SUCCESS

THE SUPREME CP/M CODE EMULATION SYSTEM

by

Brian Watson

DOCUMENTATION BY

Dr. HELMUT AIGNER & Dr. ANDY AIGNER



Copyright 1988 Digital Precision Ltd.
Published by Freddy Vachha for DP, 222 The Avenue, London E4 9SE

SUCCESS for the Sinclair QL

C O N T E N T S

1. INTRODUCTION
1.1 Why an Emulator?
1.2 Getting Started
1.3 How to Succeed with SUCCESS

2. RUNNING CP/M SOFTWARE

3. SUCCESS AT THE CP/M COMMAND LEVEL
3.1 Filenames
3.2 Peripheral Devices
3.3 Control Codes
3.3.1 Used on input
3.3.2 Used on output
3.4 Operating System Commands
3.5 Standard .COM-File (Transient) Commands
3.6 Special SUCCESS Commands
3.7 Terminal Emulation

4. SUCCESS AT THE MACHINE-CODE LEVEL
4.1 How SUCCESS is Organised
4.2 BIOS
4.2.1 Resources
4.2.2 The Standard Jump Block
4.2.3 BIOS Jump Block Extensions for SUCCESS
4.2.4 Special Extension: DEFKEY
4.2.5 Defining Drive G:
4.3 CP/M-Z80 Disk Structure

SUCCESS for the Sinclair QL

C O N T E N T S

1. INTRODUCTION
1.1 Why an Emulator?
1.2 Getting Started
1.3 How to Succeed with SUCCESS

2. RUNNING CP/M SOFTWARE

3. SUCCESS AT THE CP/M COMMAND LEVEL
3.1 Filenames
3.2 Peripheral Devices
3.3 Control Codes
3.3.1 Used on input
3.3.2 Used on output
3.4 Operating System Commands
3.5 Standard .COM-File (Transient) Commands
3.6 Special SUCCESS Commands
3.7 Terminal Emulation

4. SUCCESS AT THE MACHINE-CODE LEVEL
4.1 How SUCCESS is Organised
4.2 BIOS
4.2.1 Resources
4.2.2 The Standard Jump Block
4.2.3 BIOS Jump Block Extensions for SUCCESS
4.2.4 Special Extension: DEFKEY
4.2.5 Defining Drive G:
4.3 CP/M-Z80 Disk Structure

1. INTRODUCTION

1.1 Why an Emulator?

Whoever wishes to utilise any of the thousands of programs written for the CP/M-80 operating system on the Sinclair QL or CST Thor could either get a third processor - a very expensive option - or, preferably, obtain a Z80 emulator (Z80 being the name of the microprocessor for which CP/M was written). The latter option, though somewhat slower, is reasonably cheap. CP/M, by the way, stands for Control System for Microcomputers; it was developed by Digital Research.

An emulator is a program which will allow something to run on a machine which would normally only work on another machine. SUCCESS (SUpreme Cp/m Code Emulation SyStem) is an emulator for the Sinclair QL that will work in a CP/M (version 2.2) environment. With some extra programming it could, however, be used to emulate any Z80 environment (a Spectrum emulation, argghhhh!). SUCCESS will handle CP/M programs up to a length of 64 Kbytes. (This restriction may be lifted in a later version.) It will only run on a disk-based system.

The program works by taking each Z80 instruction from memory and then executing a 68000 machine-code subroutine depending on the instruction value. Being written in machine code, it is quite fast, permitting an emulation speed of roughly 1.0 MHz on a standard QL (about 1.8 MHz when on EPROM or in expansion memory).

1.2 Getting Started

Do curb your impatience to get SUCCESS up and running right away. Disks have been known to become faulty, and files to be erased by mistake. Murphy's Law decidedly states that it will happen to YOU if you fail to make a backup copy of your SUCCESS disk.

To make a backup copy, type

```
LRUN flp1_clone_bas
```

Following the program prompts, put the master disk in drive 1 and examine your blank disk, checking whether it is write-protected; if necessary, unprotect it. Now press Enter. Once all the essential files are copied, you will be asked whether the file named cpmfiles is to be copied, too. This is merely to reassure you in case you are getting impatient while 1042 sectors are being copied. We suggest you reply 'Y' to the prompt regardless: if you are a novice at CP/M, you can't know yet whether you will need cpmfiles; and if you're not, you'll need it for sure!

When the drives have stopped whirring and the lights gone out, remove your master disk and stow it in a safe place. Only the cloned disk should be used henceforth (except for making new clones in case something happens to the first one). The working copy you have just made is intended to work from drive 1; so put it there now before continuing.

Before continuing with SUCCESS, we strongly recommend that you load the file updates_doc into QUILL and peruse it for stop-press info on improvements not contained in the manual.

1.3 Copyright Notice

SUCCESS, like the rest of DP's programs, is not protected against unauthorised copying and theft. This has obvious advantages for you, in that you can make backups freely and transfer the program suite freely from one medium to another. But the absence of copy protection has a big disadvantage for us: it means that every time you sell or give away a copy of SUCCESS to someone, we lose a sale that would have helped us to fund more development of QL software.

The only part of the package that may be copied freely is the CP/M operating system (version 2.2), which is in the public domain. No cash advantage should be asked for or accepted on its redistribution.

There is no QL software publisher so large and committed as DIGITAL PRECISION. But the flow of new QL software from DP, and the availability of older titles, depends upon the honesty of a circle of buyers.

DIGITAL PRECISION will remain active in the QL market for the foreseeable future, but future products can only appear if people are willing to pay for them.

In effect, against the trend - especially on top-quality products - we have taken a gamble on your honesty. Please don't let us down.

We offer rewards for information which enables action (civil and criminal) to be taken against software pirates, big and small. Of course, we hope to spend our time more productively!

1.3 How to Succeed with SUCCESS

There are several distinct uses you may want to make of SUCCESS. Sorted by level of sophistication, they are:

(a) running any CP/M program in your possession on the QL or Thor;
(b) using CP/M commands, mainly for some things that cannot be done as easily with QDOS and SuperBASIC;
(c) writing and running your own Z80 assembler programs.

Operating instructions for (a) will be found in Chapter 2; for (b), in Chapter 3; and for (c), in Chapter 4. However, this manual cannot be a substitute for a full treatment of the CP/M operating system and its handling. Getting such a book is suggested for level (a), recommended for level (b), and imperative for level (c). Here are two titles you may find useful:

B. Brigham
CP/M Programmer's Encyclopaedia
Que

Peter Gosling
Using CP/M
Macmillan

If you wish to go into Z80 assembler language, one of many excellent tutorials is:

Alan Tully
Z80 Reference Guide
Melbourne House

There are also a number of fine books on Z80 machine code for the ZX Spectrum.

You may further wish to contact a CP/M user group in order to obtain CP/M software, much of which is in the public domain, as well as for advice. The biggest of these in the UK is:

PD Software
90 Braybourne Close
Uxbridge, Middx, UB8 1UJ
Tel: 01 8642633 or 0895 51978

Don't forget to specify the size of disk appropriate to your disk drive!

