DESIGNS DEPARTMENT

DESIGNS DEPARTMENT HANDBOOK

No. 2.465(82)

ZELDA Development System

EP1M/27

Users' Handbook

Part 2 - Firmware

BRITISH BROADCASTING CORPORATION
ENGINEERING DIVISION

Issue 1
23/3/83

DESIGNS DEPARTMENT HANDBOOK

No. 2.465(82)

ZELDA Development System

EP1M/27

Users' Handbook

Part 2 - Firmware

.....
(D. C. Savage)
for Head of Designs Department

Written by: R. T. Russell

Issue 1
23/3/83

DESIGNS DEPARTMENT HANDBOOK

No. 2.465(82)

ZELDA Development System

EP1M/27

Users' Handbook

Part 2 - Firmware

CONTENTS

DESIGNS DEPARTMENT HANDBOOK

No. 2.465(82)

ZELDA Development System

EP1M/27

Users' Handbook

Part 2 - Firmware

1. Introduction

ZELDA (Zeus Editor, Loader, Disk operating system and Assembler) is a Z80-based software development system built from Zeus modules. The system is a microcomputer with twin 8-inch floppy disks, 49 Kbytes of memory, keyboard, VDU and printer. It may be used for a variety of applications other than Z80 assembly language software development by using programs supplied on disk with the system.

A significant proportion of the operating system firmware, viz. the Editor, Z80 Assembler and Loader, is closely based on the ASMB-80 system supplied by Mostek Corporation, with whom the Copyright rests. No part of this software may be incorporated into any BBC equipment which might be licenced for manufacture or sale outside the Corporation.

2. Documentation

The Zelda System Users' Handbook is divided into four parts. These are:

Part 1 - Hardware 2.464(82)

Describes the configuration of the sub-units, user I/O connections, how to expand the system, and is supplied with the Handbooks of all sub-units.

Part 2 - Firmware (this document)

Describes the resident Monitor, Text Editor, Z80 Mnemonic Assembler, Relocating Linking Loader and Peripheral Interchange Program, and explains how to use them.

Part 3 - System Utilities 2.466(82)

Describes the utility programs supplied with the system on floppy disk.

Part 4 - Writing Applications Software 2.467(82)

Describes in detail the resident routines available to the user, and how to write programs for use on ZELDA.

Other relevant documents include:

Zeus System EDI	EDI 10412
Zeus Users' Manual	DDTM
A Modular 8-bit Microcomputer	DDTM 2.447(80)
Automatic Fault Detection	DDTM 2.448(80)
Equipment with Customer Options	DDTM 2.449(80)
BBC Code of Practice in the use of PROMs	DDTM 1.155(80)

3. Nomenclature

References will be made in this handbook to "devices" and to "filenames". A "device" is an input/output peripheral for which a software driver is provided and to which a two-letter mnemonic has been allocated (e.g. TK:, the terminal keyboard). Some devices, notably the floppy disk (DK:), have a file structure and require a "filename" to identify a particular unit of data (file).

The general format of a device/filename specification is as follows:

DEV:FILNAM.E

Where:

DEV: is a device mnemonic. Valid devices are DK0: (the left-hand disk drive), DK1: (the right-hand disk drive), TK: (the keyboard), TT: (the VDU display), VP: (the "video pager"), PT: (the parallel printer driver) and BB: (a "byte bucket") plus any additional devices provided by the user with a mnemonic and a suitable software driver. If the device is omitted, DK0: is assumed. DK: is the same as DK0:. Details of device drivers are given in Appendices 4 and 5.

FILNAM is a file name of six or less characters (if more than six characters are entered the filename is truncated to the first six). For devices other than DK0: and DK1: the filename is usually irrelevant and, if entered, is ignored.

.E is a single-character filename "extension" or file type. This may be any alphanumeric character but certain characters are reserved to indicate particular file types, for example:

.S for an assembly language source code file
.O for an Intel format object code file
.L for a listing file (intended to be printed)
.C for a binary file (8-bit data)

If the extension is omitted, a "default" extension will generally be assumed. For example, the assembler will assume an extension of .S for the source input file if no other is specified. A list of commonly-used extensions is given in Appendix 7.

In general, if an error is made when entering a device/filename the BACKSPACE key can be used to correct it. Pressing ESCape returns control to the monitor.

4. The System Monitor

The System Monitor provides the user with a set of commands which allow him to monitor and control various features of the development system. For example it provides facilities for examining and modifying memory and ports, executing programs etc. It is entered automatically after a system reset, following successful completion of the self-test sequence. The monitor prompt is a full-stop (.).

4.1 RAM usage

256 bytes of RAM (FF00-FFFF) are used for temporary storage and push-down stack (for return addresses etc.). This RAM also holds an image (or map) of all the user's internal CPU registers, a user's push-down stack (separate from the monitor's stack) and space for user-defined mnemonics. The area between the user's stack and the user's mnemonics has no defined boundary; the user must define this boundary with a tradeoff between stack size and the number of new mnemonics defined (see Appendix 4).

4.2 CPU register map

To preserve the state of the CPU in a user's program while debugging, the monitor keeps an image or map of all the user's CPU registers. This is referred to as the User Register Map. The monitor sets the CPU registers equal to the register map when control is transferred to a user program (E or S commands), and saves the registers in the map when control is returned to the monitor at a breakpoint (B command) or on completion of a program step (S command). The monitor allows display and/or modification of the register map by means of the M command. The user register map resides in the top 26 bytes of the scratchpad RAM.

4.3 Channelled input/output

The concept of channelled input/output gives the user the ability to select software device drivers to perform I/O to or from a peripheral through a "channel". A specific channel defines a fixed memory location where the address of the selected I/O device driver is stored. When doing I/O, the monitor fetches the device driver address from this location and transfers control to the driver for completion of the input or output function. The user selects the I/O driver for a peripheral by changing the contents of the fixed memory location for the channel through which the I/O is to be done.

The monitor has six I/O channels:

0	"Console Input"	:CI
1	"Console Output"	:CO
2	"Object Input"	:OI
3	"Object Output"	:OO
4	"Source Input"	:SI
5	"Source Output"	:SO

Channels 0 and 1 are dedicated to console input/output and should not normally be altered by the user. They are initialised to special drivers which, as well as accessing the keyboard and VDU, perform certain housekeeping tasks such as sampling the "ready" signal from the disk drives. It is important that these functions are not inhibited. The other four channels are available for use by system programs (assembler etc.), utility programs and user programs. The names (and associated mnemonics) of these channels have no particular significance; channels 2 and 4 can be used for input and channels 3 and 5 for output.

4.4 Command formats

Monitor commands consist of three parts:

4.4.1 A single letter IDENTIFIER.
4.4.2 An OPERAND or operands separated by commas or blanks.
4.4.3 A TERMINATOR either to abort the command or cause it to be executed.

On entering a command letter, the monitor echoes the letter, prints a space and then waits for the user to key-in the appropriate operand(s) in the format described later. A command is not executed until terminated by a carriage return; if terminated by a full-point the command is aborted. You cannot correct a mis-typed monitor command by using the backspace key.

