer value
Exit: DE holds the result of the division, HL holds the remainder, AF and BC are corrupt, and all others are preserved
Notes: Division of signed integers does not produce the same result as with unsigned integers

&BDBE INTERNAL SUBROUTINE — not useful

&BDC1 INTERNAL SUBROUTINE — not useful

&BDC4 INTEGER COMPARISON
Action: Compares two signed integer values
Entry: HL holds the first integer value, and DE holds the second integer value
Exit: A holds the result of the comparison process, the flags are corrupt, and all others are preserved
Notes: After this routine has been called, the value in A depends on the result of the comparison as follows:
if the first real number is greater than the second real number, then A holds &01
if the first real number is the same as the second real number, then A holds &00
if the second real number is greater than the first real number, then A holds &FF
With signed integers, the range of values runs from &8000 (-32768) via zero to &7FFF (+32767) and so any value which is greater than &8000 is considered as being less than a value of &7FFF or less

&BDC7 INTEGER UNARY MINUS
Action: Reverses the sign of an integer value (by subtracting it from &10000)
Entry: HL holds the integer value
Exit: HL holds the new value of the integer number, AF is corrupt, and all other registers are preserved

&BDCA INTEGER SIGNUM/SGN
Action: Tests a signed integer value
Entry: HL holds the integer value
Exit: A holds the result of this comparison process, the flags are corrupt, and all others are preserved
Notes: After this routine has been called, the value in A depends on the result of the comparison as follows:
if the integer number is greater than 0 and is less than &8000, then A holds &01
if the integer number is the same as 0, then A holds &00
if the integer number is greater than &7FFF and less than or equal to &FFFF, then A holds &FF
See INTEGER COMPARISON for more details on the way that signed integers are laid out

Maths Subroutines for the 664 and 6128 only

&BD5E TEXT INPUT
Action: Allows upto 255 characters to be input from the keyboard into a buffer
Entry: HL points to the start of the buffer — a NUL character must be placed after any characters already present, or at the start of the buffer if there is no text
Exit: A has the last key pressed, HL points to the start of the buffer, the flags are corrupt, and all others are preserved
Notes: This routine prints any existing contents of the buffer (upto the NUL character) and then echoes any keys used; it allows full line editing with the cursor keys and DEL, etc; it is exited only by use of ENTER or ESC

&BD7F REAL RND
Action: Creates a new RND real value at a location pointed to by HL
Entry: HL points to the destination for the result
Exit: HL points to the RND value, AF, BC, DE and IX registers are corrupt; and all others are preserved

&BD8B REAL RND(0)
Action: Returns the last RND value created, and puts it in a location pointed to by HL
Entry: HL points to the place where the value is to be returned to
Exit: HL points to the value created, AF, DE and IX are corrupt, and all other registers are preserved
Notes: See the BASIC handbook for more details on RND(0)

AMSDOS and BIOS Firmware

A **&C033 BIOS SET MESSAGE**
Action: Enables or disables disc error messages
Entry: To enable messages, A holds 00; to disable messages, A holds &FF
Exit: A holds the previous state, HL and the flags are corrupt, and all others are preserved
Notes: Enabling and disabling the messages can also be achieved by poking &BE78 with &00 or &FF

B **&C036 BIOS SETUP DISC**
Action: Sets the parameters which effect the disc speed
Entry: HL holds the address of the nine bytes which make up the parameter block
Exit: AF, BC, DE and HL are corrupt, and all other registers are preserved
Notes: The parameter block is arranged as follows:

bytes 0 & 1	— the motor on time in 20mS units;	the default is &0032;	the fastest is &0023
bytes 2 & 3	— the motor off time in 20mS units;	the default is &00FA;	the fastest is &00C8
byte 4	— the write off time in 10uS units;	the default is &AF;	should not be changed
byte 5	— the head settle time in 1mS units;	the default is &0F;	should not be changed
byte 6	— the step rate time in 1mS units;	the default is &0C;	the fastest is &0A
byte 7	— the head unload delay;	the default is &01;	should not be changed
byte 8	— a byte of &03 and this should be left unaltered		

C **&C039 BIOS SELECT FORMAT**
Action: Sets a format for a disc
Entry: A holds the type of format that is to be selected
Exit: AF, BC, DE and HL are corrupt, and all the other registers are preserved
Notes: To select one of the normal disc formats, the following values should be put into the A register:
Data format — &C1
System format — &41
IBM format — &01
This routine sets the extended disc parameter block (XDPB) at &A890 to &A8A8 — to set other formats, the XDPB must be altered directly

D **&C03C BIOS READ SECTOR**
Action: Reads a sector from a disc into memory
Entry: HL holds the address in memory where the sector will be read to, E holds the drive number (&00 for drive A, and &01 for drive B), D holds the track number, and C holds the sector number
Exit: If the sector was read properly, then Carry is true, A holds 0, and HL is preserved;
if the read failed, then Carry is false, A holds an error number, and HL is corrupt;
in either case, the other flags are corrupt, and all other registers are preserved

E **&C03F BIOS WRITE SECTOR**
Action: Writes a sector from memory onto disc
Entry: HL holds the address of memory which will be written to the disc, E holds the drive number (&00 for drive A, and &01 for drive B), D holds the track number, and C holds the sector number
Exit: If the sector was written properly, then Carry is true, A holds 0, and HL is preserved;
if the write failed, then Carry is false, A holds an error number, and HL is corrupt;
in either case, the other flags are corrupt, and all other registers are preserved

F &C042 BIOS FORMAT TRACK

Action: Formats a complete track, inserts sectors, and fills the track with bytes of &E5

Entry: HL contains the address of the header information buffer which holds the header information blocks, E contains the drive number (&00 for drive A, and &01 for drive B), and D holds the track number

Exit: if the formatting process was successful, then Carry is true, A holds 0, and HL is preserved;
if the formatting process failed, then Carry is false, A holds an error number, and HL is corrupt;
in either case, the other flags are corrupt, and all the other registers are preserved

Notes: The header information block is laid out as follows:
byte 0 — holds the track number
byte 1 — holds the head number (set to zero)
byte 2 — holds the sector number
byte 3 — holds $\log_2$(sector size)–7 (usually either &02=512 bytes, or &03=1024 bytes)
Header information blocks must be set up contiguously for every sector on the track, and in the same sequence that they are to be laid down (eg &C1, &C6, &C2, &C7, &C3, &C8, &C4, &C9, &C5)

G &C045 BIOS MOVE TRACK

Action: Moves the disc drive head to the specified track

Entry: E holds the drive number (&00 for drive A, and &01 for drive B), and D holds the track number

Exit: If the head was moved successfully, then Carry is true, A holds 0, and HL is preserved;
if the move failed, then Carry is false, A holds an error number, and HL is corrupt;
in both cases, the other flags are corrupt, and all other registers are preserved

Notes: There is normally no need to call this routine as READ SECTOR, WRITE SECTOR and FORMAT TRACK automatically move the head to the correct position

H &C048 BIOS GET STATUS

Action: Returns the status of the specified drive

Entry: A holds the drive number (&00 for drive A, and &01 for drive B)

Exit: If Carry is true, then A holds the status byte, and HL is preserved;
if Carry is false, then A is corrupt, and HL holds the address of the byte before the status byte;
in either case, the other flags are preserved, and all other registers are preserved

Notes: The status byte indicates the drive's status as follows:
if bit 6 is set, then either the write protect is set or the disc is missing
if bit 5 is set, then the drive is ready and the disc is fitted (whether the disc is formatted or not)
if bit 4 is set, then the head is at track 0

I &C04B BIOS SET RETRY COUNT

Action: Sets the number of times the operation is retried in the event of disc error

Entry: A holds the number of retries required

Exit: A holds the previous number of retries, HL and the flags are corrupt, and all others are preserved

Notes: The default setting is &10, and the minimum setting is &01; the number of retries can also be altered by poking &BE66 with the required value

&C56C GET SECTOR DATA

Action: Gets the data of a sector on the current track

Entry: E holds the drive number

Exit: If a formatted disc is present, then Carry is true, and HL is preserved;
if an unformatted disc is present or the disc is missing, then Carry is false, and HL holds the address of the byte before the status byte;
in either case, A and the other flags are corrupt, and all other registers are preserved

Notes: The track number is held at &BE4F, the head number is held as &BE50, the sector number is held at &BE51, and the $\log_2$(sector size)–7 is held at &BE52; disc parameters do not need to be set to the format of the disc; this routine is best used with the disc error messages turned off

Index to information boxes

Instruction Set for the Z80 Microprocessor

The following list contains all the normal machine code instructions for the microprocessor, plus a number of undocumented ones. The latter comprise those which operate on the high or low bytes of the Index registers (IX and IY) which are notated here as HIX, LIX, HIY and LIY — some assemblers may use the form IXH, etc — and a set of rotation instructions complementary to SRL, which are designated SLL.

The Opcodes and T states

Within the instructions, a number of abbreviations are used:

d	= displacement (a value from –128 (&80) to +127 (&7F))
n	= a single byte value (from 0 (&00) to 255 (&FF))
hilo	= a double byte value (from –32769 (&8000) via 0 to 65535 (&FFFF))
addr	= an address value (from 0 (&0000) to 65535 (&FFFF))

(in the sequence of opcode bytes, 'addr' and 'hilo' are entered low byte first)

The next two columns detail the number of bytes applicable to each instruction, and the number of T states (clock pulses) that each requires — some have two figures which are distinguised as follows:

f	— means 'the number of T states required when the condition is false'
t	— means 'the number of T states needed when the condition is true'
=	— means 'the number of T states needed when either BC=0 and/or A matches the contents of HL'
#	— means 'the number of T states required when both the above conditions are false'
z	— means 'the number of T states needed when B=0'
nz	— means 'the number of T states required when B<>0'

The Flag Register

The last columns give the effect on the flag bits which each instructions causes:

?	— means the setting of the bit is unpredicatable
–	— means the setting of the bit is unchanged
0	— means that the flag bit is reset to zero
1	— means that the flag bit is set to one

In addition, the Sign flag (bit 7) is also set:

7	— if bit 7 of the A register is set
15	— if bit 15 of the HL register pair (ie bit 7 of the H register) is set
=7	— if bit 7 of the A register would be set by subtraction in lieu of CP

The Zero flag (bit 6) is also set:

z	— if the A register or the HL register pair equals zero
=	— if the A register matches the compared register or value
=A	— if the A register matches the contents of the address pointed to by HL
<>B	— if the B register holds zero
<>b	— if the bit tested is zero

The Parity/Overflow flag (bit 2) is also set:

p	— if the register concerned contains an even number of set bits
v	— if an overflow has occured in Two's Complement arithmetic
BC	— if BC is not zero
A80	— if the A register was &80 before this instruction was performed
i	— to the contents of the microprocessor's internal interrupt register

The Carry flag (bit 0) is also set:

c	— if an addition resulted in a carry out of bit 7 (for a register) or bit 15 (for a register pair)
b	— if a subtraction required a borrow from bit 7 (for a register) or bit 15 (for a register pair)
<	— if the A register is less than the value or register that is being compared
r0	— by the bit rotated in from bit 0 of the register concerned
r7	— by the bit rotated in from bit 7 of the register concerned
x	— if the Carry was reset (ie zero) before this instruction was performed
A0	— if the A register was &00 before this instruction was performed