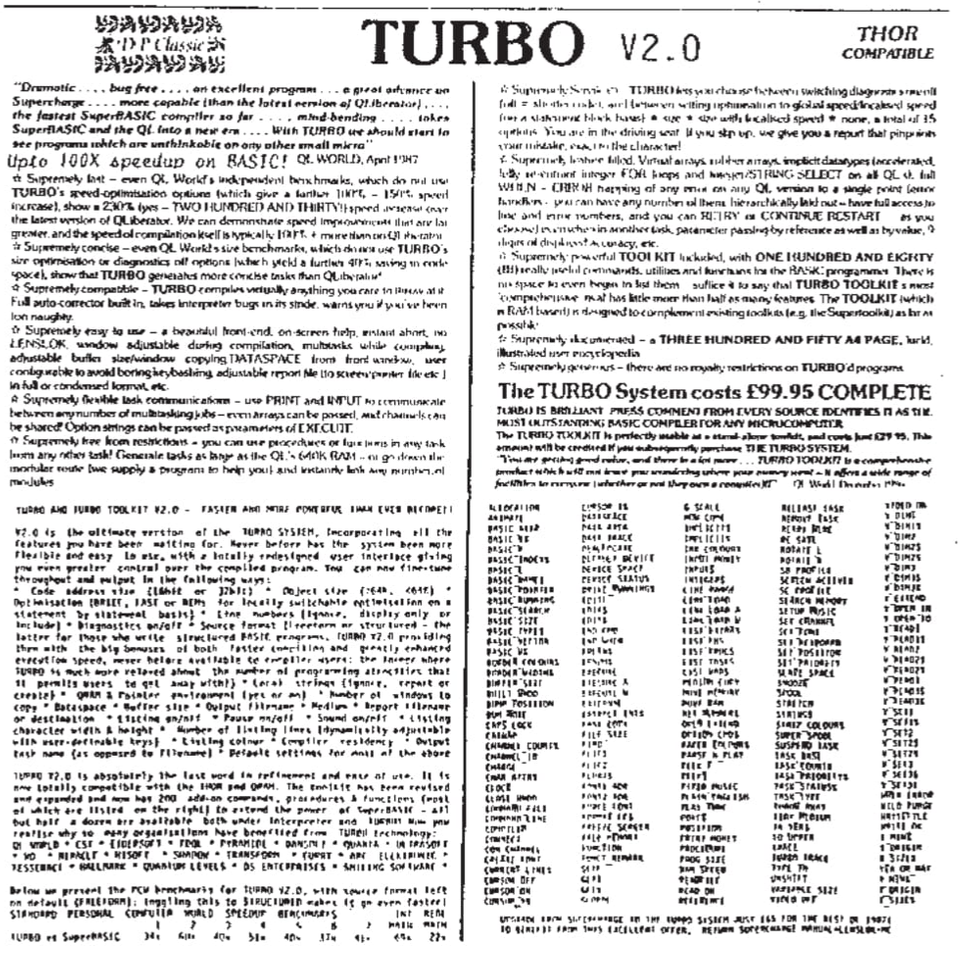
Other sources of CP/M software include:

Grey Matter
4 Prigg Meadow
Ashburton, Devon, TQ13 7DF
Tel: 0364 53499

Shareware
87 High St
Tonbridge, Kent
Tel: 0732 771344

Seltec
Farley Hall, Wokingham Rd
Bracknell, Berks, RG12 5EU
Tel: 0344 863020

OR BUY A COPY OF PERSONAL COMPUTER WORLD OR PRACTICAL COMPUTING: YOU'LL FIND LOTS OF ALTERNATIVE SOURCES.



QPC Classic

TURBO V2.0

THOR COMPATIBLE

"Dramatic . . . . bug free . . . . an excellent program . . . a great advance on Supercharge . . . . more capable (than the latest version of QLiberator) . . . . the fastest SuperBASIC compiler so far . . . . mind-bending . . . . takes SuperBASIC and the QL into a new era . . . . With TURBO we should start to see programs which are unthinkable on any other small micro"

Upto 100X speedup on BASIC! QL WORLD, April 1987

☆ Supremely fast – even QL World's independent benchmarks, which do not use TURBO's speed-optimisation options (which give a further 100% – 150% speed increase), show a 230% (yes – TWO HUNDRED AND THIRTY!) speed increase over the latest version of QLiberator. We can demonstrate speed improvements that are far greater, and the speed of compilation itself is typically 100% + more than on QL liberator!

☆ Supremely concise – even QL World's size benchmarks, which do not use TURBO's size optimisation or diagnostics off options (which yield a further 40% saving in code space), show that TURBO generates more concise tasks than QLiberator!

☆ Supremely compatible – TURBO compiles virtually anything you care to throw at it. Full auto-corrector built in, takes interpreter bugs in its stride, warns you if you've been too naughty.

☆ Supremely easy to use – a beautiful front-end, on-screen help, instant abort, no LENSLOK, window adjustable during compilation, multitasks while compiling, adjustable buffer size/window copying/DATASPACE from front window, user configurable to avoid boring key-bashing, adjustable report file (to screen/printer file etc.) in full or condensed format, etc.

☆ Supremely flexible task communications – use PRINT and INPUT to communicate between any number of multitasking jobs – even arrays can be passed, and channels can be shared! Option strings can be passed as parameters of EXEC/EXT.

☆ Supremely free from restrictions – you can use procedures or functions in any task from any other task! Generate tasks as large as the QL's 640K RAM – or go down the modular route (we supply a program to help you) and instantly link any number of modules.

TURBO AND TURBO TOOLKIT V2.0 – FASTER AND MORE POWERFUL THAN EVER BEFORE!!

V2.0 is the ultimate version of the TURBO SYSTEM, incorporating all the features you have been waiting for. Never before has the system been more flexible and easy to use, with a totally redesigned user interface giving you even greater control over the compiled program. You can now fine-tune throughout and output in the following ways:

* Code address size (16bit or 32bit) * Object size (near, 64K) * Optimisation (BRIEF, FAST or BOTH for locally switchable optimisation on a statement by statement basis) * Line numbers (ignore, display only or include) * Diagnostics on/off * Source format (freeform or structured – the latter for those who write structured BASIC programs, TURBO V2.0 providing them with the big bonuses of both faster compiling and greatly enhanced execution speed, never before available to compiler users; the former where TURBO is much more relaxed about the number of programming atrocities that it permits users to get away with!) * Local strings (ignore, report or create) * QRAM & Pointer environment (yes or no) * Number of windows to copy * Dataspace * Buffer size * Output filename * Medium * Report filename or destination * Listing on/off * Pause on/off * Sound on/off * Listing character width & height * Number of listing lines (dynamically adjustable with user-definable keys) * Listing colour * Compiler residency * Output task name (as opposed to filename) * Default settings for most of the above.