4.5 Command identifiers

The following summarises the 14 different commands designated by single letter command identifiers:

4.5.1 B - (Only if single-step card MN8/3 is fitted) Insert a breakpoint in the user's program (in RAM or ROM) which transfers control back to the monitor. This allows the user to intercept his program at a specific point and examine memory and CPU registers to determine if his program is working correctly.

4.5.2 C - Copy the contents of a block of memory to another location in memory.

4.5.3 D - Dump the contents of a memory block as Intel format object code.

4.5.4 E - Execute (i.e. transfer control to) a user's program or system program.

4.5.5 F - Fill an area of memory with a given byte value.

4.5.6 G - Get (load) into memory the data from a binary (C-format) file.

4.5.7 L - Load into memory the data from an Intel format object code input.

4.5.8 M - Display, modify or tabulate the contents of memory.

4.5.9 P - Display and/or update the contents of an I/O port.

4.5.10 R - Display the contents of the user's registers.

4.5.11 S - (Only if single-step card MN8/3 is fitted) Execute user's program in single-step mode; the user's registers are printed after each program step.

4.5.12 T - Enter typewriter (or transfer) mode. Anything typed is simply echoed to the display.

4.5.13 W - Write (dump) an area of memory as a binary (C-format) file.

4.5.14 X - Load and execute a binary (C-format) file.

4.6 Command operands

A command operand represents a sixteen-bit number. Arithmetic expressions using addition (+) and subtraction (-) operators are permitted, so the general form for an operand is aaaa-bbbb+...+zzzz. The elements aaaa, bbbb etc. may be in one of the following forms:

1. 0-9,A-F A hexadecimal constant (leading zeroes need not be entered). If more than four digits are entered, only the last four have significance; advantage can be taken of this feature to correct a mis-typed value.

2. :MN The mnemonic :MN represents a 16-bit number, its value being defined by an entry in either the resident or user mnemonic table (see later).

3. $ Represents the "current address" +1. This is used in the M command when calculating relative jump displacements.

An equals sign (=) may be entered within the string of elements in an operand to display the value of the operand so far. Examples of typical operands are:

1. 4F7F — The operand value is equal to 4F7FH.

2. :PC — The mnemonic :PC is equivalent to address FFFEH so the operand is equal to FFFEH.

3. 5038-5000 — The operand value will be 38H.

4. 5038-5000=0038 — The same as 3 except that "=" was entered to display the operand value.

5. 5038-$	If the "current address" is 5000H then $=5001H and the operand equals 37H.
6. 5038-$=0037	The same as 5 except that the equals sign was entered.
7. 305038	More than 4 digits entered. Operand value = 5038H.
8. 305038=5038	Same as 7 but with equals sign entered.

Mnemonics consist of one or two characters preceded by a colon (note that this is different from the form of a device/filename specification where the colon follows the device mnemonic). The resident mnemonics recognised by the monitor are as follows:

:PC *	User's program counter.
:A	User's A register.
:F	User's F register (condition flags).
:I	User's I register.
:IF	User's IFF register (interrupt enable).
:B	User's B register.
:C	User's C register.
:D	User's D register.
:E	User's E register.
:H	User's H register.
:L	User's L register.
:IX *	User's IX register.
:IY *	User's IY register.
:SP *	User's stack pointer.
:CI *	Address of console input I/O driver.
:CO *	Address of console output I/O driver.
:OI *	Address of object input I/O driver.
:OO *	Address of object output I/O driver.
:SI *	Address of source input I/O driver.
:SO *	Address of source output I/O driver.
:TK	Keyboard driver routine.
:TT	Visual Display Unit driver routine.
:VP	Video Pager VDU driver routine.
:DK	Disk driver routine.
:PT	Printer driver routine.
:BB	"Byte bucket" driver routine.
:ED	Text editor execute address.
:ER	Text editor re-entry point.
:AS	Assembler execute address.
:PI	PIP execute address.
:TB	Symbol Table output routine execute address.

Those mnemonics marked with an asterisk (*) in the table above represent a memory address at which a two-byte value is stored. The remainder represent an address which is either the storage location for a single-byte value or is simply the start address of a software routine.

If a command requires more than one operand, the operands must be separated by single spaces or commas.

4.7 Detailed command descriptions

This section describes each monitor command in detail. The command format is shown, followed by a description and examples. For the purposes of this section, the conventions used are:

1. aaaa,...,zzzz — Denote 16-bit operand values as described in the previous section.
2. t — Denotes the command terminator; carriage return (cr), carat (^) or full-point (.).
3. ___ — Underlined items are typed in by the user. Non-underlined items are generated by the system. The underlines themselves are NOT entered.

4.7.1 B COMMAND - SET A BREAKPOINT.

This command causes the setting of a "trap" or breakpoint at a specified memory address. Upon encountering the breakpoint, the user's program will transfer control back to the monitor where registers, ports and memory contents may be inspected. A breakpoint is usually set within a section of executable code (a program) in which case the instruction at the breakpoint address is executed before control is returned to the monitor. Alternatively the breakpoint address may correspond to a data storage location in which case control is returned to the monitor when this location is accessed. Care should be taken if this second form of breakpoint is used as, should the memory location contain an instruction prefix code (CB, DD, ED or FD), then the breakpoint address will be automatically incremented. Only one breakpoint at a time is allowed.

Formats: .B aaaa,bt Set breakpoint at aaaa.
.B t Clear breakpoint.

If b=0 then only PC and AF are printed when the breakpoint is encountered. If b<>0 then all the user's registers are printed (see R command for the format of the register print-out). The breakpoint is automatically cleared when encountered, or it may be cleared by setting another breakpoint or by typing B return.

If the MN8/3 unit is not fitted this command has no effect.

4.7.2 C COMMAND - COPY MEMORY BLOCKS.

The copy command permits any block of memory data to be moved to any area of RAM. The move may be forward or backward and the new block may or may not overlap with the original block.

Format: .C aaaa,bbbb,cccct

The memory block aaaa to bbbb inclusive is copied to the memory block starting at address cccc.

Example:

.C 100,200,1200cr

Copy locations 100H to 200H inclusive to locations 1200H to 1300H.

4.7.3 D COMMAND - DUMP MEMORY.

The dump command writes a specified block of memory to the specified device/file in Intel format. This format is compatible with the Load command.

Format: .D aaaa,bbbbt Dump aaaa to bbbb inclusive.
DUMP TO? dev:filnam.e cr

The object code output includes 80 NULs of leader and 80 NULs of trailer, intended for when paper tape is used for storage. The default extension is .O and the default device is DK0:. If nothing is entered after the "DUMP TO?" prompt (carriage return alone typed) then the destination for the data is determined by the previous contents of the object-output channel. If this was undefined, results are unpredictable.

4.7.4 E COMMAND - EXECUTE PROGRAM.

The E command is used to execute all programs, including user's programs and Assembler, Text Editor etc.

Formats: .E aaaat Transfer control to address aaaa.
.E t Transfer control to address (:PC).