Instruction	Opcode	Bytes	Ts	S	Z	P	C
ADC A,n	CE n	2	7	7	z	v	c
ADC A,A	8F	1	4	7	z	v	c
ADC A,B	88	1	4	7	z	v	c
ADC A,C	89	1	4	7	z	v	c
ADC A,D	8A	1	4	7	z	v	c
ADC A,E	8B	1	4	7	z	v	c
ADC A,H	8C	1	4	7	z	v	c
ADC A,L	8D	1	4	7	z	v	c
ADC A,(HL)	8E	1	7	7	z	v	c
ADC A,(IX+d)	DD 8E d	3	19	7	z	v	c
ADC A,(IY+d)	FD 8E d	3	19	7	z	v	c
ADC A,HIX	DD 8C	2	8	7	z	v	c
ADC A,LIX	DD 8D	2	8	7	z	v	c
ADC A,HIY	FD 8C	2	8	7	z	v	c
ADC A,LIY	FD 8D	2	8	7	z	v	c
ADC HL,BC	ED 4A	2	15	15	z	v	c
ADC HL,DE	ED 5A	2	15	15	z	v	c
ADC HL,HL	ED 6A	2	15	15	z	v	c
ADC HL,SP	ED 7A	2	15	15	z	v	c
ADD A,n	C6 n	2	7	7	z	v	c
ADD A,A	87	1	4	7	z	v	c
ADD A,B	80	1	4	7	z	v	c
ADD A,C	81	1	4	7	z	v	c
ADD A,D	82	1	4	7	z	v	c
ADD A,E	83	1	4	7	z	v	c
ADD A,H	84	1	4	7	z	v	c
ADD A,L	85	1	4	7	z	v	c
ADD A,(HL)	86	1	7	7	z	v	c
ADD A,(IX+d)	DD 86 d	3	19	7	z	v	c
ADD A,(IY+d)	FD 86 d	3	19	7	z	v	c
ADD A,HIX	DD 86 84	2	8	7	z	v	c
ADD A,LIX	DD 86 85	2	8	7	z	v	c
ADD A,HIY	FD 86 84	2	8	7	z	v	c
ADD A,LIY	FD 86 85	2	8	7	z	v	c
ADD HL,BC	09	1	11	-	-	-	c
ADD HL,DE	19	1	11	-	-	-	c
ADD HL,HL	29	1	11	-	-	-	c
ADD HL,SP	39	1	11	-	-	-	c
ADD IX,BC	DD 09	2	15	-	-	-	c
ADD IX,DE	DD 19	2	15	-	-	-	c
ADD IX,HL	DD 29	2	15	-	-	-	c
ADD IX,SP	DD 39	2	15	-	-	-	c
ADD IY,BC	FD 09	2	15	-	-	-	c
ADD IY,DE	FD 19	2	15	-	-	-	c
ADD IY,HL	FD 29	2	15	-	-	-	c
ADD IY,SP	FD 39	2	15	-	-	-	c
AND n	E6 n	2	7	7	z	p	c
AND A	A7	1	4	7	z	p	c
AND B	A0	1	4	7	z	p	c
AND C	A1	1	4	7	z	p	c
AND D	A2	1	4	7	z	p	c
AND E	A3	1	4	7	z	p	c
AND H	A4	1	4	7	z	p	c
AND L	A5	1	4	7	z	p	c
AND (HL)	A6	1	7	7	z	p	c
AND (IX+d)	DD A6 d	3	19	7	z	p	c
AND (IY+d)	FD A6 d	3	19	7	z	p	c
AND HIX	DD A4	2	8	7	z	v	c

Instruction	Opcode	Bytes	Ts	S	Z	P	C
AND LIX	DD A5	2	8	7	z	v	c
AND HIY	FD A4	2	8	7	z	v	c
AND LIY	FD A5	2	8	7	z	v	c
BIT 0,A	CB 47	2	8	?	<>b	?	-
BIT 0,B	CB 40	2	8	?	<>b	?	-
BIT 0,C	CB 41	2	8	?	<>b	?	-
BIT 0,D	CB 42	2	8	?	<>b	?	-
BIT 0,E	CB 43	2	8	?	<>b	?	-
BIT 0,H	CB 44	2	8	?	<>b	?	-
BIT 0,L	CB 45	2	8	?	<>b	?	-
BIT 0,(HL)	CB 46	2	12	?	<>b	?	-
BIT 0,(IX+d)	CB DD 46	4	20	?	<>b	?	-
BIT 0,(IY+d)	CB FD 46	4	20	?	<>b	?	-
BIT 1,A	CB 4F	2	8	?	<>b	?	-
BIT 1,B	CB 48	2	8	?	<>b	?	-
BIT 1,C	CB 49	2	8	?	<>b	?	-
BIT 1,D	CB 4A	2	8	?	<>b	?	-
BIT 1,E	CB 4B	2	8	?	<>b	?	-
BIT 1,H	CB 4C	2	8	?	<>b	?	-
BIT 1,L	CB 4D	2	8	?	<>b	?	-
BIT 1,(HL)	CB 4E	2	12	?	<>b	?	-
BIT 1,(IX+d)	CB DD 4E	4	20	?	<>b	?	-
BIT 1,(IY+d)	CB FD 4E	4	20	?	<>b	?	-
BIT 2,A	CB 57	2	8	?	<>b	?	-
BIT 2,B	CB 50	2	8	?	<>b	?	-
BIT 2,C	CB 51	2	8	?	<>b	?	-
BIT 2,D	CB 52	2	8	?	<>b	?	-
BIT 2,E	CB 53	2	8	?	<>b	?	-
BIT 2,H	CB 54	2	8	?	<>b	?	-
BIT 2,L	CB 55	2	8	?	<>b	?	-
BIT 2,(HL)	CB 56	2	12	?	<>b	?	-
BIT 2,(IX+d)	CB DD 56	4	20	?	<>b	?	-
BIT 2,(IY+d)	CB FD 56	4	20	?	<>b	?	-
BIT 3,A	CB 5F	2	8	?	<>b	?	-
BIT 3,B	CB 58	2	8	?	<>b	?	-
BIT 3,C	CB 59	2	8	?	<>b	?	-
BIT 3,D	CB 5A	2	8	?	<>b	?	-
BIT 3,E	CB 5B	2	8	?	<>b	?	-
BIT 3,H	CB 5C	2	8	?	<>b	?	-
BIT 3,L	CB 5D	2	8	?	<>b	?	-
BIT 3,(HL)	CB 5E	2	12	?	<>b	?	-
BIT 3,(IX+d)	CB DD 5E	4	20	?	<>b	?	-
BIT 3,(IY+d)	CB FD 5E	4	20	?	<>b	?	-
BIT 4,A	CB 67	2	8	?	<>b	?	-
BIT 4,B	CB 60	2	8	?	<>b	?	-
BIT 4,C	CB 61	2	8	?	<>b	?	-
BIT 4,D	CB 62	2	8	?	<>b	?	-
BIT 4,E	CB 63	2	8	?	<>b	?	-
BIT 4,H	CB 64	2	8	?	<>b	?	-
BIT 4,L	CB 65	2	8	?	<>b	?	-
BIT 4,(HL)	CB 66	2	12	?	<>b	?	-
BIT 4,(IX+d)	CB DD 66	4	20	?	<>b	?	-
BIT 4,(IY+d)	CB FD 66	4	20	?	<>b	?	-
BIT 5,A	CB 6F	2	8	?	<>b	?	-
BIT 5,B	CB 68	2	8	?	<>b	?	-
BIT 5,C	CB 69	2	8	?	<>b	?	-
BIT 5,D	CB 6A	2	8	?	<>b	?	-
BIT 5,E	CB 6B	2	8	?	<>b	?	-

Instruction	Opcode	Bytes	Ts	S	Z	P	C
BIT 5,H	CB 6C	2	8	?	<>b	?	-
BIT 5,L	CB 6D	2	8	?	<>b	?	-
BIT 5,(HL)	CB 6E	2	12	?	<>b	?	-
BIT 5,(IX+d)	CB DD 6E	4	20	?	<>b	?	-
BIT 5,(IY+d)	CB FD 6E	4	20	?	<>b	?	-
BIT 6,A	CB 77	2	8	?	<>b	?	-
BIT 6,B	CB 70	2	8	?	<>b	?	-
BIT 6,C	CB 71	2	8	?	<>b	?	-
BIT 6,D	CB 72	2	8	?	<>b	?	-
BIT 6,E	CB 73	2	8	?	<>b	?	-
BIT 6,H	CB 74	2	8	?	<>b	?	-
BIT 6,L	CB 75	2	8	?	<>b	?	-
BIT 6,(HL)	CB 76	2	12	?	<>b	?	-
BIT 6,(IX+d)	CB DD 76	4	20	?	<>b	?	-
BIT 6,(IY+d)	CB FD 76	4	20	?	<>b	?	-
BIT 7,A	CB 7F	2	8	?	<>b	?	-
BIT 7,B	CB 78	2	8	?	<>b	?	-
BIT 7,C	CB 79	2	8	?	<>b	?	-
BIT 7,D	CB 7A	2	8	?	<>b	?	-
BIT 7,E	CB 7B	2	8	?	<>b	?	-
BIT 7,H	CB 7C	2	8	?	<>b	?	-
BIT 7,L	CB 7D	2	8	?	<>b	?	-
BIT 7,(HL)	CB 7E	2	12	?	<>b	?	-
BIT 7,(IX+d)	CB DD 7E	4	20	?	<>b	?	-
BIT 7,(IY+d)	CB FD 7E	4	20	?	<>b	?	-
CALL addr	CD dr ad	3	17	-	-	-	-
CALL nz,addr	C4 dr ad	3	ı17ı10	-	-	-	-
CALL z,addr	CC dr ad	3	ı17ı10	-	-	-	-
CALL nc,addr	D4 dr ad	3	ı17ı10	-	-	-	-
CALL c,addr	DC dr ad	3	ı17ı10	-	-	-	-
CALL po,addr	E4 dr ad	3	ı17ı10	-	-	-	-
CALL pe,addr	EC dr ad	3	ı17ı10	-	-	-	-
CALL p,addr	F4 dr ad	3	ı17ı10	-	-	-	-
CALL m,addr	FC dr ad	3	ı17ı10	-	-	-	-
CCF	3F	1	4	-	-	-	x
CP n	FE n	2	7	=7	=	v	<
CP A	BF	1	4	=7	=	v	<
CP B	B8	1	4	=7	=	v	<
CP C	B9	1	4	=7	=	v	<
CP D	BA	1	4	=7	=	v	<
CP E	BB	1	4	=7	=	v	<
CP H	BC	1	4	=7	=	v	<
CP L	BD	1	4	=7	=	v	<
CP (HL)	BE	1	7	=7	=	v	<
CP (IX+d)	DD BE d	3	19	=7	=	v	<
CP (IY+d)	FD BE d	3	19	=7	=	v	<
CP HIX	DD BC	2	8	=7	=	v	<
CP LIX	DD BD	2	8	=7	=	v	<
CP HIY	FD BC	2	8	=7	=	v	<
CP LIY	FD BD	2	8	=7	=	v	<
CPD	ED A9	2	16	?	=A	BC	-
CPDR	ED B9	2	=16#21	?	=A	BC	-
CPI	ED A1	2	16	?	=A	BC	-
CPIR	ED B2	2	=16#21	?	=A	BC	-
CPL	2F	1	4	-	-	-	-
DAA	27	1	4	7	z	p	c
DEC A	3D	1	4	7	z	v	-
DEC B	05	1	4	7	z	v	-
DEC C	0D	1	4	7	z	v	-
DEC D	15	1	4	7	z	v	-
DEC E	1D	1	4	7	z	v	-
DEC H	25	1	4	7	z	v	-
DEC L	2D	1	4	7	z	v	-
DEC (HL)	35	1	11	7	z	v	-
DEC (IX+d)	DD 35 d	3	23	7	z	v	-
DEC (IY+d)	FD 35 d	3	23	7	z	v	-
DEC HIX	DD 25	2	8	7	z	v	-
DEC LIX	DD 2D	2	8	7	z	v	-
DEC HIY	FD 25	2	8	7	z	v	-
DEC LIY	FD 2D	2	8	7	z	v	-
DEC BC	0B	1	6	-	-	-	-
DEC DE	1B	1	6	-	-	-	-
DEC HL	2B	1	6	-	-	-	-
DEC SP	3B	1	6	-	-	-	-
DEC IX	DD 2B	2	10	-	-	-	-
DEC IY	FD 2B	2	10	-	-	-	-
DI	F3	1	4	-	-	-	-
DJNZ d	10 d	2	ı13ı8	-	-	-	-
EI	FB	1	4	-	-	-	-
EX AF,AF'	08	1	4	s'	z'	p'	c'
EX DE,HL	EB	1	4	-	-	-	-
EX (SP),HL	E3	1	19	-	-	-	-
EX (SP),IX	DD E3	2	23	-	-	-	-
EX (SP),IY	FD E3	2	23	-	-	-	-
EXX	D9	1	4	-	-	-	-
HALT	76	1	min 4	-	-	-	-
IM 0	ED 46	2	8	-	-	-	-
IM 1	ED 56	2	8	-	-	-	-
IM 2	ED 5E	2	8	-	-	-	-
IN A,(n)	DB n	2	11	-	-	-	-
IN A,(C)	ED 78	2	12	7	z	p	0
IN B,(C)	ED 40	2	12	7	z	p	0
IN C,(C)	ED 48	2	12	7	z	p	0
IN D,(C)	ED 50	2	12	7	z	p	0
IN E,(C)	ED 58	2	12	7	z	p	0
IN H,(C)	ED 60	2	12	7	z	p	0
IN L,(C)	ED 68	2	12	7	z	p	0
INC A	3C	1	4	7	z	v	-
INC B	04	1	4	7	z	v	-
INC C	0C	1	4	7	z	v	-
INC D	14	1	4	7	z	v	-
INC E	1C	1	4	7	z	v	-
INC H	24	1	4	7	z	v	-
INC L	2C	1	4	7	z	v	-
INC (HL)	34	1	11	7	z	v	-
INC (IX+d)	DD 34	2	23	7	z	v	-
INC (IY+d)	FD 34	2	23	7	z	v	-
INC HIX	DD 24	2	8	7	z	v	-
INC LIX	DD 2C	2	8	7	z	v	-
INC HIY	FD 24	2	8	7	z	v	-
INC LIY	FD 2C	2	8	7	z	v	-
INC BC	03	1	6	-	-	-	-
INC DE	13	1	6	-	-	-	-
INC HL	23	1	6	-	-	-	-
INC SP	33	1	6	-	-	-	-
INC IX	DD 23	2	10	-	-	-	-