TURBO V2.0 is absolutely the last word in refinement and ease of use. It is now totally compatible with the THOR and QPAC. The toolkit has been revised and expanded and now has 200 add-on commands, procedures & functions (most of which are listed on the right) to extend the power of SuperBASIC – all but half a dozen are available both under Interpreter and TURBO! Now you realise why so many organisations have benefited from TURBO technology:

QL WORLD * CST * EIDERSOFT * FDOL * PYRAMIDE * DANSOFT * QUANTA * INFOSOFT * MD * MIRACLE * MICROPORT * SAMDOS * TRANSDOS * EUROMART * ARK * ELLKAYTEC * TESSERACT * HALLMARK * QUANTUM LEVELS * QS ENTERPRISES * SMILING SOFTWARE *

Below we present the PCW benchmarks for TURBO V2.0, with source format left on default (FREEFORM); toggling this to STRUCTURED makes it go even faster!

STANDARD PERSONAL COMPUTER WORLD SPEEDUP BENCHMARKS

	1	2	3	4	5	6	7	INT MATH	REAL MATH
TURBO vs SuperBASIC	34x	40x	40x	51x	40x	37x	41x	69x	22x

☆ Supremely Speedy – TURBO lets you choose between switching diagnostics on/off (= slightly faster), and between setting optimisation to global speed/localised speed (via a statement/block basis) * size * size with localised speed * none, a total of 15 options. You are in the driving seat. If you slip up, we give you a report that pinpoints your mistake, exactly the character!

☆ Supremely, however filled. Virtual arrays, sub-dim arrays, implicit datatypes (accelerated, fully re-entrant integer FOR loops and integer/STRING SELECT on all QL 0. full WHEN – ERROR mapping of any error on any QL version to a single point (error handler) – you can have any number of them, hierarchically linked out – have full access to line and error numbers, and you can RETRY or CONTINUE/RESTART as you choose) to switches in another task, parameter passing by reference as well as by value, 9 digits of display/8 accuracy, etc.

☆ Supremely powerful TOOLKIT included, with ONE HUNDRED AND EIGHTY (180) really useful commands, utilities and functions for the BASIC programmer. There is no space to even begin to list them – suffice it to say that TURBO TOOLKIT's most comprehensive rival has little more than half as many features. The TOOLKIT (which is RAM based) is designed to complement existing toolkits (e.g. the Supertoolkit) as far as possible!

☆ Supremely documented – a THREE HUNDRED AND FIFTY A4 PAGE, lucid, illustrated user encyclopedia.

☆ Supremely generous – there are no royalty restrictions on TURBO'd programs.

The TURBO System costs £99.95 COMPLETE

TURBO IS BRILLIANT. PRESS COMMENT FROM EVERY SOURCE IDENTIFIES IT AS THE MOST OUTSTANDING BASIC COMPILER FOR ANY MICROCOMPUTER.

The TURBO TOOLKIT is perfectly usable as a stand-alone toolkit, and costs just £29.95. This amount will be credited if you subsequently purchase THE TURBO SYSTEM.

"You are getting good value, and there is a lot more . . . . TURBO TOOLKIT is a comprehensive product which will not leave you wondering where your memory went – it offers a wide range of facilities to everyone (whether or not they own a compiler)!" QL World December 1986

UPGRADE FROM SUPERCHARGE TO THE TURBO SYSTEM JUST £65 FOR THE REST OF 1987! TO BENEFIT FROM THIS EXCELLENT OFFER, RETURN SUPERCHARGE MANUAL+LENSLOK+ME

2. RUNNING CP/M SOFTWARE

1 Obtain the program from a CP/M library, or buy it. Don't pirate.

2 Transfer the disk's contents to QDOS:
- Place your working disk of SUCCESS in drive 1 and the program disk in drive 2
- Boot up the system, or enter LRUN flp1_boot
- You will be asked if it is running on a Thor-20 or a QL/Thor (this is because of some slight incompatibility between the 68000/8 and the 68010/20/30)
- When the A> prompt appears, enter the following:
  - If the format of the library disk is known to be one of the following, enter the appropriate command from the following list:
    - DEFINE AMS-DATA.DRV (if the disk in drive 2 is in Amstrad data format) or
    - DEFINE AMS-SYST.DRV (if it is in Amstrad system format) or
    - DEFINE BBCZ80.DRV (if it is a BBC micro Z80 second-processor disk)
  - If the format of the library disk is not known, obtain it as follows:
    - Enter ANALYSE and follow subsequent prompts
    - When the A> prompt reappears, enter
      RDQDOS DISKFMT.DRV FLP1_DISKFMT_DRV
    - On completion of this command, enter
      DEFINE DISKFMT

  CP/M is case-blind; you may prefer to use lower case.
- Check the disk format by loading and running one of the programs on the disk, making sure that it performs NO WRITING to disk; if the format turned out to be incorrect, writing would corrupt the disk. If the format does not seem to be correct, go back to the ANALYSE command and alter some of the parameters it asks for.

Once the disk format is established as correct, it will be possible to access all files on the program disk under the virtual drive name G:

3 Type G:

4 When the G> prompt appears, follow the instructions for the program in question. In many cases this will involve no more than typing in the program name.

2.3 Terminal Emulation

Most CP/M programs are designed to be used on a system which emulates a terminal, most commonly a VT52. SUCCESS supports most VT52 emulation sequences, viz:

[ESC]A - Moves cursor up one line
[ESC]B - Moves cursor down one line
[ESC]C - Moves cursor right one character
[ESC]D - Moves cursor left one character
[ESC]E - Clears entire screen
[ESC]H - Homes cursor
[ESC]J - Erases to end of screen, including current line
[ESC]K - Erases to end of line
[ESC]Yrc - Moves cursor to (r-32),(c-32)
[ESC]p - Inverse video
[ESC]q - Normal video

3. SUCCESS AT THE CP/M COMMAND LEVEL

3.1 Filenames

A CP/M filename consists of 3 parts: <drive>:<name>.<extension>

where <drive>: stands for one of the letters A to G followed by a colon

Example: A:PIP.COM

Drive assignment is as follows:

```
Drive A - FLP1_    - 512K capacity, 128 files
Drive B - FLP2_    - 512K capacity, 128 files
Drive C - MDV1_    - 100K capacity,  64 files
Drive D - MDV2_    - 100K capacity,  64 files
Drive E - RAM1_    - 200K capacity,  96 files
Drive F - RAM2_    - 200K capacity,  96 files
Drive G - FLP2_    - Format user-definable
```

Drives E and F must previously have been formatted at the SuperBASIC level. Drive G must have been established with the DEFINE command (see sections 2.1 and 4.2.5). Drive organisation can be altered by running the SuperBASIC program cpmconfig_bas.

The <drive>: part may be omitted from a filename, in which case it defaults to the current drive. On start-up A: is made the current drive; this may be changed by entering B: or C: etc.