On executing this command the monitor loads all the CPU registers from the user register map and then transfers control to the specified address. If no address is specified the value stored at PC in the register map is used. This allows for program execution to be continued at the correct address after a breakpoint or single-step, and simplifies repeated execution at the same address.

4.7.5 F COMMAND - FILL MEMORY WITH SPECIFIED VALUE.

This command allows a specified range of memory to be filled with a given 8-bit value.

Format: .F aaaa,bbbb,cct Fill memory from aaaa to bbbb inclusive with byte value cc.

The value cc is written to every memory location from address aaaa to address bbbb inclusive. If cc is omitted the memory is filled with zeroes. Note that this function is performed by copying data from one location to the next so if there is a RAM fault, or if part of the specified range is not RAM, whatever value is read from the memory will be propagated to the end of

the range regardless of the stipulated value of cc.

4.7.6 G COMMAND - LOAD A BINARY FILE INTO MEMORY.

This command allows a C-format binary file to be loaded into RAM. The file format is compatible with the Write command.

```
Format: .G aaaat     Load a C-format file
                     to address aaaa.
        GET BINARY FILE: dev:filnam.e cr
```

If the operand aaaa is omitted or is zero, then the file is loaded to the address originally specified in the W command. If aaaa is non-zero then the file is loaded to address aaaa instead. The default extension is .C and the default device DK0:. Devices other than disk are not allowed.

The file format is as follows:

```
Header: Length         2-byte block length (LS byte 1st)
        Load address   2-byte load address (LS byte 1st)
        Execute addr.  2-byte execute address
        Spare          Three unused bytes
```
The header is therefore 9 bytes in all.

Data block: Data Up to 65535 bytes of binary data.

This may be followed by another header and data block. End-of-file is indicated by the physical end of data or by a block length of zero.

4.7.7 L COMMAND - LOAD MEMORY.

The L command provides the capability to load a program and/or data into memory from the specified device/file. The data must be in Intel format.

```
Format:    .L t     Load into memory.
           LOAD FROM? dev:filnam.e cr
```

This command loads data into memory until the end-of-data record is encountered (having a record length of 0 and a record type of 1). If a checksum error is detected the current address is printed, therefore the error is in the previous n bytes where n is the number of data bytes in the previous line of object code. The default extension is .O. If no device or filename is entered, the source of data is determined by the previous contents of the object-input channel. If a filename alone is entered, the device defaults to DK0:.
See also Section 8, the Relocating Linking Loader, and Appendix 3.

4.7.8 M COMMAND - DISPLAY AND MODIFY MEMORY.

Format: .M aaaat

The monitor prints the memory address aaaa followed by the contents of that address. The values are printed in hexadecimal unless a mnemonic has been associated with the value, in which case the mnemonic is printed. If the contents are to be changed, the new value is entered by the user. The new value is in the form of an operand as described in section 7, except that the value is truncated to 8 bits (unless the address aaaa is the storage location for a 2-byte value). For example, if location 5001H was to be changed to FFH:

```
.M 5001cr
5001 A3 FFcr
5002 A4 .
.
```

If the PC register is to be changed to 7F50H:

```
.M :PCcr
:PC  433F 7F50cr
0000 20 .
.
```

When the user is examining or modifying a memory location, the accompanying terminator signals the action the monitor is to take:

1. cr — No operand entered, display next address and data.
2. ^ — No operand entered, display previous address and data.
3. aa. — Operand aa entered but "." aborts command with no change to memory contents.
4. aacr — Operand aa entered, change value to aa and step to next address.
5. aa^ — Operand aa entered, change value to aa and display same address with the new data value displayed. If this is not aa, the address was not in RAM.

Example:

```
.M 16Acr
016A 3F cr
016B 92 ^
016A 3F 34FF^
016A FF cr
016B 92 .
.
```

Examining channel assignments:

```
.M :CIcr
:CI   :TK cr
:CO   :TT cr
:OI   6363 :PR^
:OI   :PR cr
:SI   6363 cr
:SO   6300 cr
FF33 80 .
.
```

Adding a mnemonic ":MN" to point to address FFE6H (see also Appendix 4):

```
.M FF53cr
FF53 80 4Dcr        'M'
FF54 FF 4Ecr        'N'
FF55 FF E6cr        Low order 8-bits
FF56 FF FFcr        High-order 8-bits
FF57 FF 80cr        Table terminator
FF58 FF .
.
```

User-defined mnemonics have precedence over system mnemonics. Therefore in the example above, :MN has precedence over :SP defining the same address.

4.7.9 M COMMAND - TABULATE MEMORY.

This command allows the user to display, but not change, a block of memory.

Format: .M aaaa,bbbbt Tabulate locations aaaa to bbbb inclusive.

The user enters the starting and ending addresses of the memory block separated by a comma or blank. Upon terminating with carriage return, the monitor prints the contents of aaaa to bbbb inclusive with up to 16 values per line. On the right-hand side of the screen the ASCII characters corresponding to the memory contents are printed. If the value does not represent a printing character (i.e. <20H or >7FH) then a dot (.) is printed. The tabulation may be stopped at any time by typing CTRL/C (hold down CTRL and press C). A "paged" display can be obtained by pressing key F4 (followed by a full stop) before the M-command is issued; paging is turned off with key F3. Similarly, the PRINT key can be used to cause the memory tabulation to be sent to the printer (if fitted); pressing F1 turns the printer off.

Example:

```
.M 4100,4127cr
4100 2B 90 12 20 00 B7 A5 21 10 94 04 20 CA B7 44 18  +.. ...!... ..D.
4110 81 11 34 21 07 94 17 45 12 55 A5 18 21 80 C5 55  ..4!...E.U..!..U
4120 54 45 58 54 09 21 40 22                          TEXT.!@"
.
```

4.7.10 P COMMAND - DISPLAY AND/OR MODIFY PORTS.

This command allows the user to display and/or modify any of the 256 I/O ports. Note that some ports are output only and cannot be read.

Format: .P aat Display port aa.

The conventions are the same as the M command (display and modify memory).

Example:

```
.P D1cr
D1 FF CF^
D1 FF 0^
D1 FF ^
D0 00 AA^
D0 AA .
.
```

N.B. port D1 was write-only.

4.7.11 R COMMAND - DISPLAY USER'S CPU REGISTERS.

Formats:

.R t	Print the registers.
.R 1t	Print a heading then the registers.

Examples:

```
.R cr
A000 0100 0104 CFB3 C09A FFEE EDF6 9C3E C3DC FE9B D6EC F1BE FFB4
.

.R 1cr
 PC   AF   IIF  BC   DE   HL  A'F' B'C' D'E' H'L'  IX   IY   SP
A000 0181 0104 CFB3 C09A FFEE EDF6 9C3E C3DC FE9B D6EC F1BE FFB4
.
```

Register IF represents the interrupt enable flip-flop. If IF=0 then interrupts were disabled when the monitor received control. If IF=4 then interrupts were enabled. Also, the appropriate state will be restored on an Execute or Single Step command.