Instruction	Opcode	Bytes	Ts	S	Z	P	C
INC IY	FD 23	2	10	-	-	-	-
IND	ED AA	2	16	?	<>B	?	-
INDR	ED BA	2	${}_{z}16{}_{nz}21$	?	1	?	-
INI	ED A2	2	16	?	<>B	?	-
INIR	ED B2	2	${}_{z}16{}_{nz}21$	?	1	?	-
JP addr	C3 dr ad	3	10	-	-	-	-
JP nz,addr	C2 dr ad	3	10	-	-	-	-
JP z,addr	CA dr ad	3	10	-	-	-	-
JP nc,addr	D2 dr ad	3	10	-	-	-	-
JP c,addr	DA dr ad	3	10	-	-	-	-
JP po,addr	E2 dr ad	3	10	-	-	-	-
JP pe,addr	EA dr ad	3	10	-	-	-	-
JP p,addr	F2 dr ad	3	10	-	-	-	-
JP m,addr	FA dr ad	3	10	-	-	-	-
JP (HL)	E9	1	4	-	-	-	-
JP (IX)	DD E9	2	8	-	-	-	-
JP (IY)	FD E9	2	8	-	-	-	-
JR d	18 d	2	12	-	-	-	-
JR nz,d	20 d	2	${}_{t}12{}_{f}7$	-	-	-	-
JR z,d	28 d	2	${}_{t}12{}_{f}7$	-	-	-	-
JR nc,d	30 d	2	${}_{t}12{}_{f}7$	-	-	-	-
JR c,d	38 d	2	${}_{t}12{}_{f}7$	-	-	-	-
LD A,n	3E n	2	7	-	-	-	-
LD B,n	06 n	2	7	-	-	-	-
LD C,n	0E n	2	7	-	-	-	-
LD D,n	16 n	2	7	-	-	-	-
LD E,n	1E n	2	7	-	-	-	-
LD H,n	26 n	2	7	-	-	-	-
LD L,n	2E n	2	7	-	-	-	-
LD (HL),n	36 n	2	10	-	-	-	-
LD (IX+d),n	DD 36 d n	4	19	-	-	-	-
LD (IY+d),n	FD 36 d n	4	19	-	-	-	-
LD HIX	DD 26 n	3	11	-	-	-	-
LD LIX	DD 2E n	3	11	-	-	-	-
LD HIY	FD 26 n	3	11	-	-	-	-
LD LIY	FD 2E n	3	11	-	-	-	-
LD A,A	7F	1	4	-	-	-	-
LD A,B	78	1	4	-	-	-	-
LD A,C	79	1	4	-	-	-	-
LD A,D	7A	1	4	-	-	-	-
LD A,E	7B	1	4	-	-	-	-
LD A,H	7C	1	4	-	-	-	-
LD A,L	7D	1	4	-	-	-	-
LD B,A	47	1	4	-	-	-	-
LD B,B	40	1	4	-	-	-	-
LD B,C	41	1	4	-	-	-	-
LD B,D	42	1	4	-	-	-	-
LD B,E	43	1	4	-	-	-	-
LD B,H	44	1	4	-	-	-	-
LD B,L	45	1	4	-	-	-	-
LD C,A	4F	1	4	-	-	-	-
LD C,B	48	1	4	-	-	-	-
LD C,C	49	1	4	-	-	-	-
LD C,D	4A	1	4	-	-	-	-
LD C,E	4B	1	4	-	-	-	-
LD C,H	4C	1	4	-	-	-	-
LD C,L	4D	1	4	-	-	-	-
LD D,A	57	1	4	-	-	-	-
LD D,B	50	1	4	-	-	-	-
LD D,C	51	1	4	-	-	-	-
LD D,D	52	1	4	-	-	-	-
LD D,E	53	1	4	-	-	-	-
LD D,H	54	1	4	-	-	-	-
LD D,L	55	1	4	-	-	-	-
LD E,A	5F	1	4	-	-	-	-
LD E,B	58	1	4	-	-	-	-
LD E,C	59	1	4	-	-	-	-
LD E,D	5A	1	4	-	-	-	-
LD E,E	5B	1	4	-	-	-	-
LD E,H	5C	1	4	-	-	-	-
LD E,L	5D	1	4	-	-	-	-
LD H,A	67	1	4	-	-	-	-
LD H,B	60	1	4	-	-	-	-
LD H,C	61	1	4	-	-	-	-
LD H,D	62	1	4	-	-	-	-
LD H,E	63	1	4	-	-	-	-
LD H,H	64	1	4	-	-	-	-
LD H,L	65	1	4	-	-	-	-
LD L,A	6F	1	4	-	-	-	-
LD L,B	68	1	4	-	-	-	-
LD L,C	69	1	4	-	-	-	-
LD L,D	6A	1	4	-	-	-	-
LD L,E	6B	1	4	-	-	-	-
LD L,H	6C	1	4	-	-	-	-
LD L,L	6D	1	4	-	-	-	-
LD (HL),A	77	1	7	-	-	-	-
LD (HL),B	70	1	7	-	-	-	-
LD (HL),C	71	1	7	-	-	-	-
LD (HL),D	72	1	7	-	-	-	-
LD (HL),E	73	1	7	-	-	-	-
LD (HL),H	74	1	7	-	-	-	-
LD (HL),L	75	1	7	-	-	-	-
LD (IX+d),A	DD 77 d	3	19	-	-	-	-
LD (IX+d),B	DD 70 d	3	19	-	-	-	-
LD (IX+d),C	DD 71 d	3	19	-	-	-	-
LD (IX+d),D	DD 72 d	3	19	-	-	-	-
LD (IX+d),E	DD 73 d	3	19	-	-	-	-
LD (IX+d),H	DD 74 d	3	19	-	-	-	-
LD (IX+d),L	DD 75 d	3	19	-	-	-	-
LD (IY+d),A	FD 77 d	3	19	-	-	-	-
LD (IY+d),B	FD 70 d	3	19	-	-	-	-
LD (IY+d),C	FD 71 d	3	19	-	-	-	-
LD (IY+d),D	FD 72 d	3	19	-	-	-	-
LD (IY+d),E	FD 73 d	3	19	-	-	-	-
LD (IY+d),H	FD 74 d	3	19	-	-	-	-
LD (IY+d),L	FD 75 d	3	19	-	-	-	-
LD HIX,A	DD 67	2	8	-	-	-	-
LD HIX,B	DD 60	2	8	-	-	-	-
LD HIX,C	DD 61	2	8	-	-	-	-
LD HIX,D	DD 62	2	8	-	-	-	-
LD HIX,E	DD 63	2	8	-	-	-	-
LD HIY,A	FD 67	2	8	-	-	-	-
LD HIY,B	FD 60	2	8	-	-	-	-
LD HIY,C	FD 61	2	8	-	-	-	-
LD HIY,D	FD 62	2	8	-	-	-	-
LD HIY,E	FD 63	2	8	-	-	-	-

Instruction	Opcode	Bytes	Ts	S	Z	P	C
LD LIX,A	DD 6F	2	8	-	-	-	-
LD LIX,B	DD 68	2	8	-	-	-	-
LD LIX,C	DD 69	2	8	-	-	-	-
LD LIX,D	DD 6A	2	8	-	-	-	-
LD LIX,E	DD 6B	2	8	-	-	-	-
LD LIY,A	FD 6F	2	8	-	-	-	-
LD LIY,B	FD 68	2	8	-	-	-	-
LD LIY,C	FD 69	2	8	-	-	-	-
LD LIY,D	FD 6A	2	8	-	-	-	-
LD LIY,E	FD 6B	2	8	-	-	-	-
LD A,(HL)	7E	1	7	-	-	-	-
LD B,(HL)	46	1	7	-	-	-	-
LD C,(HL)	4E	1	7	-	-	-	-
LD D,(HL)	56	1	7	-	-	-	-
LD E,(HL)	5E	1	7	-	-	-	-
LD H,(HL)	66	1	7	-	-	-	-
LD L,(HL)	6E	1	7	-	-	-	-
LD A,(IX+d)	DD 7E d	3	19	-	-	-	-
LD B,(IX+d)	DD 46 d	3	19	-	-	-	-
LD C,(IX+d)	DD 4E d	3	19	-	-	-	-
LD D,(IX+d)	DD 56 d	3	19	-	-	-	-
LD E,(IX+d)	DD 5E d	3	19	-	-	-	-
LD H,(IX+d)	DD 66 d	3	19	-	-	-	-
LD L,(IX+d)	DD 6E d	3	19	-	-	-	-
LD A,(IY+d)	FD 7E d	3	19	-	-	-	-
LD B,(IY+d)	FD 46 d	3	19	-	-	-	-
LD C,(IY+d)	DD 4E d	3	19	-	-	-	-
LD D,(IY+d)	FD 56 d	3	19	-	-	-	-
LD E,(IY+d)	FD 5E d	3	19	-	-	-	-
LD H,(IY+d)	FD 66 d	3	19	-	-	-	-
LD L,(IY+d)	FD 6E d	3	19	-	-	-	-
LD A,HIX	DD 7C	2	8	-	-	-	-
LD B,HIX	DD 44	2	8	-	-	-	-
LD C,HIX	DD 4C	2	8	-	-	-	-
LD D,HIX	DD 54	2	8	-	-	-	-
LD E,HIX	DD 5C	2	8	-	-	-	-
LD A,HIY	FD 7C	2	8	-	-	-	-
LD B,HIY	FD 44	2	8	-	-	-	-
LD C,HIY	FD 4C	2	8	-	-	-	-
LD D,HIY	FD 54	2	8	-	-	-	-
LD E,HIY	FD 5C	2	8	-	-	-	-
LD A,LIX	DD 7D	2	8	-	-	-	-
LD B,LIX	DD 45	2	8	-	-	-	-
LD C,LIX	DD 4D	2	8	-	-	-	-
LD D,LIX	DD 55	2	8	-	-	-	-
LD E,LIX	DD 5D	2	8	-	-	-	-
LD A,LIY	FD 7D	2	8	-	-	-	-
LD B,LIY	FD 45	2	8	-	-	-	-
LD C,LIY	FD 4D	2	8	-	-	-	-
LD D,LIY	FD 55	2	8	-	-	-	-
LD E,LIY	FD 5D	2	8	-	-	-	-
LD HIX,HIX	DD 64	2	8	-	-	-	-
LD HIX,LIX	DD 65	2	8	-	-	-	-
LD HIY,HIY	FD 64	2	8	-	-	-	-
LD HIY,LIY	FD 65	2	8	-	-	-	-
LD LIX,HIX	DD 6C	2	8	-	-	-	-
LD LIX,LIX	DD 6D	2	8	-	-	-	-
LD LIY,HIY	FD 6C	2	8	-	-	-	-