The <name> part of a filename may consist of between 1 and 8 characters. All characters except space and <>.,::=?*[]_ are permitted (note that underline is illegal, as opposed to QDOS rules), and lower-case letters are automatically converted to upper case on input (this is true throughout CP/M).

The .<extension> part of a filename consists of a dot (full stop, period) and up to three characters as above. It may be omitted, in which case it defaults to an empty string. Commonly used extensions are:

```
ASM - Assembler source file          KEY - function-key definition file
BAK - backup file                    LIB - library file
COM - command file                   MAC - Macro Assembler source file
DAT - general data file              PRN - Assembler list file
DOC - document file                  SUB - task submit file
DRV - raw-disk definition file       TXT - text file
HEX - Intel Hex object file          $$$ - temporary file
```

The use of wildcard filenames is permissible in certain commands. A wildcard filename differs from a simple filename in that both the name and the extension parts the characters ? and * are also permitted, e.g. B:MAIL?.* (covering B:MAIL0.DAT , B:MAIL4.DOC , etc; but not, say, B:MAIL15.TXT).

3.2 Peripheral Devices

The devices that CP/M allows have been assigned to specific QDOS devices as follows:

Device CON:	As CON_480X250A16X3
Device LST:	As SER1HR - changeable by means of the cpm_config program
Device PTP:	User-definable, device asked for
Device RDR:	User-definable, device asked for

These names may occur in various CP/M commands.

3.3 Control Codes

The following control codes are recognised. The ^ sign stands for combination with the CTRL key; e.g., ^X means that CTRL and X must be pressed simultaneously.

3.3.1 Used on input:

^C (first on line:) terminates command execution, causes soft re-boot. Use this if '>' prompt does not appear after an error message
Note: Use CTRL + SHIFT + C to change between processes (like CTRL + C (Thor: Sys Req) under QDOS.

^E ('end line':) starts a new physical line on screen; no change is made to the logical line

^H ('backstep', also backspace key): deletes last character both from the screen and from the input buffer

^J ('line feed') and

^M ('return'; also Enter key): terminates an input line and passes input buffer contents to program

^R ('retype':) appends a hash sign to current input line, abandons this line and retypes current contents of input buffer on next line

^U ('undo:) appends a hash sign to current input line, abandons this line, empties the input buffer and starts a new input line

^X ('cancel':) removes current input line from the screen and empties the input buffer

3.3.2 Used on output:

^I ('horizontal tab'; also Tab key): generates the number of spaces required to take the cursor to the next column number divisible by 8

^P ('printer':) toggles printer (device LST) echoing of screen output. Printer echoing will be switched off by a soft re-boot.

^S ('scroll lock':) toggles screen scrolling

3.4 Operating System Commands

DIR
DIR <drive>:
DIR <filename>
DIR <wildcard> Lists non-system files

ERA <filename>
ERA <wildcard> Deletes files; stops at a write-protected file

REN <newfilename>=<oldfilename>
REN <newfilename> <oldfilename>
Renames a file (unless write-protected)
Note that the new name is given first

SAVE <pages> <filename>
Opens a new file <filename> and stores in it <pages> * 256 bytes, starting from the initial address of the Transient Program Area

TYPE <filename> Copies a file to screen

USER <number> Switches to a different user area (0 on start-up)
Range for <number> is 0 to 15

3.5 Standard .COM-File (Transient) Commands

ASM <filename>
Assembles an 8080 machine-code file. The file has to be created using ED (or any other text file editor). It will produce two files, <filename>.PRN - Printer output and <filename>.HEX. The .HEX file must be put through the LOAD program to produce a .COM file.

DDT Does not work; use DDTZ

DDTZ <filename>
This is a Dynamic Debugging Tool for Z80's. It is a piece of public-domain software which helps in program debugging. It is better than the (not included) standard DDT. The following sub-commands are recognised by the DDTZ program (command lines must not have more than 32 characters, or numeric arguments more than 4 digits):

A<address> ('assemble')
Stores command mnemonics (with hex arguments) starting at <address>

D<start>,<end> ('dump')
Lists memory between <start> and <end> addresses in both hex and ASCII format

F<start>,<end>,<value> ('fill')
Fills memory between <start> and <end> addresses with <value>

G<start>,<end1>,<end2> ('go')
Runs program, starting at address <start> until the command at either <end1> or <end2> is reached

H<value1>,<value2> ('hex compute')
Computes both <value1>+<value2> and <value1>-<value2> in hex (up to 4 digits)

I<filename> ('input')
Selects <filename> for next input operation

L<start>,<end> ('list')
Lists command mnemonics between <start> and <end> addresses

M<start>,<end>,<destination> ('move')
Copies memory block between <start> and <stop> addresses to destination

R<offset> ('read')
Reads a file whose start address is <offset> bytes from the current program address

S<address> ('set')
Shows (and permits modification of) byte value at <address> in hex

T<number> ('trace')
Executes <number> program steps, showing processor status after each step

U<number> ('untrace')
Ditto, but shows processor status after last step only

X ('examine')
Shows processor status (flags, registers, current command)

X<register>
Shows (and permits modification of) the value of <register>

DUMP Does not work; use DDTZ

ED <filename>
Edits a file and produces a backup of its last version under the name <drive>:<name>.BAK. The following sub-ommands are recognised within the ED program:

nA Adds lines of text from original file to text buffer

nW Writes lines of text from the text buffer into the temporary file

nX Writes sections of text into a temporary file

R Inserts text from the temporary file at the text pointer position

R <filename> Inserts text from file at the text pointer position

+/-B Moves text pointer to start or end of text

+/-nC Shifts text pointer by n characters

+/-nL Shifts text pointer by n lines

nF<text> Finds n-th occurrence of string

nN<text> Ditto, but reads new lines from original file as required

+/-n Shifts text pointer and types line of text pointer

+/-nT Types n lines of text

+/-P Types n screen pages and shifts text pointer

I Switches to text mode

I<text>^Z Inserts <text> in command mode

I<text><Enter> Ditto, including end-of-line marker

```
        +/-nD       Deletes n characters
        +/-nK       Deletes n lines
        nS<oldtext>^Z<newtext>^Z
                    Removes <oldtext>, inserting <newtext> instead
        J<searchstring1>^Z<newtext>^Z<searchstring2>^Z
                    Removes text between <searchstring1> and
                    <searchstring2>, inserting <newtext> instead
        E           Ends editing session, makes changes permanent
        H           Ends current session and reopens file for editing
        O           Returns to state of last ED call of file
        Q           Ends editing session, abandons changes
        nM<commands>  Executes <commands> n times
        OV          Shows buffer contents
        +/-V        Toggles line numbering on/off
        +/-U        Toggles upper-case conversion of input on/off
        nZ          Pause for n fiftieths of a second

LOAD    <filename>
                    Converts a file from Intel Hex format into a binary
                    .COM file.  This can subsequently be run under CP/M
                    by entering the <name> part of <filename>

PIP                           ('peripheral interchange processor')
PIP     <destination file>=<source list>
PIP     <destination file>_<source list>
PIP     <drive>:=<wildcard>
PIP     <drive>:_<wildcard>
PIP     <filename>=<drive>:
PIP     <filename>_<drive>:
                    Copies  information from  one peripheral  device to
                    another. Note that the  destination is always given
                    first.  <source  list> may  be one or  more devices
                    and/or filenames (comma-separated).  If no argument
                    is given, an  asterisk prompt will be  given at the
                    start of each line, and arguments as above can then
                    be entered line by  line.  Pressing ENTER will quit
                    this mode.
                    PIP is  often  used  to  copy  files  from one disk
                    drive to another; e.g. when a raw disk is defined,
                    PIP A:=G:*.*
                    copies all the files to the correct-format disks,
                    or
                    PIP LST:=PROGRAM.PRN
                    produces a printed copy of PROGRAM.PRN
```