4.7.12 S COMMAND - SINGLE STEP.

Formats: .S aaaa,bb,ct
.S t

Initial use of the S command is by the first format shown. aaaa is the address in the user's program at which single-step execution is to begin; bb is the number of steps to be executed before pausing and c specifies the long or short register print-out. If c=0 or is omitted, then only PC and AF are printed after each step. If c=1 then all the registers are printed (in R command format). If bb is omitted, one step is executed.

After the specified number of program steps (instructions) have been executed, the monitor waits for either "." or a carriage return to be typed. "." terminates the single-step command and carriage return causes one more step to be executed. After the command has been terminated, execution of the user's program can be continued from where it was terminated by typing E followed by carriage return. Alternatively, single-stepping can be resumed at the same point by typing S return.

If the single-step card (MN8/3) is not fitted the S command acts exactly like the E command.

Examples:

```
.S 4100,3cr
4101 0181
4102 00C1
4103 00C0 .
.

.S 4106,1,1cr
4107 00C0 0104 CFB3 C09A FFEE EDF6 9C3E C3DC FE9B D6EC F1BE FFB4 cr
410A 00C0 0104 CFB3 C09A 9410 EDF6 9C3E C3DC FE9B D6EC F1BE FFB4 cr
410B 00C0 0104 D0B3 C09A 9410 EDF6 9C3E C3DC FE9B D6EC F1BE FFB4 .
.S cr
410D 00C0 0104 D0B3 C09A 9410 EDF6 9C3E C3DC FE9B D6EC F1BE FFB4 .
.
```

4.7.13 T COMMAND - TYPEWRITER (or TRANSFER) MODE.

Format: .T t (typewriter mode)
.T at (general transfer mode)

If T is entered with no operand the monitor enters a mode in which characters read from the console input channel (keyboard) are simply echoed to the console output channel (screen). Command characters are not recognised. This mode is terminated when CTRL/C is read. For details of control characters which are recognised see the description of CONOUT in Appendix 5.

In the generalised version of this command the operand a determines which pair of channels is used, and also whether the

driver assigned to the input channel is initialised when the first character is read. The values which a may take are:

a	Input channel	Output channel	Input initialised?
0	CI	CO	No
8	CI	CO	Yes
2	OI	OO	No
A	OI	OO	Yes
4	SI	SO	No
C	SI	SO	Yes

The drivers assigned to the appropriate input and output channels are called alternately until ETX (CTRL/C) is read on the input channel, whereupon the system returns to command mode.

4.7.14 W COMMAND - WRITE BINARY FILE.

This command allows RAM contents to be stored in a binary format, thereby occupying less space and speeding the write and read operations.

Format: .W aaaa,bbbb,ccct
WRITE BINARY FILE: dev:filnam.e cr

The first operand aaaa is the address of the first byte to be written. The second operand bbbb is the address of the last byte to be written. The third operand cccc is the execute address used by the X command; if omitted the execute address defaults to aaaa. The default extension is .C.

After the specified address range has been written the prompt "NEXT BLOCK" is displayed. Typing just carriage return terminates the command; alternatively a second block of contiguous addresses can be stored by entering aaaa,bbbb,cccc as before. Any number of blocks of contiguous RAM can be stored on the same file in this way. Only on the last block is the execute address relevant.

For the file format, see under G command.

4.7.15 X COMMAND - EXECUTE BINARY FILE.

This is identical to the G command except that, after loading, the file is automatically executed at the address specified in the W command.

Format: .X t
EXECUTE BINARY FILE: dev:filnam.e cr

The default extension is .C.

5. PIP - Peripheral Interchange Program

5.1 Description

The Peripheral Interchange Program (PIP for short) is an integral part of the Disk Operating System. Its main functions are to allow manipulation of disk files, e.g. deleting and renaming files, and to provide a general means of transferring data from any device (peripheral) or disk file to any other device or file. PIP also provides the facility of listing an index of disk contents.

5.2 Using PIP

PIP is executed by means of a mnemonic address. The mnemonic for PIP is ":PI" and therefore the appropriate monitor command is:

```
.E :PIcr
*
```

PIP responds with an asterisk as a prompt.
An alternative method of entering PIP from the monitor is to press ENTER twice.

The general operating procedure is to type a command, immediately following the asterisk prompt, and cause it to be executed by pressing carriage return. The backspace key may be used to correct a mis-typed entry. If CTRL/U is typed instead of carriage return then the entire line is ignored. Pressing ESCape exits to the monitor.

PIP commands consist of a command word, which may be abbreviated to the initial letter, followed by the appropriate qualifying information. Only one command per line is permitted. PIP outputs a question mark (?) if it does not understand the command.

5.3 Device Mnemonics

For PIP to read data from, or write data to, a peripheral it must have a mnemonic; an absolute memory address is not acceptable. However this is not a real limitation as PIP searches both the resident and user (RAM) mnemonic tables so if a special device driver is given a mnemonic this automatically becomes accessible in PIP.

The main resident mnemonics are as follows:

```
TK:  Keyboard (console)
TT:  VDU display (console)
DK:  Floppy disk
VP:  "Video Pager"
PT:  Parallel printer
BB:  Byte Bucket
```

5.4 The "?" convention

Certain PIP commands allow a character or characters in a filename or extension to be replaced by a question mark (?). This character is a "wild card" and has the meaning "any character" in this position or positions. For example FILNAM.? means a filename FILNAM with any extension and A????? means any filename beginning with the letter A.

5.5 PIP commands

All PIP commands may be abbreviated to just the initial letter. F1, F2 etc. refer to device/filename specifications as described in section 3. Items in square brackets are optional.

5.5.1 COPY F1=F2[,F3...]

This is the generalised copy command which gives PIP its name. ASCII data from the device or file specified to the right of the equals sign is copied to the device or file to the left of the equals sign. If more than one file is specified to the right of the = sign then F1 will consist of these files chained together (concatenated) in the given order (only disk files may be concatenated in this way).
The default extension for F1 is .S and that for F2, F3 the extension specified for F1 (i.e. .S if none specified for F1). If a filename alone is specified then the device defaults to DK0:. If an extension alone is specified for F3 etc. then the filename is assumed to be the same as F2. Question marks cannot be used to cause the copying of multiple files by a single command. Use the utility program MFT instead.
If a device (e.g. DK1:) is specified in the file list to the right of the equals sign then this device becomes the default for the rest of the line until another device is specified. E.g. COPY F1=DK1:F2,F3 means both F2 and F3 are to be read from disk 1.
If F2 has the extension .C (binary file) then the copy command performs a binary copy - concatenation is not possible in this mode. Otherwise ASCII data is assumed; NULs are removed and ETX is recognised as a terminator.

Examples:

COPY VP:=PROGRM.L
Copy the assembly listing DK0:PROGRM.L to the console display using the video pager driver.

C NEWFIL=TK:
Read from the console keyboard (until CTRL/C or a time-out) and store the data as a file DK0:NEWFIL.S.

C DK1:FILE2.O=FILE1.O,.P
Write to disk 1 a file called FILE2.O consisting of DK0:FILE1.O and DK0:FILE1.P concatenated.