Instruction	Opcode	Bytes	Ts	S	Z	P	C
LD LIY,LIY	FD 6D	2	8	-	-	-	-
LD A,(BC)	0A	1	7	-	-	-	-
LD A,(DE)	1A	1	7	-	-	-	-
LD A,(HL)	7E	1	7	-	-	-	-
LD (BC),A	02	1	7	-	-	-	-
LD (DE),A	12	1	7	-	-	-	-
LD (HL),A	77	1	7	-	-	-	-
LD SP,HL	F9	1	6	-	-	-	-
LD SP,IX	DD F9	2	10	-	-	-	-
LD SP,IY	FD F9	2	10	-	-	-	-
LD BC,hilo	01 lo hi	3	10	-	-	-	-
LD DE,hilo	11 lo hi	3	10	-	-	-	-
LD HL,hilo	21 lo hi	3	10	-	-	-	-
LD SP,hilo	31 lo hi	3	10	-	-	-	-
LD IX,hilo	DD 21 lo hi	4	14	-	-	-	-
LD IY,hilo	FD 21 lo hi	4	14	-	-	-	-
LD A,(addr)	3A dr ad	3	13	-	-	-	-
LD HL,(addr)	2A dr ad	3	16	-	-	-	-
LD IX,(addr)	DD 2A dr ad	4	20	-	-	-	-
LD IY,(addr)	FD 2A dr ad	4	20	-	-	-	-
LD BC,(addr)	ED 4B dr ad	4	20	-	-	-	-
LD DE,(addr)	ED 5B dr ad	4	20	-	-	-	-
LD HL,(addr)	ED 6B dr ad	4	20	-	-	-	-
LD SP,(addr)	ED 7B dr ad	4	20	-	-	-	-
LD (addr),A	32 dr ad	3	13	-	-	-	-
LD (addr),HL	22 dr ad	3	16	-	-	-	-
LD (addr),IX	DD 22 dr ad	4	20	-	-	-	-
LD (addr),IY	FD 22 dr ad	4	20	-	-	-	-
LD (addr),BC	ED 43 dr ad	4	20	-	-	-	-
LD (addr),DE	ED 53 dr ad	4	20	-	-	-	-
LD (addr),HL	ED 63 dr ad	4	20	-	-	-	-
LD (addr),SP	ED 73 dr ad	4	20	-	-	-	-
LD A,I	ED 57	2	9	7	z	i	0
LD A,R	ED 5F	2	9	7	z	i	0
LD I,A	ED 47	2	9	-	-	-	-
LD R,A	ED 4F	2	9	-	-	-	-
LDD	ED A8	2	16	-	-	BC	-
LDDR	ED B8	2	z16nz21	-	-	0	-
LDI	ED A0	2	16	-	-	BC	-
LDIR	ED B0	2	z16nz21	-	-	0	-
NEG	ED 44	2	8	7	z	A80	A0
NOP	00	1	4	-	-	-	-
OR n	F6 n	2	7	7	z	p	0
OR A	B7	1	4	7	z	p	0
OR B	B0	1	4	7	z	p	0
OR C	B1	1	4	7	z	p	0
OR D	B2	1	4	7	z	p	0
OR E	B3	1	4	7	z	p	0
OR H	B4	1	4	7	z	p	0
OR L	B5	1	4	7	z	p	0
OR (HL)	B6	1	7	7	z	P	0
OR (IX+d)	DD B6 d	3	19	7	z	p	0
OR (IY+d)	FD B6 d	3	19	7	z	p	0
OR HIX	DD B4	2	8	7	z	p	0
OR HIY	FD B4	2	8	7	z	p	0
OR LIX	DD B5	2	8	7	z	p	0
OR LIY	FD B5	2	8	7	z	p	0
OUT (n),A	D3 n	2	11	-	-	-	-

Instruction	Opcode	Bytes	Ts	S	Z	P	C
OUT (C),A	ED 79	2	12	-	-	-	-
OUT (C),B	ED 41	2	12	-	-	-	-
OUT (C),C	ED 49	2	12	-	-	-	-
OUT (C),D	ED 51	2	12	-	-	-	-
OUT (C),E	ED 59	2	12	-	-	-	-
OUT (C),H	ED 61	2	12	-	-	-	-
OUT (C),L	ED 69	2	12	-	-	-	-
OTDR	ED BB	2	z16nz21	?	1	?	-
OTIR	ED B3	2	z16nz21	?	1	?	-
OUTD	ED AB	2	16	?	<>B	?	-
OUTI	ED A3	2	16	?	<>B	?	-
POP AF	F1	1	10		POP flags		
POP BC	C1	1	10	-	-	-	-
POP DE	D1	1	10	-	-	-	-
POP HL	E1	1	10	-	-	-	-
POP IX	DD E1	2	14	-	-	-	-
POP IY	FD E1	2	14	-	-	-	-
PUSH AF	F5	1	11	-	-	-	-
PUSH BC	C5	1	11	-	-	-	-
PUSH DE	D5	1	11	-	-	-	-
PUSH HL	E5	1	11	-	-	-	-
PUSH IX	DD E5	2	15	-	-	-	-
PUSH IY	FD E5	2	15	-	-	-	-
RES 0,A	CB 87	2	8	-	-	-	-
RES 0,B	CB 80	2	8	-	-	-	-
RES 0,C	CB 81	2	8	-	-	-	-
RES 0,D	CB 82	2	8	-	-	-	-
RES 0,E	CB 83	2	8	-	-	-	-
RES 0,H	CB 84	2	8	-	-	-	-
RES 0,L	CB 85	2	8	-	-	-	-
RES 1,A	CB 8F	2	8	-	-	-	-
RES 1,B	CB 88	2	8	-	-	-	-
RES 1,C	CB 89	2	8	-	-	-	-
RES 1,D	CB 8A	2	8	-	-	-	-
RES 1,E	CB 8B	2	8	-	-	-	-
RES 1,H	CB 8C	2	8	-	-	-	-
RES 1,L	CB 8D	2	8	-	-	-	-
RES 2,A	CB 97	2	8	-	-	-	-
RES 2,B	CB 90	2	8	-	-	-	-
RES 2,C	CB 91	2	8	-	-	-	-
RES 2,D	CB 92	2	8	-	-	-	-
RES 2,E	CB 93	2	8	-	-	-	-
RES 2,H	CB 94	2	8	-	-	-	-
RES 2,L	CB 95	2	8	-	-	-	-
RES 3,A	CB 9F	2	8	-	-	-	-
RES 3,B	CB 98	2	8	-	-	-	-
RES 3,C	CB 99	2	8	-	-	-	-
RES 3,D	CB 9A	2	8	-	-	-	-
RES 3,E	CB 9B	2	8	-	-	-	-
RES 3,H	CB 9C	2	8	-	-	-	-
RES 3,L	CB 9D	2	8	-	-	-	-
RES 4,A	CB A7	2	8	-	-	-	-
RES 4,B	CB A0	2	8	-	-	-	-
RES 4,C	CB A1	2	8	-	-	-	-
RES 4,D	CB A2	2	8	-	-	-	-
RES 4,E	CB A3	2	8	-	-	-	-
RES 4,H	CB A4	2	8	-	-	-	-
RES 4,L	CB A5	2	8	-	-	-	-

Instruction	Opcode	Bytes	Ts	S	Z	P	C
RES 5,A	CB AF	2	8	-	-	-	-
RES 5,B	CB A8	2	8	-	-	-	-
RES 5,C	CB A9	2	8	-	-	-	-
RES 5,D	CB AA	2	8	-	-	-	-
RES 5,E	CB AB	2	8	-	-	-	-
RES 5,H	CB AC	2	8	-	-	-	-
RES 5,L	CB AD	2	8	-	-	-	-
RES 6,A	CB B7	2	8	-	-	-	-
RES 6,B	CB B0	2	8	-	-	-	-
RES 6,C	CB B1	2	8	-	-	-	-
RES 6,D	CB B2	2	8	-	-	-	-
RES 6,E	CB B3	2	8	-	-	-	-
RES 6,H	CB B4	2	8	-	-	-	-
RES 6,L	CB B5	2	8	-	-	-	-
RES 7,A	CB BF	2	8	-	-	-	-
RES 7,B	CB B8	2	8	-	-	-	-
RES 7,C	CB B9	2	8	-	-	-	-
RES 7,D	CB BA	2	8	-	-	-	-
RES 7,E	CB BB	2	8	-	-	-	-
RES 7,H	CB BC	2	8	-	-	-	-
RES 7,L	CB BD	2	8	-	-	-	-
RES 0,(HL)	CB 86	2	15	-	-	-	-
RES 1,(HL)	CB 8E	2	15	-	-	-	-
RES 2,(HL)	CB 96	2	15	-	-	-	-
RES 3,(HL)	CB 9E	2	15	-	-	-	-
RES 4,(HL)	CB A6	2	15	-	-	-	-
RES 5,(HL)	CB AE	2	15	-	-	-	-
RES 6,(HL)	CB B6	2	15	-	-	-	-
RES 7,(HL)	CB BE	2	15	-	-	-	-
RES 0,(IX+d)	DD CB d 86	4	23	-	-	-	-
RES 0,(IY+d)	FD CB d 86	4	23	-	-	-	-
RES 1,(IX+d)	DD CB d 8E	4	23	-	-	-	-
RES 1,(IY+d)	FD CB d 8E	4	23	-	-	-	-
RES 2,(IX+d)	DD CB d 96	4	23	-	-	-	-
RES 2,(IY+d)	FD CB d 96	4	23	-	-	-	-
RES 3,(IX+d)	DD CB d 9E	4	23	-	-	-	-
RES 3,(IY+d)	FD CB d 9E	4	23	-	-	-	-
RES 4,(IX+d)	DD CB d A6	4	23	-	-	-	-
RES 4,(IY+d)	FD CB d A6	4	23	-	-	-	-
RES 5,(IX+d)	DD CB d AE	4	23	-	-	-	-
RES 5,(IY+d)	FD CB d AE	4	23	-	-	-	-
RES 6,(IX+d)	DD CB d B6	4	23	-	-	-	-
RES 6,(IY+d)	FD CB d B6	4	23	-	-	-	-
RES 7,(IX+d)	DD CB d BE	4	23	-	-	-	-
RES 7,(IY+d)	FD CB d BE	4	23	-	-	-	-
RET	C9	1	10	-	-	-	-
RET NZ	C0	1	t11f8	-	-	-	-
RET Z	C8	1	t11f8	-	-	-	-
RET NC	D0	1	t11f8	-	-	-	-
RET C	D8	1	t11f8	-	-	-	-
RET PO	E0	1	t11f8	-	-	-	-
RET PE	E8	1	t11f8	-	-	-	-
RET P	F0	1	t11f8	-	-	-	-
RET M	F8	1	t11f8	-	-	-	-
RETI	ED 4D	2	14	-	-	-	-
RETN	ED 45	2	14	-	-	-	-
RLA	17	1	4	-	-	-	r7
RL A	CB 17	2	8	7	z	p	r7