STAT <command line>
Supplies information about drives. The STAT program recognises the following sub-commands:

STAT
Supplies information on write-protection of drives and free space thereon

STAT <drive>:
Supplies information on free space on the disk in <drive>

STAT <filename>:
STAT <wildcard>:
Supplies information on size of files, file characteristics and free space on disk

STAT <filename>$<characteristic>
STAT <wildcard>$<characteristic>
Sets or unsets file characteristics
<characteristic> may be:
DIR - make into non-system file
R/O - read only
R/W - read/write
SYS - make into system file

STAT <device>:=<defined device>:
Assigns a pre-defined device name to one of the four device names (BAT/CRT/TTY/UC1 to CON, CRT/LPT/TTY/UL1 to LST, PTP/TTY/UP1/UP2 to PUN, PTR/TTY/UR1/UR2 to RDR)

STAT <drive>:=R/O
Write-protects <drive>

STAT DEV:
Lists device assignments

STAT DSK:
STAT <drive>:DSK
Tabulates properties of currently active disks

STAT USR: Tabulates user areas on disk and indicates current user number

STAT VAL: Tabulates valid STAT sub-commands and arguments

SUBMIT <filename>
SUBMIT <filename> <argument list>
Loads and executes a file consisting of CP/M commands. The command file may include variables of the form $1, $2 etc, which will be replaced with the elements of the (space-delimited) argument list. The command file may contain comment lines starting with any of the characters <>.,;:=?*_ or with either a left or right curly bracket.

SYSGEN
does not work; nor is it needed, the operating system being already on the working disk

XSUB
(in command files only:)
Loads and executes one line from the temporary file created by SUBMIT whenever the running program expects console input

3.6 Special SUCCESS Commands

ANALYSE

This utility, written in 68000 code, is to be used in conjunction with the DEFINE command (see below) and raw-disk read-write routines. It will attempt to extract as many details about a disk format as possible, starting with the disk sector size and density. This takes APSOLUTELY AGES (there are lots of possibilities to be exhausted) so don't jump to conclusions about a machine crash. When ready with this part, the routine will ask you several questions. If you are not sure of some of the parameters, guess. The utility will create a QDOS file named FLP1_DISKFMT_DRV, which you must copy to the CP/M file, using the RDQDOS utility (see below). Note that ANALYSE will not be able to fully recognise some of the more obscure CP/M format variants: if it can't, it will fill in whatever details/parameters it can and will leave the rest for you to manually edit-in.

CAT
CAT <devname>

Displays the directory of a QDOS device, e.g.
CAT FLP2_
The default is FLP1_

DEFINE <definefile>

Defines the raw-disk drive, G:. The file specified contains information about the disk that is to be used. The default extension is '.DRV' . A few files for the more popular Z80 disk types will be found on the disk, viz.

AMS-DATA.DRV - Amstrad data-format disks
AMS-SYST.DRV - Amstrad system-format disks
BBCZ80.DRV - BBC micro Z80 2nd processor disks

These files show the exact format that the data should be in. When defining a .DRV file for a new type of drive, refer to the above versions. They all assume that the relevant size of disk drive is connected to (and defined as) FLP2_.

DEFKEY <keynum>,'string'
DEFKEY <keynum>
DEFKEY <keyfile>

Defines a function key. If there is no string, the definition will be deleted. Control characters within the string are defined as \<chr> (<chr> standing for any upper-case character). If a filename is specified, an attempt is made to load this file, the default extension being '.KEY'. The file holds a series of lines containing a key number and a string definition each. It can be created using ED. The key numbers are:

Key	Normal	Ctrl	Shift	Ctrl+Shift
F1	232	233	234	235
F2	236	237	238	239
F3	240	241	242	243
F4	244	245	246	247
F5	248	N/A	250	251

(On the Thor F6 to F10 correspond to Shift + F1 to F5 respectively.)

Arrows Up	Down	Left	Right
224	225	226	227

```
e.g.: DEFKEY 232,'DIR A:\M'
      DEFKEY 236,''
      DEFKEY DBASE.KEY
```

A number of key assignment files are already provided, viz.

```
DBASE.KEY
DEFAULTS.KEY
KEYPAD.KEY
```

DELQDOS <qdosname>

Deletes a QDOS file after asking for confirmation, e.g.
DELQDOS FLP1_CPMSYSTEM

FORMAT

Asks for the drive to be formatted and then creates the CPMFILES file on it, provided the medium has been previously been formatted at the SuperBASIC level (this is for safety reasons). Also performs system regeneration on drive A: and B:.

RDQDOS <cpmname> <qdosname>

Copies a QDOS-format file to a CP/M format file. If the CP/M file exists or the QDOS file does not exist, the copy will be aborted.

e.g.: RDQDOS INTRO.DOC FLP1_INTRO_LIS

SETBAUD <baudrate>

Selects the serial-port transmission and reception rate, which should be one of:
150, 300, 600, 1200, 2400, 4800, 9600, 19200 (xmit only). If an invalid baud rate is typed, an error is given and the baud rate left as it was.

WRQDOS <cpmname> <qdosname>

Copies a CP/M-format file to a QDOS-format file. If the CP/M file does not exist or the QDOS file does, the copy will be aborted.