5.5.2 RENAME F1=F2

Rename file F2 as F1. Any device specified for F1 is ignored, even if different from F2. If a filename alone is specified for F2 then DK0: is assumed. The default extension for F1 is .S and that for F2 is F1's extension.
If an extension alone is specified for F2, then the filename is assumed to be the same as F1.
Question marks are allowed as long as they are in equivalent positions in both F1 and F2, in which case all files fitting the description are renamed.

Examples:

RENAME NEWFIL.X=OLDFIL.X
The file DK0:OLDFIL.X is renamed as DK0:NEWFIL.X.

R FILE2.?=FILE1.?
All files FILE1.? are renamed as FILE2.?, i.e. FILE1.S becomes FILE2.S, FILE1.O becomes FILE2.O etc.

5.5.3 DELETE F1[,F2...]

Delete the specified file(s). The default extension is .L. If an extension alone is specified for F2 etc. then the filename is assumed to be the same as F1. If an extension alone is specified for F1 the the filename is effectively ??????.
Question marks are allowed; all files fitting the description are deleted.

Examples:

DELETE FILE1,NEWFIL.X
Delete DK0:FILE1.L and DK0:NEWFIL.X

DEL DK1:.L
Delete all .L (listing) files on disk 1.

D PROGRM.S,.O,.L
Delete DK0:PROGRM.S, DK0:PROGRM.O and DK0:PROGRM.L

Note: Different disks must not be mixed in one delete command.
A delete command results in the message CONFIRM?; type Y to delete the files, any other character to abort the deletion.
The command D ??????.? will NOT delete all files; control is lost and a system reset must be performed. To delete all files use the utility program INIT.

5.5.4 LIST [F1,F2...]

List an index of disk files, with their sizes in sectors and how much of the disk is unused. LIST alone gives all the files on disk 0, if followed by a qualifier then only those files fitting the description are included in the index. Question marks are allowed.

Examples:

LIST DK1:
Lists the contents of disk 1.

LIST .S,.O
Lists the source and object files on disk 0.

L FILNAM
Lists the files FILNAM.?, i.e. FILNAM with any extension.

L A?????
Lists the files starting with the letter A.

A sector holds 126 bytes of data so if the file size is indicated as, for example, 10 sectors then the length of the file is between 1135 and 1260 bytes.

The LIST command automatically invokes the video-pager VDU driver (VP:) so that output is paused after 20 lines; press <RETURN> for the next page or SPACE for one more line.

6. Text Editor

6.1 Introduction

The text editor is intended for the origination and modification of assembly language source programs. It is line-based and allows random-access editing of ASCII character strings read into memory from disk or other storage device. The following commands are provided:

6.1.1	An	Advance record pointer n records.
6.1.2	Bn	Backup record pointer n records.
6.1.3	Cn dS1dS2d	Change string S1 to string S2 for n occurrences.
6.1.4	Dn	Delete next n records.
6.1.5	En	Exchange n records with new records to be inserted.
6.1.6	I	Insert records after current record.
6.1.7	Ln	Search forward for line number n.
6.1.8	Mn	Enter commands into one of two alternative command buffers (pseudo-macro).
6.1.9	N	Print first, last and current line numbers.
6.1.10	Pn	Output ("punch") n records to object output channel, without line numbers.
6.1.11	Qn	Same as Pn but NULs not required from keyboard.
6.1.12	R	Read records into buffer from source input channel until buffer full or source exhausted.
6.1.13	Sn dS1d	Search forward for nth occurrence of string S1.
6.1.14	T	Insert records at the top of the buffer.
6.1.15	Vn	Output ("view") n records to console output channel, with line numbers.
6.1.16	Wn	Output ("write") n records to source output channel, with line numbers.
6.1.17	Xn	Execute command buffer n (pseudo-macro).
6.1.18	Y	Delete entire buffer contents.
6.1.19	Z	Composite command equivalent to B Q Y R N except that the buffer is not completely filled.

6.2 Terminology

The following terms are used in the descriptions of editor commands:

6.2.1	Source	ASCII characters (file) comprising the assembly language program etc.
6.2.2	Record	A single source statement, delimited by carriage return.
6.2.3	Buffer	Computer memory area used to store a portion of the source.
6.2.4	Pointer	The position in the buffer (always the beginning of a record) where the next action of the editor will be initiated.
6.2.5	Current	The record in the buffer pointed to by the pointer.

6.2.6 Insert — Insertion of a new record into the buffer immediately following the record pointed to by the pointer.

6.2.7 Delete — Removal from the buffer of the record pointed to by the pointer.

6.3 Text Editor Commands

In the following command descriptions, the first alphabetic character designates the command and n represents a decimal value from 0 to 9999. If n is omitted it is assumed to be zero. The editor prints a greater-than sign (>) when ready to accept a command string. When entering data from the console channel (usually keyboard) the user may delete the previous character by typing backspace or DEL. When using DEL a backslash (\) will be printed for each character deleted; this facility is intended for use with a printer terminal rather than a VDU. The user may delete the entire line by typing CTRL/U. Pressing ESC returns control to the monitor - take care not to do this accidentally. TAB characters entered from the keyboard or contained in the source file are expanded on the screen, but not in the buffer memory.

6.3.1 R - READ source statements from the source input channel. The read continues until the buffer is full, the end of data (ETX) is read, 80 consecutive NUL or line-feed characters are read or a time-out occurs. Each use of the R command stores new source records after the last record in the buffer. The R command leaves 83 or less bytes for editing if the buffer is filled (BUFFER FULL message).

6.3.2 Bn - BACKUP the record pointer n records from the current record. If n is zero, the pointer is positioned at the first record in the buffer.

6.3.3 An - ADVANCE the record pointer n records from the current record. If n is zero, the pointer is positioned at the beginning of the last record in the buffer.

6.3.4 Pn - Output n records starting with the current record. If n is zero the editor will output all records from the current record to the end of the buffer. The output is initiated by typing NUL (CTRL/@) on the keyboard, and is terminated by typing another NUL. The editor will not print another prompt until this is done. The records are output to the Object Output channel.

6.3.5 Qn - Output n records starting with the current record. This command is identical to the P command except that it is not necessary to type NULs.

6.3.6 Sn ´string´ - SEARCH the buffer memory, starting at the current record, until the nth occurrence of the string supplied between the delimiters is found. Then position the pointer at the beginning of the record containing the string. Any character which is not in the string

may be used as the delimiter. If n is zero the first occurrence of the string is sought. If the nth occurrence of the string is not found, "EOF" is printed and the pointer is positioned at the last record in the buffer.

6.3.7 Cn ´string1´string2´ - CHANGE the next n occurrences of string1 to string2 starting with the current record. The pointer will be positioned at the record where string1 occurred for the nth time. The delimiter may be any character not in either string1 or string2. If n is zero, then the first occurrence of string1 is changed. If the nth occurrence of string1 is not found, "EOF" is printed and the pointer is positioned at the last record in the buffer. After each string substitution the pointer is repositioned at the beginning of the record so if string1 is a sub-string of string2, and n is greater than 1, the results may not be as expected.