Instruction	Opcode	Bytes	Ts	S	Z	P	C
RL B	CB 10	2	8	7	z	p	r7
RL C	CB 11	2	8	7	z	p	r7
RL D	CB 12	2	8	7	z	p	r7
RL E	CB 13	2	8	7	z	p	r7
RL H	CB 14	2	8	7	z	p	r7
RL L	CB 15	2	8	7	z	p	r7
RL (HL)	CB 16	2	15	7	z	p	r7
RL (IX+d)	DD CB d 16	4	23	7	z	p	r7
RL (IY+d)	FD CB d 16	4	23	7	z	p	r7
RLCA	07	1	4	-	-	-	r7
RLC A	CB 07	2	8	7	z	p	r7
RLC B	CB 00	2	8	7	z	p	r7
RLC C	CB 01	2	8	7	z	p	r7
RLC D	CB 02	2	8	7	z	p	r7
RLC E	CB 03	2	8	7	z	p	r7
RLC H	CB 04	2	8	7	z	p	r7
RLC L	CB 05	2	8	7	z	p	r7
RLC (HL)	CB 06	2	15	7	z	p	r7
RLC (IX+d)	DD CB d 06	4	23	7	z	p	r7
RLC (IY+d)	FD CB d 06	4	23	7	z	p	r7
RLD	ED 6F	2	18	7	z	p	-
RRA	1F	1	4	-	-	-	r0
RR A	CB 1F	2	8	7	z	p	r0
RR B	CB 18	2	8	7	z	p	r0
RR C	CB 19	2	8	7	z	p	r0
RR D	CB 1A	2	8	7	z	p	r0
RR E	CB 1B	2	8	7	z	p	r0
RR H	CB 1C	2	8	7	z	p	r0
RR L	CB 1D	2	8	7	z	p	r0
RR (HL)	CB 1E	2	15	7	z	p	r0
RR (IX+d)	DD CB d 1E	4	23	7	z	p	r0
RR (IY+d)	FD CB d 1E	4	23	7	z	p	r0
RRCA	0F	1	4	-	-	-	r0
RRC A	CB 0F	2	8	7	z	p	r0
RRC B	CB 08	2	8	7	z	p	r0
RRC C	CB 09	2	8	7	z	p	r0
RRC D	CB 0A	2	8	7	z	p	r0
RRC E	CB 0B	2	8	7	z	p	r0
RRC H	CB 0C	2	8	7	z	p	r0
RRC L	CB 0D	2	8	7	z	p	r0
RRC (HL)	CB 0E	2	15	7	z	p	r0
RRC (IX+d)	DD CB d 0E	4	23	7	z	p	r0
RRC (IY+d)	FD CB d 0E	4	23	7	z	p	r0
RRD	ED 67	2	18	7	z	p	-
RST 0	C7	1	11	-	-	-	-
RST 1,addr	CF dr ad	3	(11)	-	-	-	-
RST 2,addr	D7 dr ad	3	(11)	-	-	-	-
RST 3,addr	DF dr ad	3	(11)	-	-	-	-
RST 4	E7	1	11	-	-	-	-
RST 5,addr	EF dr ad	3	(11)	-	-	-	-
RST 6	F7	1	11	-	-	-	-
RST 7	FF	1	11	-	-	-	-
SBC A,n	DE n	2	7	7	z	v	b
SBC A,A	9F	1	4	7	z	v	b
SBC A,B	98	1	4	7	z	v	b
SBC A,C	99	1	4	7	z	v	b
SBC A,D	9A	1	4	7	z	v	b
SBC A,E	9B	1	4	7	z	v	b
SBC A,H	9C	1	4	7	z	v	b
SBC A,L	9D	1	4	7	z	v	b
SBC A,(HL)	9E	1	7	7	z	v	b
SBC A,(IX+d)	DD 9E d	3	19	7	z	v	b
SBC A,(IY+d)	FD 9E d	3	19	7	z	v	b
SBC A,HIX	DD 9C	2	8	7	z	v	b
SBC A,HIY	FD 9C	2	8	7	z	v	b
SBC A,LIX	DD 9D	2	8	7	z	v	b
SBC A,LIY	FD 9D	2	8	7	z	v	b
SBC HL,BC	ED 42	2	15	15	z	v	b
SBC HL,DE	ED 52	2	15	15	z	v	b
SBC HL,HL	ED 62	2	15	15	z	v	b
SBC HL,SP	ED 72	2	15	15	z	v	b
SCF	37	1	4	-	-	-	1
SET 0,A	CB C7	2	8	-	-	-	-
SET 0,B	CB C0	2	8	-	-	-	-
SET 0,C	CB C1	2	8	-	-	-	-
SET 0,D	CB C2	2	8	-	-	-	-
SET 0,E	CB C3	2	8	-	-	-	-
SET 0,H	CB C4	2	8	-	-	-	-
SET 0,L	CB C5	2	8	-	-	-	-
SET 1,A	CB CF	2	8	-	-	-	-
SET 1,B	CB C8	2	8	-	-	-	-
SET 1,C	CB C9	2	8	-	-	-	-
SET 1,D	CB CA	2	8	-	-	-	-
SET 1,E	CB CB	2	8	-	-	-	-
SET 1,H	CB CC	2	8	-	-	-	-
SET 1,L	CB CD	2	8	-	-	-	-
SET 2,A	CB D7	2	8	-	-	-	-
SET 2,B	CB D0	2	8	-	-	-	-
SET 2,C	CB D1	2	8	-	-	-	-
SET 2,D	CB D2	2	8	-	-	-	-
SET 2,E	CB D3	2	8	-	-	-	-
SET 2,H	CB D4	2	8	-	-	-	-
SET 2,L	CB D5	2	8	-	-	-	-
SET 3,A	CB DF	2	8	-	-	-	-
SET 3,B	CB D8	2	8	-	-	-	-
SET 3,C	CB D9	2	8	-	-	-	-
SET 3,D	CB DA	2	8	-	-	-	-
SET 3,E	CB DB	2	8	-	-	-	-
SET 3,H	CB DC	2	8	-	-	-	-
SET 3,L	CB DD	2	8	-	-	-	-
SET 4,A	CB E7	2	8	-	-	-	-
SET 4,B	CB E0	2	8	-	-	-	-
SET 4,C	CB E1	2	8	-	-	-	-
SET 4,D	CB E2	2	8	-	-	-	-
SET 4,E	CB E3	2	8	-	-	-	-
SET 4,H	CB E4	2	8	-	-	-	-
SET 4,L	CB E5	2	8	-	-	-	-
SET 5,A	CB EF	2	8	-	-	-	-
SET 5,B	CB E8	2	8	-	-	-	-
SET 5,C	CB E9	2	8	-	-	-	-
SET 5,D	CB EA	2	8	-	-	-	-
SET 5,E	CB EB	2	8	-	-	-	-
SET 5,H	CB EC	2	8	-	-	-	-
SET 5,L	CB ED	2	8	-	-	-	-
SET 6,A	CB F7	2	8	-	-	-	-
SET 6,B	CB F0	2	8	-	-	-	-

Instruction	Opcode	Bytes	Ts	S	Z	P	C
SET 6,C	CB F1	2	8	-	-	-	-
SET 6,D	CB F2	2	8	-	-	-	-
SET 6,E	CB F3	2	8	-	-	-	-
SET 6,H	CB F4	2	8	-	-	-	-
SET 6,L	CB F5	2	8	-	-	-	-
SET 7,A	CB FF	2	8	-	-	-	-
SET 7,B	CB F8	2	8	-	-	-	-
SET 7,C	CB F9	2	8	-	-	-	-
SET 7,D	CB FA	2	8	-	-	-	-
SET 7,E	CB FB	2	8	-	-	-	-
SET 7,H	CB FC	2	8	-	-	-	-
SET 7,L	CB FD	2	8	-	-	-	-
SET 0,(HL)	CB C6	2	15	-	-	-	-
SET 1,(HL)	CB CE	2	15	-	-	-	-
SET 2,(HL)	CB D6	2	15	-	-	-	-
SET 3,(HL)	CB DE	2	15	-	-	-	-
SET 4,(HL)	CB E6	2	15	-	-	-	-
SET 5,(HL)	CB EE	2	15	-	-	-	-
SET 6,(HL)	CB F6	2	15	-	-	-	-
SET 7,(HL)	CB FE	2	15	-	-	-	-
SET 0,(IX+d)	DD CB d C6	4	23	-	-	-	-
SET 1,(IX+d)	DD CB d CE	4	23	-	-	-	-
SET 2,(IX+d)	DD CB d D6	4	23	-	-	-	-
SET 3,(IX+d)	DD CB d DE	4	23	-	-	-	-
SET 4,(IX+d)	DD CB d E6	4	23	-	-	-	-
SET 5,(IX+d)	DD CB d EE	4	23	-	-	-	-
SET 6,(IX+d)	DD CB d F6	4	23	-	-	-	-
SET 7,(IX+d)	DD CB d FE	4	23	-	-	-	-
SET 0,(IY+d)	FD CB d C6	4	23	-	-	-	-
SET 1,(IY+d)	FD CB d CE	4	23	-	-	-	-
SET 2,(IY+d)	FD CB d D6	4	23	-	-	-	-
SET 3,(IY+d)	FD CB d DE	4	23	-	-	-	-
SET 4,(IY+d)	FD CB d E6	4	23	-	-	-	-
SET 5,(IY+d)	FD CB d EE	4	23	-	-	-	-
SET 6,(IY+d)	FD CB d F6	4	23	-	-	-	-
SET 7,(IY+d)	FD CB d FE	4	23	-	-	-	-
SLA A	CB 27	2	8	7	z	p	r7
SLA B	CB 20	2	8	7	z	p	r7
SLA C	CB 21	2	8	7	z	p	r7
SLA D	CB 22	2	8	7	z	p	r7
SLA E	CB 23	2	8	7	z	p	r7
SLA H	CB 24	2	8	7	z	p	r7
SLA L	CB 25	2	8	7	z	p	r7
SLA (HL)	CB 26	2	15	7	z	p	r7
SLA (IX+d)	DD CB d 26	4	23	7	z	p	r7
SLA (IY+d)	FD CB d 26	4	23	7	z	p	r7
SLL A	CB 37	2	8	7	z	p	r7
SLL B	CB 30	2	8	7	z	p	r7
SLL C	CB 31	2	8	7	z	p	r7
SLL D	CB 32	2	8	7	z	p	r7
SLL E	CB 33	2	8	7	z	p	r7
SLL H	CB 34	2	8	7	z	p	r7
SLL L	CB 35	2	8	7	z	p	r7
SLL (HL)	CB 36	2	15	7	z	p	r7
SLL (IX+d)	DD CB d 36	4	23	7	z	p	r7
SLL (IY+d)	FD CB d 36	4	23	7	z	p	r7
SRA A	CB 2F	2	8	7	z	p	r0
SRA B	CB 28	2	8	7	z	p	r0
SRA C	CB 29	2	8	7	z	p	r0
SRA D	CB 2A	2	8	7	z	p	r0
SRA E	CB 2B	2	8	7	z	p	r0
SRA H	CB 2C	2	8	7	z	p	r0
SRA L	CB 2D	2	8	7	z	p	r0
SRA (HL)	CB 2E	2	15	7	z	p	r0
SRA (IX+d)	DD CB d 2E	4	23	7	z	p	r0
SRA (IY+d)	FD CB d 2E	4	23	7	z	p	r0
SRL A	CB 3F	2	8	7	z	p	r0
SRL B	CB 38	2	8	7	z	p	r0
SRL C	CB 39	2	8	7	z	p	r0
SRL D	CB 3A	2	8	7	z	p	r0
SRL E	CB 3B	2	8	7	z	p	r0
SRL H	CB 3C	2	8	7	z	p	r0
SRL L	CB 3D	2	8	7	z	p	r0
SRL (HL)	CB 3E	2	15	7	z	p	r0
SRL (IX+d)	DD CB d 3E	4	23	7	z	p	r0
SRL (IY+d)	FD CB d 3E	4	23	7	z	p	r0
SUB n	D6 n	2	7	7	z	v	b
SUB A	97	1	4	7	z	v	b
SUB B	90	1	4	7	z	v	b
SUB C	91	1	4	7	z	v	b
SUB D	92	1	4	7	z	v	b
SUB E	93	1	4	7	z	v	b
SUB H	94	1	4	7	z	v	b
SUB L	95	1	4	7	z	v	b
SUB (HL)	96	1	7	7	z	v	b
SUB (IX+d)	DD 96 d	3	19	7	z	v	b
SUB (IY+d)	FD 96 d	3	19	7	z	v	b
SUB HIX	DD AC	2	8	7	z	v	b
SUB HIY	FD AC	2	8	7	z	v	b
SUB LIX	DD AD	2	8	7	z	v	b
SUB LIY	FD AD	2	8	7	z	v	b
XOR n	EE n	2	7	7	z	p	0
XOR A	AF	1	4	7	z	p	0
XOR B	A8	1	4	7	z	p	0
XOR C	A9	1	4	7	z	p	0
XOR D	AA	1	4	7	z	p	0
XOR E	AB	1	4	7	z	p	0
XOR H	AC	1	4	7	z	p	0
XOR L	AD	1	4	7	z	p	0
XOR (HL)	AC	1	7	7	z	p	0
XOR (IX+d)	DD AC d	3	19	7	z	p	0
XOR (IY+d)	FD AC d	3	19	7	z	p	0
XOR HIX	DD AC	2	8	7	z	p	0
XOR LIX	DD AC	2	8	7	z	p	0
XOR HIY	FD AD	2	8	7	z	p	0
XOR LIY	FD AD	2	8	7	z	p	0

The flag register is bit significant, and is defined as follows:

bit 7 — Sign | bit 6 — Zero | bit 5 — unused | bit 4 — Half Carry (cannot test)
bit 3 — unused | bit 2 — Parity/Overflow | bit 1 — Add/Subtract (cannot test) | bit 0 — Carry

APPENDIX A: TABLES OF BASIC TOKENS

When the Operating System stores BASIC commands and functions in memory, it does not store them as a string letters, but instead it uses a system of substitute values, which are called 'tokens'.