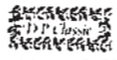
e.g.: WRQDOS BOOT.680 FLP1_BOOT

3.7 Terminal Emulation

Most CP/M programs are designed to be used on a system which emulates a terminal, most commonly a VT52. SUCCESS supports most VT52 emulation sequences, viz:

[ESC]A - Moves cursor up one line
[ESC]B - Moves cursor down one line
[ESC]C - Moves cursor right one character
[ESC]D - Moves cursor left one character
[ESC]E - Clears entire screen
[ESC]H - Homes cursor
[ESC]J - Erases to end of screen, including current line
[ESC]K - Erases to end of line
[ESC]Yrc - Moves cursor to (r-32),(c-32)
[ESC]p - Inverse video
[ESC]q - Normal video

3D PRECISION



3D PRECISION is a very classy 3-D system with STEREO ("Swift Translation and Elegant Rotation of Elements and Objects"). It is two programs in one. A full-feature, menu-driven 3D DESIGNER, with continuous, preprogrammed and coordinate-based drawing, with pan, scroll, translate, rotate about any axis (not just the x-y-z axes, and not just in awkwardly big steps!), recolour, rescale, recentre, zoom, edit/delete (points, lines and "positions"), element reposition, stretch, automirror, change perspective (viewing distance and viewing angle) by moving and rotating the "camera". Movement is multispeed. Fully rubber-banded autogeometrics are provided. Defaults are redefinable via a configurator - this is very flexible, even the keys to be used for various manipulations may be reassigned!!. Any shape can be coped with - no requirement for any regularity / uniform variation of cross section. All modes, screen colours, ink colours, windows etc are supported, and plotting can be done either in normal mode and in XOR. Comprehensive file management utilities are supplied. Screens (ie; 2-D projections) can be output to printer (any EPSON compatible) or file for subsequent editing or adjusting with (say) EYE-Q: 2-D to 3-D and back!
The second program is an easy-to-use 3D TOOLKIT, adding hosts of ultra high speed commands to SuperBASIC. Many examples are provided in the excellent manual. You too can now achieve the absolutely spectacular 3-D animation you see on TV, without being an expert programmer or having any knowledge at all of machine code. The rapidity of line-drawing is to be seen to be believed, the screen's speed is amazing! Even the most complex 3-D special effects can be achieved, as well as technical or artistic illustration and manipulations.
Commands added by the toolkit to SuperBASIC for use in your BASIC or compiled programs:
GBACKPLANE / GCAMADDR / GCENTEROBJ / GCHANGEOBJ / GCLEAR / GCLOSE / GCLS / GDATAADDR / GDEFINEOBJ / GDRAWTYPE / GERROR / GFACTOR / GFINDDOT / GFINDLINE / GFINDMAX / GFINDPOS / GINK / GINSERTOBJ / GLOADOBJ / GMAKE / GMARKDOT / GMARKLINE / GMARKOBJ / GMARKPOS / GMODE / GMOVEPOS / GMOVE / GNOOBJMAX / GNOOBJ / GOBJADDR / GOBJMAX / GOVER / GPAPER / GPLACE / GPLOT / GREMOVEDOT / GREMOVELINE / GREMOVEOBJ / GREMOVEPOS / GROTATEPOSX / GROTATEPOSY / GROTATEPOSZ / GROTATEPOS / GROTATEX / GROTATEY / GROTATEZ / GROTATE / GRXVALUE / GRYVALUE / GRZVALUE / GSAVEOBJ / GSCALE / GSETDOT / GSETLINE / GSETPOS / GSIZEOBJ / GSTART / GSWAP / GTRAPERROR / GWINDOW
Routines supplied for assembly language users for use in M68000 programs:
ADDRCAMERA / ADDRCAMOBJ / ADDROBJECT / CENTEROBJECT / CHANGEOBJECT / CLEAROBJECT / CLOSE / CLS / CONV16 / DEFINEOBJECT / DOT3D / DOT / DRAWTYPE / FINDMAX / INK / INSERTOBJECT / LINE3D / LINE / LOADOBJECT / MAKEOBJECT / MARKOBJECT / MODE / MOVECAMERA / MOVEOBJECT / MOVEPOS / OVER / PAPER / PLACE3D / PLACE / PLOTFREE / PLOTOBJECT / REMOVEDOT / REMOVELINE / REMOVEOBJECT / REMOVEPOS / ROTATE16 / ROTATECAMERA / ROTATEINIT16 / ROTATEOBJECT / ROTATEPOSX / ROTATEPOSY / ROTATEPOSZ ROTATEPOS / ROTATEX16 / ROTATEY16 / ROTATEZ16 / SAVEOBJECT / SCALE / SIZEOBJECT / START / SWAPOBJECT / WINDOW / WORKFREE
Default values and parameters accessible to assembly language users:
ADDREXECLIST / AFTERFILE / BASICDEF / CAMADDR / CHANNEL / COLOUR / COMMAND / DOTTYPEADDR / DOTTYPE / DRAWSTAK / ERRORNO FILEPOS / FLAGS / GVARCAMCENX / GVARCAMCENY / GVARCAMCENZ / GVARDISTX16 / GVARDISTX / GVARDISTY16 / GVARDISTY / GVARDISTZ16 / GVARDISTZ / GVARDOTTAB / GVARFLAG / GVARHLIM / GVARLINETAB / GVARNDM / GVARND / GVARNLM / GVARNL / GVARNPM GVARNP / GVAROBJLEN / GVARPOSTAB / GVARSF / GVARSIZE / GVARSX / GVARSY / GVARSZ / GVARX1 / GVARX2 / GVARX16 / GVARXA / GVARXLH / GVARXL / GVARX / GVARY1 / GVARY2 / GVARY16 / GVARYA / GVARYLH / GVARYL / GVARY / GVARZ16 / GVARZA / GVARZL / GVARZ / LINETYPEADDR / LINETYPE / MEMTOP / MODE / NOOBJMAX / NOOBJN / OBJMAX / OVER / POSFLAG / POSINFO / POSITION / POSXD / POSX / POSYD / POSY / POSZD / POSZ / WORKTAB / X / Y
The system provides facilities for artistic and technical drawing and modelling in 3-D, with excellent aids for conceptualisation. It also gives you the power to achieve all the effects and controls in your own BASIC, DIGITAL C or assembler programs. 3D PRECISION costs just £49.95 complete with a large and lucid A4 manual.

4. SUCCESS AT THE MACHINE-CODE LEVEL

4.1 How SUCCESS Is Organised

The emulator consists of two parts:

- the Z80 interpreter, which does all the hard work;
- the CP/M operating system proper, which comes in three sections:
  - BIOS (Basic Input/Output System), i.e. the hardware-dependent piece of code, which is invoked when a Z80 IN or OUT instruction is used and does all the communication between the Z80 (pseudo) and 68000 processors. It is this part that would have to be altered if any other system but CP/M was to be emulated
  - BDOS (Basic Disk Operating System), which utilises BIOS to provide file handling
  - CCP (Console Command Processor), the part the user communicates with

When the Emulator is loaded, it will search for a file called BIOS_CDE on drive FLP1. If this (68000 code) file is not found, an error is given and Z80 interpretation is aborted.

Once BIOS is loaded and executing, it will attempt to open a file called FLP1_CPMFILES. If unsuccessful, it will wait indefinitely, requiring the QL to be reset. On successfully opening the file, BDOS and CCP are loaded into memory. This done, CP/M will be executed.

The CCP - This is what the user communicates with

4.2 BIOS

4.2.1 Resources

BIOS for this system consists of a series of Z80 OUT instructions and some data needed within the Z80 environment. The OUT instructions are used to "hook" out to the 68000 processor. When an OUT is executed, a piece of 68000 code (held in the file BIOS_CDE on disk) takes over for maximum speed and easy interfacing to the QDOS operating system.