6.3.8 I - INSERT records after the current record. The prompt character ´<´ indicates that the editor is ready to accept a line for insertion. A carriage return must terminate each record inserted (line feeds are supplied by the editor). A line containing no characters terminates the insert mode, the pointer is positioned at the record immediately following the inserted records. Care should be taken when the BUFFER FULL message is printed because a further insertion could corrupt the editor´s work area. Space for more insertion can be created by issuing the commands B Qn B Dn where the number of lines output is the same as the number deleted.

6.3.9 T - Insert records at the top of the buffer. This is the same as the I command except that the new records are inserted before the first record in the buffer.

6.3.10 Dn - DELETE n records beginning with the current record. If n is zero, one record is deleted.

6.3.11 En - Delete n records and insert new records in their place. Once in the insert mode, operation is identical to the I command. This command should not be used to insert records at the end of the buffer as they will be inserted before, rather than after, the last record in the buffer.

6.3.12 Wn and Vn - Print n records (with line numbers) starting with the current record. The pointer is unaffected. If n is zero, one record is printed. The W command directs the output to the source-output channel; the V command directs output to the console output channel. Note that the source output channel is not initialised on entry to the editor.

6.3.13 Ln - Search for line number n, starting with the current record. If line n is not found, "EOF" is printed and the pointer is set at the last record in the buffer.

6.3.14 N - Output to the console the line numbers of the first record in the buffer, the last record read by the R or Z command, and the current record. If the buffer is completely empty these numbers are not meaningful.

6.3.15 Mn - Enter a command string into one of two command buffers depending on the value of n (1 or 2). The command buffers will accept character strings of up to 40 characters. No error is issued if more than 40 characters are entered and editor variables could be corrupted if overrun occurs. If n is zero, command buffer 1 is selected.

6.3.16 Xn - Execute one of two command buffers depending on the value of n (1 or 2). The command buffers are executed exactly as if the command string was entered after the normal prompt (>). Once a command buffer has been executed, control is transferred back to the user. If n is zero, command buffer 1 is selected.

6.3.17 Y - Delete the entire buffer's contents. This is much faster than the equivalent command B D9999.

6.3.18 Z - Get next buffer's worth of file for editing. This composite command is equivalent to B Q Y R N except that the "read" will leave at least 512 bytes of the buffer unfilled, even if BUFFER FULL is printed. The command is intended to simplify the editing of large files which will not fit into the buffer in their entirety. The first buffer's worth of the file is read in using Z. After editing, the next buffer's worth is fetched using Z again. After the entire file has been edited, the remaining contents of the buffer are written to the output file, once again by issuing the Z command (see Section 9).

6.4 Using the Editor

If it is intended to use the W command, the source-output channel must be set up manually (although note that it is initialised to PT: on a system reset):

```
.M :SOcr
:SO  xxxx :PT^
:SO  :PT  .
.
```

The text editor is entered using the following monitor command:

```
.E :EDcr
FROM? dev:file1.e cr
TO? dev:file2.e cr
>
```

After FROM? is entered the device or filename from which the file to be edited is to be read. The default extension is .S. If the editor is to be used to create a new source file then this may be left blank, but take care not to issue an R or Z

command (the new file should be written to disk using B Q).

More than one filename, separated by commas or blanks, may be entered after FROM?, in which case the files will be concatenated on being read into the editor buffer (although 80 consecutive NULs or line-feeds will still terminate reading). If carriage return alone is typed after FROM?, the source input channel is left unaltered and this allows a device which has no mnemonic to be used by entering its address into the source-input channel before executing the editor.

After TO? is entered the device or filename to which the edited program is to be written. The default extension is .S. If the editor is to be used to create a new file, its name should be entered here.

If carriage return alone is entered in response to TO?, then the filename defaults to that entered after FROM?, the extension defaults to .N ("new"), and the device defaults to DK0:. If nothing was entered after FROM? either, the object-output channel is left unaltered.

Once editing is complete, the buffer's contents must be written to the output file using the P, Q or Z command. ESC may then be pressed to exit to the monitor, any disk files being written will be closed and the disk directory updated.

The editor uses the following input/output channels:
Source Input channel: Data input using R or Z commands.
Object Output channel: Data output using P, Q or Z commands.
Source Output channel: Data output using W command.

6.5 Notes

6.5.1 Records (separated by carriage returns) are counted as they are read in. Line numbers reside in memory with their lines and correspond to line numbers assigned by the assembler. These numbers cannot be altered; when an insertion is done each line inserted receives the same line number (0000). When a line is deleted its corresponding line number is also deleted.

6.5.2 A disk time-out is implemented whereby if no key is pressed for ten minutes the disk drive motors are switched off. If this occurs whilst editing a file, and if some of the file has already been written to the output channel, then when the rest of the file is written to disk a DIRECTORY ERROR will result and the edit will be lost. This is because once the disk stops spinning the system has no way of telling if it has been changed and it cannot allow the directory of another disk to be corrupted.

6.5.3 In some instances the user may want to re-enter the editor without altering the buffer contents. This can be achieved by issuing the monitor command:

```
.E :ERcr
>
```

This can be useful if ESC was pressed by mistake. However, some flags are reset and any file which was being written will have been closed. The result is that if an R command is issued, records are read from the beginning of the input file even if they had been read already. Also, if a Q command is issued the likelihood is that a FILE ALREADY EXISTS error will be produced. The best method of recovering from having pressed ESC by mistake is to CHANGE THE DISK, re-enter the editor and dump the buffer contents by typing B Q <cr> ESC. The edited file can be recovered by concatenating the two output files produced, and splicing the resulting file into the original program.

6.5.4 On output from the editor, records are separated by CR LF (carriage return, line feed) whether or not line feeds were present in the input file.

7. Z80 Mnemonic Assembler

7.1 Introduction

The assembler reads a source program written in Z80 mnemonics, containing assembler directives and pseudo-ops, and outputs an assembly listing and object code. The object code is in industry standard "Intel" format modified to include relocating and linking information (see Appendix 3).

7.2 Capabilities

The assembler recognises all standard Z80 mnemonics. It supports conditional assemblies, global symbols, relocatable programs and a printed symbol table. The assembler can assemble any length program, limited only by symbol table size which is user selectable. An option in the assembler activates linkages to a RAM-loadable program which can provide macro facilities or permit the assembly of programs written for processors other than the Z80.

7.3 Definitions

7.3.1 SOURCE MODULE - the user's source program. Each source module is assembled into one object module by the assembler. The end of a source module is defined by an 'END' pseudo-op or an EOT character (CTRL/D). The source module is read via the source-input channel.

7.3.2 OBJECT MODULE - the object output of the assembler for one source module. The object module contains linking information, address and relocating information, machine code and checksum information for use by the relocating linking loader. The object module is in ASCII. A complete definition of the object format is given in Appendix 3. The object output is written to the object-output channel.

7.3.3 LOCAL SYMBOL - a symbol in a source module which appears in the label field of a source statement.

7.3.4 INTERNAL SYMBOL - a symbol in a source (and object) module which is to be made known to all other modules which are loaded with it by the linking loader. Internal symbols are defined by the GLOBAL pseudo-op. An internal symbol must appear in the label field of the same source module. Internal symbols are assumed to be addresses, not constants, and will be relocated by the loader.