The 464 does not have some of these commands, and they are indicated by an asterisk (*); in addition, it does not have certain combined commands. It does have MID$ as an undocumented command, and although it recognises the function DEC$, it does not perform it.

BASIC COMMAND TOKENS

00	End of Line marker
01	':' (statement separator)
02	Integer or '%' variable
03	String or '$' variable
04	Real or '!' variable
05 to 0A are not used	
0B	DEFINT variable
0C	DEFSTR variable
0D	DEFREAL or undefined variable
0E	0 (number)
0F	1 (integer number)
10	2 (integer number)
11	3 (integer number)
12	4 (integer number)
13	5 (integer number)
14	6 (integer number)
15	7 (integer number)
16	8 (integer number)
17	9 (integer number)
18 is not used	
19	integer numbers in the range from 10 to 255; the value is held in the next byte of the program
1A	integer numbers in the range frim 256 to 32767 — the value is held in the next 2 bytes of the program
1B	binary numbers, &X — the value is contained in the next 2 bytes of the program
1C	hexadecimal numbers, & — the value is held in the next 2 bytes of the program
1D	program line number; converted to the 'address' before the start of the program line, and is held in the next 2 bytes (found when this part of has already been run)
1E	program line number — still held as a line number since this part of the program has not been run yet; the value is held in the next 2 bytes
1F	integers less than –32767 or greater than 32767, and floating point numbers; the value is held in the next 5 bytes of the program
20	SPACE; is used as a separator
21 is not used	
22	Quotation mark; delimits a string
23	Hash '#'; for windows and strings
24 to 27 are not used	
28	open bracket '('
29	close bracket ')'
2A to 2B are not used	
2C	comma; used as separator in Print items, and between parameters
2D	hyphen; used with DEFINT, etc
2E to 3A are not used	
3B	semi-colon — used as a separator for PRINT items
3C to 7B are not used	
7C,00	'|' — precedes an RSX command; the byte of &00 is inserted when the program is stored and will not appear on listing the program
7D to 7F are not used	
80	AFTER
81	AUTO
82	BORDER
83	CALL
84	CAT
85	CHAIN
86	CLEAR
87	CLG
88	CLOSEIN
89	CLOSEOUT
8A	CLS
8B	CONT
8C	DATA
8D	DEF
8E	DEFINT
8F	DEFREAL
90	DEFSTR
91	DEG
92	DELETE
93	DIM
94	DRAW
95	DRAWR
96	EDIT
01,97	ELSE (the &01 byte is inserted in the program when stored and will not appear when listed)
98	END
99	ENT
9A	ENV
9B	ERASE
9C	ERROR
9D	EVERY
9E	FOR
9F	GOSUB or GO SUB
A0	GOTO or GO TO
A1	IF
A2	INK
A3	INPUT
A4	KEY
A5	LET
A6	LINE
A7	LIST
A8	LOAD
A9	LOCATE
AA	MEMORY
AB	MERGE
AC	MID$ (function but no &FF byte)
AD	MODE
AE	MOVE
AF	MOVER
B0	NEXT
B1	NEW
B2	ON
B3	ON BREAK
B4	ON ERROR GOTO
B5	ON SQ
B6	OPENIN
B7	OPENOUT
B8	ORIGIN
B9	OUT
BA	PAPER
BB	PEN
BC	PLOT
BD	PLOTR
BE	POKE
BF	PRINT
01,C0	short form of REM (apostrophe); see ELSE for the &01 byte
C1	RAD
C2	RANDOMIZE
C3	READ
C4	RELEASE
C5	REM (written in full)
C6	RENUM
C7	RESTORE

C8	RESUME
C9	RETURN
CA	RUN
CB	SAVE
CC	SOUND
CD	SPEED
CE	STOP
CF	SYMBOL
D0	TAG
D1	TAGOFF
D2	TROFF
D3	TRON
D4	WAIT
D5	WEND
D6	WHILE
D7	WIDTH
D8	WINDOW
D9	WRITE
DA	ZONE
DB	DI
DC	EI
DD	FILL *
DE	GRAPHICS *
DF	MASK *
E0	FRAME *
E1	CURSOR *
E2 is not used	
E3	ERL (function but no &FF byte)
E4	FN (command when used with DEF token, and also a function)
E5	SPC
E6	STEP
E7	SWAP
E8 to E9 are not used	
EA	TAB
EB	THEN
EC	TO
ED	USING
EE	>
EF	=
F0	>=
F1	<
F2	<>
F3	<=
F4	+
F5	–
F6	*
F7	/
F8	^
F9	\
FA	AND
FB	MOD
FC	OR
FD	XOR
FE	NOT
FF	the prefix for a function

BASIC FUNCTION TOKENS – these functions are all preceded by a byte of &FF

00	ABS
01	ACS
02	ATN
03	CHR$
04	CINT
05	COS
06	CREAL
07	EXP
08	FIX
09	FRE
0A	INKEY
0B	INP
0C	INT
0D	JOY
0E	LEN
0F	LOG
10	LOG10
11	LOWER$
12	PEEK
13	REMAIN
14	SGN
15	SIN
16	SPACE$
17	SQ
18	SQR
19	STR$
1A	TAN
1B	UNT
1C	UPPER$
1D	VAL
1E to 3F are not used	
40	EOF
41	ERR
42	HIMEM
43	INKEY$
44	PI
45	RND
46	TIME
47	XPOS
48	YPOS
49	DERR *
4A to 70 are not used	
71	BIN$
72	DEC$ *
73	HEX$
74	INSTR
75	LEFT$
76	MAX
77	MIN
78	POS
79	RIGHT$
7A	ROUND
7B	STRING$
7C	TEST
7D	TESTR
7E	COPYCHR$ *
7F	VPOS

APPENDIX B: CPC PORT ADDRESSES

PORT	OUTPUT	INPUT
&7Fxx	Video Gate Array	Do not use
&BCxx	CRTC address	Do not use
&BDxx	CRTC data	Do not use
&BExx	Do not use	Reserved (CRTC status)
&BFxx	Do not use	CRTC data
&DFxx	Expansion ROM select	Not used
&EFxx	Printer port latch	Do not use
&F4xx	PPI port A data	PPI port A data
&F5xx	PPI port B data	PPI port B data
&F7xx	PPI control	Undefined
&F8nn	Expansion bus	Expansion bus
&F9nn	Expansion bus	Expansion bus
&FAnn	Expansion bus	Expansion bus
&FBnn	Expansion bus	Expansion bus

Values for 'nn' are reserved as follows:

&00 to &7B	Do not use
&7C to &7F	Reserved for disc interface
&80 to &BB	Do not use
&BC to &BF	Undefined (future use)
&C0 to &DB	Do not use
&DC to &DF	Reserved for comms interfaces
&E0 to &FE	For user peripherals
&FF	Reset peripherals

Note: These addresses apply to the CPC range of computers only, and may be incorrect for the Plus computers.

APPENDIX C: 464 to 6128 memory address conversion chart

This appendix is designed to enable 464 owners to convert the addresses present in their machines (listed in the left hand column) into the equivalent 6128 address, where one exists.

AC00	AC00	AC80-91	AC66-77	AE43	AE2A	B0BB	B09D
AC01-03		AC92-A3	AC78-89	AE45	AE2C	B0BD	
AC04-06		ACA4-DA2	AC8A-D88	AE46-4A	AE2D-31	B0BF	
AC07-09		ADA3-A5	AD89-8B	AE4B-52		B0C1	B09F
AC0A-0C		ADA6	AD8C		AE32-39	B0C2	B0A0
AC0D-0F		ADA8	AD8E	AE53-57	AE3A-3E	B0C3	B0A1
AC10-12		ADAA	AD90	AE57	AE3E	B0C4	B0A2
AC13-15			AD91	AE58	AE3F	B0C5	B0A3
AC16-18		ADAB	AD92	AE58-5B		B0C7-FF	
AC19-1B		ADAD	AD94		AE3F-43		B0A5-FF
AC1C	AC01	ADAF	AD96	AE5C-6A	AE44-50		
AC1D	AC02	ADB1	AD98		AE4C	B100	B82D
AC1F	AC04	ADB2-BA	AD99-A1		AE4E	B101	B82E
AC21	AC06	ADB3	AD9A		AE4F	B102	B82F
AC22	AC07	ADB4	AD9B	AE68	AE50	B104	B831
AC23		ADB5	AD9C	AE6B-6D		B105	B832
	AC08	ADB7	AD9E		AE51	B107-86	B834-B3
AC24	AC09	ADB8	AD9F	AE6E	AE52	B187-8A	B8B4-B7
AC25	AC0A	ADB9	ADA0	AE6F	AE53	B189	B8B6
	AC0B	ADBB-CA	ADA2-B1	AE70		B18B	B8B8
AC26	AC0C	ADBC	ADA3		AE54	B18C	B8B9
AC27-2B	AC0D-11	ADBD	ADA4	AE72	AE55	B18E	B8BB
AC2C	AC12	ADBE	ADA5	AE74	AE57	B190	B8BD
AC2E	AC14	ADBF-C1	ADA6-A8	AE75	AE58	B192	B8BF
AC30	AC16	ADC2-C4	ADA9-AB	AE77	AE5A	B193	B8C0
AC31	AC17	ADC5-C7	ADAC-AE	AE79	AE5C	B194	B8C1
AC32	AC18	ADC8-CA	ADAF-B1	AE7A		B195	B8C2
AC34	AC1A	ADC8-CF	ADB2-B6		AE5D	B196-A5	B8C3-D2
AC36	AC1C	ADD0-E03	ADB7-EA	AE7B	AE5E	B1A6	B8D3
AC38-43	AC1E-29	AE04	ADEB	AE7D	AE60		B8D5
AC3A	AC20	AE06-0B	ADED-F2	AE7F	AE62	B1A8	B8D6
AC3B	AC21	AE0C-25	ADF3-E0C	AE81	AE64	B1A9	B8D7
AC3C	AC22	AE26	AE0D	AE83	AE66	B1AB	B8D9
AC3E	AC24	AE27	AE0E	AE85	AE68		B8DA
AC3F	AC25	AE29	AE10	AE87	AE6A	B1AC-B9	B8DC-E9
AC40	AC26	AE2B	AE12	AE89	AE6C	B1AE	B8DE
AC42	AC28	AE2D	AE14		AE6E	B1B0	B8E0
AC44-4F	AC2A-35	AE2E	AE15	AE8B-B08A	AE6F-B06E	B1B2	B8E2
AC50-5B	AC36-41	AE30	AE17	B08B	B06F	B1B4	B8E4
AC5C-6D	AC42-53	AE32	AE19	B08D	B071	B1B6	B8E6
AC5E	AC44	AE34	AE1B	B08F	B073	B1B8	B8E8
AC60	AC46	AE36	AE1D	B091		B1BA-C7	
AC62	AC48	AE38	AE1F		B075		B8EA-F9
AC64	AC4A	AE39	AE20	B092			B8FA-FF
AC66	AC4C	AE3A	AE21	B094	B076		
AC68	AC4E	AE3B	AE22	B096	B078	B1C8	B7C3
AC69	AC4F	AE3D	AE24	B098	B07A	B1C9	B7C4
AC6A	AC50	AE3F	AE26	B09A	B07C		B7C5
AC6C	AC52	AE41	AE28	B09C-B9	B07E-9B	B1CB	B7C6
AC6E-7F	AC54-65	AE42	AE29	B0BA	B09C	B1CC-CE	B7C7-C9