BIOS provides the user with the seven disk drives described in section 3.2.

Before Drive G can be used, it has to be configured. This may be done at machine-code level through one of the BIOS extensions as shown later.

For the assignment of the devices that CP/M allows to specific QDOS devices see section 3.2. The LST device can be redefined using the cpmconfig_bas program.

4.2.2 The Standard Jump Block

The jump block for BIOS is of standard form with three new jumps added to allow utilisation of QDOS devices and access to the 68000. Thus the jump block is organised as follows:

```
Initialisations:
          $00 - hard restart
          $01 - soft re-boot
I/O channels:
          $02 - console status
          $03 - console input
          $04 - console output
          $05 - printer output
          $06 - paper tape punch
          $07 - paper tape reader
Drive control:
          $08 - set read/write head to track 00
          $09 - select drive
          $0A - select track
          $0B - select sector
Disk I/O:
          $0C - set data buffer address
          $0D - read sector
          $0E - write sector
Other functions:
          $0F - inquire printer status
          $10 - translate record number
BIOS extensions(available in SUCCESS only):
          $11 - device allocation
          $12 - byte/string device I/O
          $13 - execute 68000 code in Z80 memory
          $14 - define function keys
```

4.2.3 BIOS Jump Block Extensions for SUCCESS

QDOS$ALLOC- BIOS call $11

This routine allows a single device to be allocated or machine resources to be set up. In all cases except SETBAUD, HL points to the filename and MUST BE EVEN.

The Accumulator contains a key, one of:-

```
 0 - IO$CLOSE     - Close the currently opened file
 1 - IO$OPEN      - Open old file for reading and writing
 2 - IO$OPENNEW   - Open new file for writing only
 3 - IO$OPENDIR   - Open device directory
 4 - IO$DELETE    - Delete file without confirmation
 5 - SETBAUD      - Set serial-port baud rate (in HL)
```

On return the Accumulator contains either 0 (if successful) or the standard QDOS error keys.

Note: The channel is closed on a soft or hard system reboot.

QDOS$UTIL - BIOS call $12

This routine allows data to be transferred to and from a device. On entry the Accumulator should contain:-

```
0 - IO$FBYTE - Read byte from device (returned in register C)
1 - IO$FSTRG - Read string of characters:
               HL = Base address
               BC = Length (amount read on return)
2 - IO$SBYTE - Write byte to device (in register C)
3 - IO$SSTRG - Write string of bytes:
               HL = Base address
               BC = Length (amount written on return)
```

On return the A register contains either 0 (if successful) or the standard QDOS error key.

GO68000 - BIOS call $13

This routine will execute a piece of 68000 code in Z80 memory. The user must ensure that 68000 registers A2 to A6 and D3 to D7 are not modified.

```
HL - Base of program
BC - $4AFB (identification key)
```

If the HL register has an odd value or the BC register does not contain the identification key, a message will be given and the system will automatically perform a soft reboot.

The 68000 registers have the following values:

```
D3.W - DE registers (read only)
D4.W - BC registers
D5.W - HL registers
D6.B - F  register
D7.B - A  register
A0   - Channel ID of main display
A1   - Pointer to Z80 execution loop
A2   - Pointer to internal data
A3   - Z80 stack address
A4   - Base of workspace
A5   - Base of Z80 memory
A6   - Next instruction to execute
```

To get the value of the Z80 stack pointer, subtract A5 from A3. Since any modification of registers A1-A6 would be fatal, they are preserved before the routine is called and restored afterwards.

All of the BIOS extensions need not be called using the jump block. Like all BIOS routines, they can be called using the Z80 OUT instruction.

For example: OUT ($11),A will execute the QDOS$ALLOC routine.

4.2.4 Special Extension: DEFKEY

Another extension has been added but is not available from within BIOS and MUST therefore be called using an OUT instruction:

DEFKEY - OUT ($14),A

This routine will allow a function key to be defined as a string of characters. The key to be defined is held in the accumulator. The key values are given in section 3.6.

The HL register pair should point to an area of memory which contains the definition (0 if the definition is to be deleted). The definition can be any valid characters finished by a NULL (code 0) character. The maximum length of the definition should be no more than 128 characters, any more being normally ignored.

4.2.5 Defining drive G:

When the DEFKEY routine is called with A=255, drive G is to be defined. HL points to an area of memory containing a definition (if HL=0, the definition is deleted). Drive G can only be used after it has been defined (else BDOS will return a "Bdos Err On G: Select"). To use this function, you will need to know a lot about the format of the disks to be read or written to.

WARNING! Special care is necessary if drive G: is to be used for raw-disk writing (as opposed to read-only operations). If not correctly set up in this mode, a QDOS disk may easily have its format destroyed. Do test it with a blank disk first - Digital Precision will accept no responsibility for any data corruption caused by use of this facility. Before using a disk in drive G:, press CTRL-C to ensure that it is correctly logged on. Similarly, if data on the disk has been altered, press CTRL-C BEFORE you remove it to ensure that the data is correctly written to it.

Note that the user also has to define the disk parameter block (the parameter block for drive G: is located at $EADC in Z80 memory, using the standard CP/M environment). The disk parameter header is set for this drive and cannot be altered (nor is there any reason why it should).

The disk parameter header contains the following data:

SPT (2 bytes) - Logical sectors per track (each sector is 128 bytes long)
BSH (1 byte) - Block Shift Factor, ie; that power of two which gives the number of logical sectors in each block on disk (2 raised to the power BSH = number of logical sectors per block)
BLM (1 byte) - Block allocation mask = BSH - 1
EXM (1 byte) - Extent mask, ie. the number of extents per directory entry
DSM (2 bytes) - Number of blocks that are physically present on disk
DRM (2 bytes) - Number of directory entries per disk
AL0 (1 byte) and AL1 (1 byte) - Initial allocation table contents (so that directory information will not be overwritten)
CKS (2 bytes) - Number of checked directory entries
OFF (2 bytes) - Sector offset, ie. number of tracks to be missed initially

When the routine to define drive G is called, HL points to a raw-disk parameter block. This contains the following data:

RAW$DRV (1 byte) - Drive number to be used
= 1 for FLP1, 2 for FLP2, 3 for FLP3, 4 for FLP4
RAW$PSS (1 byte) - Physical sector size
= 0 for 128, 1 for 256, 2 for 512, 4 for 1024 bytes.
RAW$NPS (1 byte) - Number of physical sectors
RAW$NPT (1 byte) - Number of physical tracks
RAW$DSF (1 byte) - Number of steps in between tracks-1
This is to allow 40-track disks to be read/written to in an 80-track drive (in that case it would be 1, normally 0)
RAW$FLG (1 byte) -
If bit 0 = 1, this is a double-density disk
If bit 1 = 1, this is a double-sided disk (not used)
If bit 2 = 1, then side 2's track numbers are reversed (this is the case with BBC disks)
RAW$WKSPC (2 bytes) - Pointer to a work space, which should be the same size as a single physical sector
RAW$LPSMAP (NPS bytes) - logical-to-physical sector map - included because some disks have the sectors interleaved to make disk disk operation faster, but required even if there is 1:1 sector interleaving

Since this is an exceptionally useful feature (for extracting programs from other CP/M disks), a utility program has been included to help configure drive G:. This utility (called DEFINE, see section 4.2) requires a file containing parameters, viz. the absolutely essential details needed for most of the CP/M formats (any weird formats would require a machine-code defining program). Some of the more common definitions are supplied as files: the BBC CP/M-format disks (double-sided, 80-track), Amstrad CP/M-format disks (both data- and system-format disks, single-sided, 40 tracks; this assumes that a 40-track 3" drive is plugged in).