7.3.5 EXTERNAL SYMBOL - a symbol which is used in a source (and object) module but which is not a local symbol (does not appear in the label field of a statement). External symbols are defined by the GLOBAL pseudo-op. External symbols may not appear in an expression which uses operators. An external symbol is a reference to a

symbol which exists, and is defined as internal, in another program module.

7.3.6 GLOBAL DEFINITION - both internal and external symbols are defined as "GLOBAL" in a source module. The assembler determines which are internal and which are external.

7.3.7 POSITION INDEPENDENT - a program which can be placed anywhere in memory. It does not require relocating information in the object module.

7.3.8 ABSOLUTE - a program which has no relocation information in the object module. An absolute program which is not position independent can be loaded only in one place in memory in order to work properly.

7.3.9 RELOCATABLE - a program which has extra information in the object module which allows a relocating loader to place the program anywhere in memory.

7.3.10 LINKABLE - a program which has extra information in the object module which defines internal and external symbols. A linking loader uses the information to resolve external references to internal symbols in another module.

7.4 Assembly language syntax

An assembly language program (source module) consists of labels, opcodes, pseudo-ops, operands and comments in a sequence which defines the user's program. The assembly language conventions are described below:

7.4.1 Delimiters

Labels, opcodes, operands and pseudo-ops must be separated from each other by one or more commas, spaces or tab characters (CTRL/I). The label may be separated from the opcode by a colon only, if desired.

7.4.2 Labels

A label is composed of between one and six characters. The characters in the label cannot include '()*+,-<>=./:; or space. In addition, the first character cannot be a number (0-9). A label can start in any column if immediately followed by a colon (:). It does not require a trailing colon if it starts in column one. For example:

```
Allowed:
   LAB
   L923:
   $25
     A25E:
Not allowed:
   9LAB     ;starts with a number
```

```
L)AB    ;illegal character
L:ABC   ;illegal character
 LAB    ;no colon and does not start in column 1
```

7.4.3 Opcodes

There are 74 generic opcodes (such as 'LD'), 25 operand keywords (such as 'A'), and 693 legitimate combinations of opcodes and operands in the Z80 instruction set. The full set of these opcodes is documented in the Mostek or Zilog "Z80 ASSEMBLY LANGUAGE PROGRAMMING MANUAL". The assembler allows one other opcode which is not explicitly shown in the manual:

```
IN   F,(C)   ;set the condition bits according
             ;to the contents of the port whose
             ;address is in the C register.
```

7.4.4 Pseudo-ops

The following pseudo-ops are recognised by the assembler:

ORG nn	Origin - sets the program counter to the value nn. If omitted, the program counter defaults to 0000.
label EQU nn	Equate - sets the value of a label to nn; can only occur once for each label.
label DEFL nn	Define label - sets the value of a label to nn; may be repeated with different values for the same label.
DEFM 'text'	Define message - generates a string of bytes in the object code having the ASCII equivalent of the characters within quotes. The character ' is represented by '' within the text string. Maximum length of the message is 63 characters; only the first 4 bytes of the object code are shown in the assembly listing.
DEFB n	Define byte - generates a single byte of object code with value n.
DEFW nn	Define word - generates a pair of bytes of object code having a 16-bit value nn. The least-significant byte comes first.
DEFS nn	Define storage - reserves nn bytes of memory starting at the current program counter. No object code is produced for these bytes. Equivalent to ORG $+nn.
END nn	End statement - defines the last line of the program. nn is optional and represents the execute address of the program. This can be used by a loader to start execution of a loaded program automatically. If omitted it defaults to the first address of the program.
GLOBAL symbol	Define global symbol - any symbol which

	is to be made known among several separately assembled modules must appear in this type of statement. The assembler determines if the symbol is internal (defined in the program) or external (used in the program but not defined as a label).
NAME symbol	Module name - This pseudo-op defines the name of a module. The name is placed in the heading of the assembly listing and is placed in the first record of the object module to identify it. If omitted the name defaults to the source module disk filename.
PSECT op	Program section - This pseudo-op may appear only once in a source module. It defines the program module attributes for the following operands: REL - relocatable (default). ABS - absolute. No relocating information is generated in the object module.
IF nn	Conditional assembly - If the expression nn is true (non-zero) the IF pseudo-op is ignored. If the expression is false (zero), the assembly of subsequent statements is disabled until an ENDIF (or END) pseudo-op. 'IF' pseudo-ops cannot be nested.
ENDIF	End of conditional assembly - re-enables assembly of subsequent statements.

7.4.5 Assembler Directives

Assembler directives are pseudo-ops which modify the assembly listing format. They are not listed in the assembly listing, but they are assigned statement numbers.

EJECT	Ejects a page of the listing.
TITLE s	Ejects a page and prints the string s at the top of each page. The title can be up to 32 characters long.
LIST	Turns the assembly listing on (default).
NLIST	Turns the assembly listing off.

7.4.6 Operands

There may be zero, one or two operands present in a statement depending upon the opcode used. An operand may take one of the following forms:

7.4.7 A generic operand such as 'A', which stands for the accumulator. Summary of generic operands:

A - A register (accumulator)
B - B register
C - C register
D - D register
E - E register
F - F register (flags)
H - H register
L - L register

AF - AF register pair
AF'- AF' register pair
BC - BC register pair
DE - DE register pair
HL - HL register pair
SP - Stack pointer register

I - I register
R - Refresh register

IX - IX index register
IY - IY index register

NZ - Not zero
Z - Zero
NC - Not carry
C - Carry
PO - Parity odd/not overflow
PE - Parity even/overflow
P - Sign positive (plus)
M - Sign negative (minus)

7.4.8 A Constant

A constant must be in the range 0-0FFFFH. It can be in the following forms:

Decimal	This is the default mode. Any number may be denoted as decimal by following it with the letter 'D'. E.g. 35, 249D.
Hexadecimal	Must begin with a number (0-9) and end with the letter 'H'. E.g. 0AF1H.
Octal	Must end with the letter 'Q' or 'O'. E.g. 377Q, 200O.
Binary	Must end with the letter 'B'. E.g. 0110111B.
ASCII	Letters enclosed in quote marks will be converted to their equivalent ASCII value. E.g. 'A' = 41H.

A constant may be preceded by a unary plus (+), minus (-) or logical not (.NOT.) operator.

7.4.9 A Label

As this is a two-pass assembler, forward references are permissible. I.e. a label used as the operand of an instruction need not have yet been defined by being in the label field of another instruction. However, a label itself cannot be defined in terms of another label which has not yet been defined. I.e.:

```
H EQU I
I EQU 7    is not allowed.
```

A label may be preceded by a unary plus, minus or logical not operator (except when the label is an external symbol).

7.4.10 The $ symbol

The symbol '$' is used to represent the value of the program counter of the current instruction. In a relative jump (JR or DJNZ) instruction $ is temporarily given the value 0 so that either JR LABEL or JR LABEL-$ is permitted. This has the effect that JR $+n, meaning jump forward n bytes, will not work.