B1CF-D6	
	B7CA-D1
B1D7	B7D2
B1D8	B7D3
B1D9-E9	B7D4-E4
B1EA-FA	B7E5-F5
B1FB	B7F6
B1FC	B7F7
B1FD	B7F8
B1FE	B7F9
B200	B7FB
	B7FD
	B7FE-801
	B802
	B804
	B805-2C
B202-06	
B207	
B208-0B	
B20C	B6B5
B20D-15	B6B6-BE
B216	
B217-1B	B6BF-C3
B21C-24	B6C4-CC
B225	
B226-2A	B6CD-D1
B22B-33	B6D2-DA
B234	
B235-39	B6DB-DF
B23A-42	B6E0-E8
B243	
B244-48	B6E9-ED
B249-51	B6EE-F6
B252	
B253-57	B6F7-FB
B258-60	B6FC-704
B261	
B262-66	B705-09
B267-6F	B70A-12
B270	
B271-75	B713-17
B276-7E	B718-20
B27F	
B280-84	B721-25
B285	B726
B286	B727
B287	B728
B288	B729
B289	B72A
B28A	B72B
B28B	B72C
B28C	B72D
B28D	B72E
B28E	
B28F	B72F
B290	B730
B291	B731
B293	B733
B294	B734

B295	B735
B296	B736
B298	B738
B29A-B7	
	B73A-57
B2B8	B758
B2B9	B759
B2BA-C2	
	B75A-62
B2C3-322	B763-C2
B323-27	
	B692
B328	B693
B32A	B695
B32C	B697
B32E	B699
B330	B69B
B332	B69D
B334	B69F
B336	B6A1
B338	B6A3
B339	B6A4
B33A	
B33C	
B33E	
B340	
B342	
B344	
B346	
	B6A5
	B6A7
	B6A9
	B6AA
	B6AB
	B6AD
	B6AE
	B6AF
	B6B0
	B6B1
	B6B2
	B6B3
	B6B4
B347-4B	
B34C-9B	B496-E5
B39C-EB	B4E6-535
B3EC-43B	B536-85
B43C-45	B586-8F
B446-DD	B590-627
B4DE	B628
B4DF	B629
B4E0	B62A
B4E1	B62B
B4E3	B62D
B4E5	B62F
B4E6	B630
B4E7	B631
B4E8	B632
B4E9	B633

B4EA	B634
B4EB-FE	B635-48
B4ED	B637
B4F1	B63B
B4F3	B63D
B4F4	B63E
B4F5	B63F
B4FF	B649
B501	B64B
B502-08	
	B64C-52
B509	B653
B50A	B654
B50B	B655
B50C	B656
B50D-13	B657-5D
B511	B65B
B514-3B	B65E-85
B51D	(B1B9)
B520	(B1BC)
B522	(B1BE)
B539	(B1D5)
	B67F
B53C	B686
B53D	B687
B53E	B688
B53F	B689
B540	B68A
B541	B68B
B543	B68D
B545	B68F
B547	B691
B549-4F	
B550	
B551	
B1ED	
B552	B1EE
B553	
B1EF	
B554	B1F0
B555-5B	
	B1F1-F7
B55C-9A	B1F8-236
B576	B212
B577	B213
B57B	B217
B57C	B218
B57D	B219
B57F	B21B
B580	B21C
B581	B21D
B583	B21F
B58B	B227
B593	B22F
B59B-D9	B237-75
B5DA-619	B276-B5
B60A	B2A6
B619	B2B5
B61A-709	B2B6-3A5

B67F	(B67F)
B6FA	B396
B70A-F9	B3A6-495
B7FA-FF	
B800	B118
BB01	B119
B802	B11A
B803	B11B
B805	B11D
B807-46	B11F-5E
B817	B12F
B818	B130
B819	B131
B81A	B132
B81C	B134
B81E	B136
B81F	B137
B821	B139
B823-46	B14B-5E
B847	B15F
B848	B160
B84A	B162
B84C-8B	B164-A3
B85C	B174
B85D	B175
B85E	B176
B85F	B177
B861	B179
B863	B17B
B864	B17C
B866	B17E
B868-8B	B180-A3
B88C-CB	B1A4-E3
B89D	B1B5
B89F	B1B7
B8A3	B1BB
B8A6	B1BE
B8CC	B1E4
B8CD	B1E5
B8CE	B1E6
	B1E7
B8D0	B1E8
B8D1	B1E9
B8D2	B1EA
B8D3	B1EB
B8D5-E3	
B8DC	B114
B8DD	B115
B8DE	B116
B8DF	B117
B8E4	B100
B8E6	B102
B8E8-EC	B104-08
B8ED-F1	B109-0D
B8F2-F6	B10E-12
B8F7	B113
B8F8-FF	

APPENDIX D: ROMCALL & RAMCALL loader

When we use a CALL command or an RSX from BASIC, we can pass up to 32 pieces of information to the machine code routine by entering such data as parameters that follow the RSX or CALL instruction. On entry to such a routine the parameters are held in a block of data. This block is arranged with the IX register pointing to the last parameter present; earlier parameters are placed in sequence above this last one, up to the first one we entered. On entry to these routines, the A register holds the number of parameters that have been entered. Each parameter is represented by two bytes which could be:

a) **The address of a string descriptor** — at this address there will be three bytes of data to represent a string expression (a string variable only, in the case of the 464). These three bytes are as follows:
   byte 1: the length of the string
   bytes 2 and 3: the address of the start of the string — note that this address is not the same as that of the string descriptor which points to the length byte above)
b) **The address of the value of a numeric variable** — this variable could be integer or real, so the value will be present there in two or five byte form.
c) **The value of a numeric expression** — this will be in two byte form, even though any part of the expression could be in Real/Floating point (five byte) form.

When we consider firmware calls, the situation becomes even more difficult. With these, any parameters required (called entry conditions) need to be present in specific microprocessor registers and not as a block of data to be picked out when needed. Some calls need no entry conditions and so can be called with no problem (for example CALL &BD19), but usually this is not the case.

There are some calls which only need a value in the A register; providing this value is less than 33, we can take advantage of the fact mentioned earlier that, on entering a routine from BASIC, the A register holds the number of parameters present.

Try CALL &BC0E,0,0 to change the screen mode to MODE 2; then try CALL &BC0E to change it to MODE 0. In the first case, there were 2 parameters present, so A held &02 on entry to the routine and this made it select MODE 2. The second call had no parameters and so A held &00, resulting in MODE 0.

With higher values required in A (up to 32), the command looks pretty unwieldy with all those parameters, but you can still use this method for some calls. By the way, anything can be used here as a parameter — ie a string (6128 only), a string variable, a numeric variable, or a number.

But what of the others. Unfortunately, the remaining registers are set by the Operating System for its needs, and so there is very little scope allowed to us to utilise ROM routines in this way.

The RSX presented here allows each register to be loaded with the required values and has two versions:

|RAMCALL is for using firmware calls (whose entry points are all in RAM)
|ROMCALL is for accessing any useful routines in the various ROMs

Values to go into registers are entered as optional parameters. Both have a similar sequence of parameters. The only difference is that the first parameter for |ROMCALL has to be the ROM select number of the ROM which contains the required routine; to access routines in the Lower ROM, use a number of –1 here. This is followed by the actual address of the routine in ROM. With |RAMCALL, this address (of the 'firmware entry point' in RAM) will be the first parameter.

After these obligatory one or two parameters come the values we wish to place in the various registers. The sequence of registers decided on — see the syntax section below — is not the standard alphabetical one but instead it is on frequency of use.

On entry to an Upper ROM routine, the IY register is given the address of that ROM's reserved work area by the Operating System, so any IY values entered with |ROMCALL may be lost; this option is included in case the Lower ROM, or a RAM routine of your own, needs it.

Register parameters are optional with the proviso that if a certain register's parameter is required, then all those before it must also be entered. Any value will do for these unwanted earlier registers; any later unentered registers will be filled with &0000. The first A register parameter also allows values to be entered to simulate the flags (Carry, Zero, etc). If required, the relevant flag bit value should be multiplied by 256 and this added to the A register value. Note that the values for the B, D and H registers will also need to be multiplied by 256, but not those for C, E and L.

The syntax of the RSXs

|ROMCALL,ROM Select no,address of routine [[[[[[,FA (note the reversal of the usual AF)],HL],DE],BC],IY],IX]
|RAMCALL,address of the firmware call [[[[[[,FA],HL],DE],BC],IY],IX]

Passing register values out of the called routine could have been accommodated by use of the '@' operator with numeric variables, but this would have increased the number of parameters necessary, or have forced the use of variables instead of numeric expressions. To keep the parameter situation simple, an output block at &BEF0 has been used, which maintains the input parameter sequence:

&BEF0	&BEF1	&BEF2	&BEF3	&BEF4	&BEF5	&BEF6	&BEF7	&BEF8/9	&BEFA/B
A	F	L	H	E	D	C	B	IY	IX

These locations may or may not hold valid data depending on the exit conditions of the called routine. The loader routine for the RSXs is given below.

The BASIC Listing

```
10 REM ROM/RAM-CALL Loader  by Bob Taylor  (copyright 1991)
20 MEMORY &7FFF:RESTORE:PRINT:PRINT "Please wait a few seconds"
30 FOR lin=0 TO &B0/8-1:total=0:FOR n=0 TO 7:READ A$
40 byte=VAL("&"+a$):POKE &8000+lin*8+n,byte
50 total=total+byte:NEXT n
60 READ a$:IF VAL("&"+a$)<>total THEN PRINT:PRINT "Error in line"lin*10+110:END
70 NEXT lin:IF PEEK(6)=&80 THEN POKE &80A4,&55:POKE &80A5,&CB
80 PRINT:PRINT "All M/C loaded":PRINT:PRINT "Press S to save M/C as ROMCALL.BIN":
PRINT "or any other key to continue":WHILE INKEY$="":WEND:IF INKEY(60)<>-1 THEN
SAVE "ROMCALL.BIN",B,&8000,&A9
90 PRINT:PRINT "To load & initialise |ROMCALL RSX just use:":PRINT "MEMORY HIMEM
-&A9:a=HIMEM+1:LOAD"CHR$(34)"ROMCALL.BIN"CHR$(34)",a:CALL a":PRINT "with the disc
or tape inserted"
100 END
110 DATA D5,62,6B,36,C9,01,18,00,2BA
120 DATA 09,EB,01,28,00,09,72,2B,1C3
130 DATA 73,44,4D,E1,23,C3,D1,BC,458
140 DATA 52,4F,4D,43,41,4C,CC,52,2DC
150 DATA 41,4D,43,41,4C,CC,00,18,242
160 DATA 80,3D,F6,80,32,FF,BE,E6,508
170 DATA 3F,28,6C,47,2F,C6,08,28,23F
180 DATA 09,30,64,21,00,00,E5,3D,1E0
190 DATA 20,FC,DD,66,01,DD,6E,00,3AB
200 DATA E5,DD,23,DD,23,10,F3,E1,4C9
210 DATA 22,FD,BE,E1,7C,65,6F,E5,4F3
220 DATA F1,E1,D1,C1,FD,E1,F5,3A,671
230 DATA FF,BE,CB,7F,28,13,DD,7E,49D
240 DATA 00,32,FF,BE,3C,3E,CF,28,360
250 DATA 0A,F1,DD,E1,DF,FD,BE,18,56B
260 DATA 0B,3E,C3,32,FC,BE,F1,DD,4C6
270 DATA E1,CD,FC,BE,22,F2,BE,F5,62F
280 DATA E1,65,6F,22,F0,BE,ED,53,4C5
290 DATA F4,BE,ED,43,F6,BE,FD,22,5B5
300 DATA F8,BE,DD,22,FA,BE,C9,3E,574
310 DATA 21,0E,00,21,93,CA,C3,1B,28B
320 DATA 00,00,00,00,00,00,00,00,000
```