The file contains the following data:-
(Note: Comments are preceded by an '!', and there must be no padding spaces at the start of a line.)

```
!
<number> ! SPT, logical sectors per track
<number> ! BLS, block size: = 1 for  1024-byte blocks,
                            = 2 for  2048-byte blocks,
                            = 3 for  4096-byte blocks,
                            = 4 for  8192-byte blocks,
                            = 5 for 16384-byte blocks.
<number> ! DSM, directory storage mask: -1
<number> ! ALV, allocation vector: address of a scratch pad area to be
           used by BDOS to keep disk storage allocation information.
<number> ! OFF, track offset
!
! Now the data needed by raw-disk read/write routines
!
<number>          ! DRV, drive to be raw-read/written to
<number>          ! PSS, physical sector size
<number>          ! NPS, number of physical sectors
<number>          ! NPT, number of physical tracks (usually 40 or 80)
<number>          ! DSF, disk stepping factor
<number>          ! FLG, see above for values
!
! Now follow with a list of the order of sectors on the disk
!
<number>, NPS numbers
```

Look at the *.DRV files on the master disk for examples of what they should look like.

4.3 CP/M-Z80 Disk Structure

BIOS utilises a large QDOS-format file called CPMFILES, which contains all of the CP/M files except those on an active drive G:. This permits many separate devices to contain the CP/M programs.

The CPMFILES file contains each 128-byte logical sector in logical order. Each logical track holds 20 logical sectors, with a varying number of logical tracks (210 on a 512K drive).

Drive capacity could have been increased, but then most freely available CP/M disks could not be easily converted to run on the system.

I N D E X

A 3.5
A:, A> 2,3.6
ADDRESS 3.5
AMSTRAD DISKS 3.6,4.2.5
ANALYSE 2,3.6
ASM 3.1,3.5
ASSEMBLE 3.5
ASSEMBLER FILES 1.3,3.1,4

B 3.5
B: 3.6
BACKSPACE KEY 3.3.1
BACKSTEP 3.3.1
BACKUP 1.2
BAK 3.1,3.5
BAUD RATE 3.6
BBC DISKS 3.6,4.2.5
BDOS 4.1
BIOS 4.1,4.2

C 3.1,3.5
CANCEL. 3.3.1
CCP 4.1
CLONE_BAS 1.2
COM 3.1
COMMANDS 1.3,3,3.5
CON 3.2
CONTROL CODES 3.3
COPYING 1.2,3.4,3.5
CP/M 1.1,1.2,4.1
CPM_CONFIG_BAS 3.1,4.2.1
CPMFILES 3.6,4.1

D 3.1,3.5
DAT 3.1
DBASE.KEY 3.6
DDT 3.5
DDTZ 3.5
DEBUGGING 3.5
DEFINE 2,3.6,4.2.5
DEFAULTS.KEY 3.6
DEFKEY 3.6,4.2.3
DELETION OF CHARACTERS 3.5
DELETION OF FILES 3.4
DELQDOS 3.6
DEVICES 3.2,4.2.1
DIR 3.4
DIRECTORY 3.4,3.6
DISK-BASED SYSTEM 1.1
DISK STRUCTURE 4.3
DOC 3.1
DRIVES 3.1
DRV 3.1,3.6
DUMP 3.5

E 3.1,3.5
ED 3.5
EDITOR 3.1,3.5
EMULATOR 1.1
END LINE 3.3.1
ENTER KEY 3.3.1
ERA 3.4
EXTENSION 3.1

F 3.1,3.5
FILENAMES 3.1
FORMAT 3.1,3.6
FUNCTION KEYS 3.6

G 3.1,3.5
G:, G> 2,3.1,3.6,4.2.1,4.2.5
GO68000 . . . . . 4.2.3

H 3.5
HEX 3.1
HORIZONTAL TAB 3.3.2

I 3.5
INSERT. 3.5
INTERPRETER 4.1

JUMP BLOCK 4.2.2,4.2.3

K 3.5
KEY. 3.1,3.6
KEYPAD.KEY 3.6

L 3.5
LIB 3.1
LINE FEED 3.3.1
LINE NUMBERING 3.5
LOAD 3.5

M 3.5
MAC 3.1
MACHINE CODE. 4

NAME 3.1
NEW LINE 3.3.1

O 3.5
OUT INSTRUCTION 4.1.1,4.2.3
OV 3.5

PARAMETER BLOCK 4.2.5
PAUSE 3.5
PERIPHERAL DEVICES 3.2,4.2.1
PIP 3.5
PIRATING 1.2
PRINTER 3.1,3.2,3.3.2
PRN 3.1
PROGRAMS 1.3
PTP 3.2

Q 3.5
QDOS$ALLOC 4.2.3
QDOS$UTIL 4.2.3

R 3.5
RAW-DISK READ/WRITE 3.5,3.6,4.2.5
RDQDOS 3.6
RDR 3.2
REFERENCE BOOKS 1.3
REGISTERS 4.2.3
REN 3.4
RETURN 3.3.1
RETYPE 3.3.1

S 3.5
SAVE 3.4
SCREEN PAGES 3.5
SCROLL LOCK 3.3.2
SETBAUD 3.6
SINGLE STEPPING 3.5
SOFT REBOOT 3.3.1
SOFTWARE 1.3
STAT 3.5
SUB 3.5
SUBMIT 3.5
SYSGEN 3.1
SYSTEM COMMANDS 3.4

T 3.5
TAB 3.3.2
TEMPORARY FILE 3.5
TERMINAL EMULATION 3.7
TEXT BUFFER 3.5
TEXT MODE 3.5
TRANSIENT COMMAND 3.5
TXT 3.1
TYPE 3.4

U 3.5
UNDERLINE 3.1
UNDO 3.3.1
UPPER CASE 2,3.5
USER 3.4

V 3.5
VT52 . . . . . 3.7

W 3.5
WRQDOS 3.6

X 3.5
XSUB 3.5

Z 3.5

68000 PROCESSOR 4.1,4.2.1,4.2.3
$$$ 3.1
* 3.1
? 3.1
^ 3.3.1,3.5