APPENDIX E: MISCELLANEOUS

BASIC Deprotection

```
10 REM DEPRO-LOADER copyright Bob Taylor 1989
20 RESTORE 110:PRINT:PRINT "Please wait a few seconds"
30 FOR lin=0 TO &40/8-1:total=0:FOR n=0 TO 7:READ a$
40 byte=VAL("&"+a$):POKE &BE80+lin*8+n,byte
50 total=total+byte:NEXT n
60 READ a$:IF VAL("&"+a$)<>total THEN PRINT:PRINT "Error in line"lin*10+110:END
70 NEXT lin:IF PEEK(6)=&80 THEN POKE &BEB3,&45
80 PRINT:PRINT "All M/C loaded":PRINT:PRINT "Press 'S' to save M/C as DEPRO.BIN":
WHILE INKEY$="":WEND:IF INKEY(60)<>-1 THEN SAVE "DEPRO.BIN",B,&BEB0,&40
90 PRINT:PRINT "To load and initialise DEPRO just enter:":PRINT "LOAD"CHR$(34)
"DEPRO.BIN"CHR$(34)":CALL &BE80":PRINT "in direct command mode with the disc or
tape inserted at the correct place":PRINT "To switch off just enter CALL &BE80,0"
100 END
110 DATA B7,3A,7A,BC,20,1C,FE,C3,424
120 DATA C8,32,BB,BE,2A,7B,BC,22,3F6
130 DATA BC,BE,3E,C3,21,AD,BE,F5,4FC
140 DATA E5,32,7A,BC,22,7B,BC,E1,487
150 DATA F1,C9,FE,C3,C0,3A,BB,BE,5EE
160 DATA 2A,BC,BE,18,EA,CD,A5,BE,4D6
170 DATA F5,AF,32,2C,AE,F1,CD,7A,4E8
180 DATA BC,18,D7,00,00,00,00,00,1AB
```

ROM Lister

```
10 MODE 2:lin=2
20 DATA 11,45,80,0E,00,CD,0F,B9
30 DATA C5,0E,00,C5,CD,0F,B9,CD
40 DATA 1F,80,C1,0C,79,FE,0F,C2
50 DATA 0B,80,C1,CD,18,B9,C9,21
60 DATA 00,C0,06,04,7E,FE,80,C8
70 DATA 12,23,13,10,F7,4E,23,46
80 DATA 0A,FE,80,F2,3C,80,12,03
90 DATA 13,C3,30,80,D6,80,12,13
100 DATA 3E,FF,12,13,C9,00,00,00
110 DATA end
120 add=&8000
130 READ a$:IF a$="end" THEN 160
140 POKE add,VAL("&"+a$):add=add+1
150 GOTO 130
160 PRINT "The ROMs available are:"
170 CALL &8000
180 add=&8045
190 FOR rom=1 TO 15
200 type=PEEK(add):mark=PEEK(add+1)
210 vers=PEEK(add+2):m=PEEK(add+3)
220 add=add+4:name$=""
230 c=PEEK(add)
240 IF c=&FF THEN GOTO 270
250 name$=name$+CHR$(c)
260 add=add+1:GOTO 230
270 add=add+1
280 LOCATE 3,lin:PRINT name$
290 LOCATE 14,lin:PRINT USING "#.";mark
300 PRINT USING "#";vers;:PRINT USING "#";m
310 IF type=0 THEN t$="FOREGROUND":GOTO 330
320 IF type=1 THEN t$="BACKGROUND" ELSE t$=
"EXTENSION"
330 LOCATE 19,lin:PRINT t$;" ROM"
340 IF PEEK(add)=0 AND PEEK(add+1)=0 AND PEEK
(add+2)=0 THEN END
350 lin=lin+1:NEXT rom
```

Snippets

To unerase files — Poke &A701 with &E5, CAT the disc, and try to load the file you want. Save it back to USER 0 if it loads correctly. If it fails to load, it is probably that it has already been written over by another file

To disable ESC — Poke &BDEE with &C9. To restore ESC, poke the same address with &C3

Special Z80 Instructions

DAA — This instruction adds or subtracts six to/from the A register nibbles according to the preceding instruction, and to the states of various flags (the half carry flag, H, is bit 4 of the flags but there are no instructions available to the programmer for testing its contents):

Previous Instruction	C	H	high nibble	low nibble	added to A	C after DAA
ADD ADC INC	0	0	0–9	0–9	&00	0
ADD ADC INC	0	0	0–8	A–F	&06	0
ADD ADC INC	0	1	0–9	0–3	&06	0
ADD ADC INC	0	0	A–F	0–9	&60	1
ADD ADC INC	0	0	9–F	A–F	&66	1
ADD ADC INC	0	1	A–F	0–3	&66	1
ADD ADC INC	1	0	0–2	0–9	&60	1
ADD ADC INC	1	0	0–2	A–F	&66	1
ADD ADC INC	1	1	0–3	0–3	&66	1
SUB SBC DEC NEG	0	0	0–9	0–9	&00	0
SUB SBC DEC NEG	0	1	0–8	6–F	&FA	0
SUB SBC DEC NEG	1	0	7–F	0–9	&A0	1
SUB SBC DEC NEG	1	1	6–F	6–F	&9A	1

EX AF,AF' — This swaps the normal and alternate register AF values; as a result, the flag byte will be filled with another flag value to that before the instruction (which can be retrieved by a further EX AF,AF')

HALT — This instruction doesn't complete until a microprocessor interrupt signal is generated

Rotation and Shift

RLC and RLCA	—	rotate to the left, and move bit 7 into the Carry and into bit 0
RRC and RRCA	—	rotate to the right, and move bit 0 into the Carry and into bit 7
RL and RLA	—	rotate to the left, and move the Carry into bit 0, and move bit 7 into the Carry
RR and RRA	—	rotate to the right, and move the Carry into bit 7, and move bit 0 into the Carry
SLA	—	shift left arithmetical, and reset bit 0 to zero, and move bit 7 into the Carry
SRA	—	shift right arithmetical, and leave bit 7 as it was, and move bit 0 into the Carry
SLL	—	shift left logical, and set bit 0 to one, and move bit 7 into the Carry
SRL	—	shift right logical, and reset bit 7 to zero, and move bit 0 into the Carry
RLD	—	rotate the nibbles to the left, and bits 0 to 3 of A are moved to the contents of HL, and bits 0 to 3 of the contents of HL are moved into bits 4 to 7 of the contents of HL, and bits 4 to 7 of the contents of HL are moved to bits 0 to 3 of A
RRD	—	rotate the nibbles to the right, and bits 0 to 3 of A are moved to the contents of HL, and bits 4 to 7 of the contents of HL are moved to bits 0 to 3 of the contents of HL, and bits 0 to 3 of the contents of HL are moved into bits 0 to 3 of A

The CRTC Registers

To change the value of these registers, the register number should be output on address &BCxx and then the data output on &BDxx (see Appendix B for more details on the CPC port addresses)

R0	Horizontal Total	63	R8	Interlace and Skew	0
R1	Horizontal Displayed	40	R9	Maximum Raster Addr	7
R2	Horizontal Sync Pos.	46	R10	Cursor Start Raster	0
R3	Sync Width	142	R11	Cursor End Raster	0
R4	Vertical Total	38	R12	Start Address (H)	48
R5	Vertical Total Adjust	0	R13	Start Address (L)	0
R6	Vertical Displayed	25	R14	Cursor Register (H)	192
R7	Vertical Sync Position	30	R15	Cursor Register (L)	0

HEXADECIMAL TO DECIMAL CONVERSION CHART

DEC	HEX	*256	DEC	HEX	*256	DEC	HEX	*256	DEC	HEX	*256
1	01	256	65	41	16640	129	81	33024	193	C1	49408
2	02	512	66	42	16896	130	82	33280	194	C2	49664
3	03	768	67	43	17152	131	83	33536	195	C3	49920
4	04	1024	68	44	17408	132	84	33792	196	C4	50176
5	05	1280	69	45	17664	133	85	34048	197	C5	50432
6	06	1536	70	46	17920	134	86	34304	198	C6	50688
7	07	1792	71	47	18176	135	87	34560	199	C7	50944
8	08	2048	72	48	18432	136	88	34816	200	C8	51200
9	09	2304	73	49	18688	137	89	35072	201	C9	51456
10	0A	2560	74	4A	18944	138	8A	35328	202	CA	51712
11	0B	2816	75	4B	19200	139	8B	35584	203	CB	51968
12	0C	3072	76	4C	19456	140	8C	35840	204	CC	52224
13	0D	3328	77	4D	19712	141	8D	36096	205	CD	52480
14	0E	3584	78	4E	19968	142	8E	36352	206	CE	52736
15	0F	3840	79	4F	20224	143	8F	36608	207	CF	52992
16	10	4096	80	50	20480	144	90	36864	208	D0	53248
17	11	4352	81	51	20736	145	91	37120	209	D1	53504
18	12	4608	82	52	20992	146	92	37376	210	D2	53760
19	13	4864	83	53	21248	147	93	37632	211	D3	54016
20	14	5120	84	54	21504	148	94	37888	212	D4	54272
21	15	4376	85	55	21760	149	95	38144	213	D5	54528
22	16	5632	86	56	22016	150	96	38400	214	D6	54784
23	17	5888	87	57	22272	151	97	38656	215	D7	55040
24	18	6144	88	58	22528	152	98	38912	216	D8	55296
25	19	6400	89	59	22784	153	99	39168	217	D9	55552
26	1A	6656	90	5A	23040	154	9A	39424	218	DA	55808
27	1B	6912	91	5B	23296	155	9B	39680	219	DB	56064
28	1C	7168	92	5C	23552	156	9C	39936	220	DC	56320
29	1D	7424	93	5D	23808	157	9D	40192	221	DD	56576
30	1E	7680	94	5E	24064	158	9E	40448	222	DE	56832
31	1F	7936	95	5F	24320	159	9F	40704	223	DF	57088
32	20	8192	96	60	24576	160	A0	40960	224	E0	57344
33	21	8448	97	61	24832	161	A1	41216	225	E1	57600
34	22	8704	98	62	25088	162	A2	41472	226	E2	57856
35	23	8960	99	63	25344	163	A3	41728	227	E3	58112
36	24	9216	100	64	25600	164	A4	41984	228	E4	58368
37	25	9472	101	65	25856	165	A5	42240	229	E5	58624
38	26	9728	102	66	26112	166	A6	42496	230	E6	58880
39	27	9984	103	67	26386	167	A7	42752	231	E7	59136
40	28	10240	104	68	26624	168	A8	43008	232	E8	59392
41	29	10496	105	69	26880	169	A9	43264	233	E9	59648
42	2A	10752	106	6A	27136	170	AA	43520	234	EA	59904
43	2B	11008	107	6B	27392	171	AB	43776	235	EB	60160
44	2C	11264	108	6C	27648	172	AC	44032	236	EC	60416
45	2D	11520	109	6D	27904	173	AD	44288	237	ED	60672
46	2E	11776	110	6E	28160	174	AE	44544	238	EE	60928
47	2F	12032	111	6F	28416	175	AF	44800	239	EF	61184
48	30	12288	112	70	28672	176	B0	45056	240	F0	61440
49	31	12544	113	71	28928	177	B1	45312	241	F1	61696
50	32	12800	114	72	29184	178	B2	45568	242	F2	61952
51	33	13056	115	73	29440	179	B3	45824	243	F3	62208
52	34	13312	116	74	29696	180	B4	46080	244	F4	62464
53	35	13568	117	75	29952	181	B5	46336	245	F5	62720
54	36	13824	118	76	30208	182	B6	46592	246	F6	62976
55	37	14080	119	77	30464	183	B7	46848	247	F7	63232
56	38	14336	120	78	30720	184	B8	47104	248	F8	63488
57	39	14592	121	79	30976	185	B9	47360	249	F9	63744
58	3A	14848	122	7A	31232	186	BA	47616	250	FA	64000
59	3B	15104	123	7B	31488	187	BB	47872	251	FB	64256
60	3C	15360	124	7C	31744	188	BC	48128	252	FC	64512
61	3D	15616	125	7D	32000	189	BD	48384	253	FD	64768
62	3E	15872	126	7E	32256	190	BE	48640	254	FE	65024
63	3F	16128	127	7F	32512	191	BF	48896	255	FF	65280
64	40	16384	128	80	32768	192	C0	49152	256	100	65536