7.4.11 An Expression

An expression consists of two or more quantities combined by means of arithmetic and logical operators, the result being evaluated as a 16-bit number. The allowed operators are as follows:

+	addition
-	subtraction, unary minus
*	multiplication
/	division
.NOT.	logical not (1's complement)
.AND.	logical AND
.OR.	logical OR
.XOR.	logical exclusive-OR
.SHR.	logical shift-right
.SHL.	logical shift-left

All operators treat their arguments as 16-bit unsigned quantities, and generate 16-bit quantities as their result. The division operator produces the arithmetic integer quotient of its operands, discarding any remainder. The SHR and SHL operators are linear shifts which shift their first operands right or left, respectively, by the number of bit positions specified by their second operands. Zeroes are shifted into the high-order or low-order bits, respectively, of their first operands.

Expressions are evaluated in the following order:

1. Parenthesised expressions.
2. * / .SHR. .SHL.
3. + -
4. .NOT. .AND.
5. .OR. .XOR.

Parentheses (brackets) may be used to modify the order of evaluation. The allowed range of an expression depends on the context of its use; an error message will be generated if this range is exceeded. In general, the limits on the range of an expression are 0 to 0FFFFH. The limits on the range of a relative jump are -126 to +129.

For relocatable programs, the assembler will output relocation information in the object module for those addresses which are to be relocated by the loader. Expressions are determined to be relocatable addresses or non-relocatable constants according to the following rules:

<constant><operator><constant>=<constant>
<constant><operator><relocatable>=<relocatable>
<relocatable><operator><constant>=<relocatable>
<relocatable><operator><relocatable>=<constant>

This is, of course, strictly accurate only for addition and subtraction operations so the user should be aware that the assembler may wrongly determine a value as relocatable or non-relocatable when other operators are used in expressions.

External symbols are not allowed in expressions. External symbols are always assumed to be relocatable values.

7.4.12 Comments

A comment is defined as any characters following a semicolon (;) in a source statement, except when the semicolon appears in quotes. Comments are ignored by the assembler but they are printed in the assembly listing. Comments can begin in any column. Note also that any program statement having an asterisk (*) in the first column is ignored by the assembler.

7.5 Assembly Listing

The assembly listing includes address, object code, statement number and the original source statement. Tab characters in the source statement are expanded in the assembly listing, tab positions being every eight columns. The value of each equated symbol will be printed with a pointer (>) next to it. Any address which is relocatable will be identified by an apostrophe (´). Macro expansions (macro option only) will be printed with a plus character (+) next to the statement number. The statement numbers and page numbers are printed in decimal.

Errors are indicated by a letter in the left margin; if the listing is inhibited (K option) lines in which an error is detected will be output to the console. The listing is output to the Source Output channel.

7.6 Absolute Modules

The pseudo-op PSECT ABS defines a module to be absolute. The program will be loaded at the exact address at which it is assembled. This can be useful to define a set of global constants which must not be relocated.

7.7 Relocatable Modules

Programs default to relocatable if the PSECT ABS pseudo-op is not specified. Only 16-bit address values can be relocated; any 8-bit quantity, whether derived from a 16-bit address value or not, will NOT be relocated.

A label equated to a relocatable value will be treated as relocatable. A label equated to a non-relocatable value will be treated as non-relocatable UNLESS it is an internal symbol (i.e. present in a GLOBAL statement) in which case it will always be treated as relocatable. External symbols will always be marked relocatable.

7.8 Global Symbol Rules

Both passes of the assembler must be done if global symbols are used. An external symbol may not appear in an expression, e.g.:

```
  GLOBAL SYM1
  CALL   SYM1           ;OK
  LD     HL,(SYM1)      ;OK
  LD     HL,SYM1+25H    ;not allowed
```

An external symbol is always considered to be a 16-bit address. Therefore an external symbol may not appear in an instruction requiring an 8-bit operand. E.g.:

```
  GLOBAL SYM1
  LD     A,SYM1         ;not allowed
  LD     (IX+SYM1),A    ;not allowed
  BIT    SYM1,A         ;not allowed
```

An external symbol cannot appear as the operand of an EQU or DEFL pseudo-op. E.g.:

```
        GLOBAL SYM1
  SYM2  EQU    SYM1     ;not allowed
```

An internal symbol is always marked relocatable, except for absolute assemblies. This is important, because an internal symbol will be relocated even though it looks like a constant. For example:

```
      PSECT  REL
      GLOBAL YY
```

```
YY  EQU    0AF3H      ;constant value
    LD     A,(YY)     ;YY will be relocated on loading.
```

This problem can be overcome by defining those symbols which correspond to constant values in a separate source module containing the PSECT ABS statement.

7.9 Using the Assembler

To execute the assembler, the user enters the following command:

.E :AScr

where E is the monitor command for execute, and :AS is the mnemonic which stands for the assembler starting address. The user must then reply to the assembler's prompts as follows:

SOURCE FROM? dev:filnam.e cr
OBJECT TO? dev:filnam.e cr
LISTING TO? dev:filnam.e cr
OPTIONS?list-of-options cr

In all three cases the device defaults to DK0:.
If no extension is given after SOURCE FROM? it defaults to .S.
If carriage return alone is entered after OBJECT TO? then the filename defaults to that given after SOURCE FROM? with an extension of .O.
If carriage return alone is entered after LISTING TO? then the filename defaults to that given after SOURCE FROM? with an extension of .L.
If carriage return alone is entered after SOURCE FROM?, OBJECT TO? and LISTING TO? then the source input, object output and source output channels are left unchanged. If they were previously undefined then results are unpredictable.

7.10 Options

The possible options are as follows:

K Kill listing. The assembly listing output is suppressed.

L Listing (default). The assembly listing is output to the source output channel.

M Macro option. This option is specified to activate a RAM-loaded macro handler or cross-assembler for another processor. The macro handler or cross assembler must have been loaded previously otherwise the assembler will crash.

N No object output. The object code output is suppressed.

O Object output (default). The object code is output to the object output channel.

P Pass 2 only. This selects and runs only pass 2 of the assembler. There must either be a valid symbol table in RAM

as a result of a previous run of pass 1 of the assembler, or option R must also be specified (see 7.12.4).

Q Quit. The operation is aborted and control returned to the monitor.

R Reset the symbol table. This option clears the symbol table of all previous symbols. It is automatically carried out before pass 1 of the assembler so the R option is used only with the P option (see 7.12.4).

S Symbol table. The symbol table is output to the source output channel following the assembly listing (if any). The symbol table is otherwise not output by the assembler (although it can be obtained by the E :TB command).

. Set the options to their default states and allow the user to try again.

Example:
OPTIONS?P N S cr

The user has selected pass 2 only, no object output and a symbol table.

7.11 Error messages

Any error detected by the assembler is indicated in the assembly listing by a single letter abbreviation in the left margin, next to the statement which is in error. If the listing is suppressed by means of the K option then the lines containing errors are output to the screen (console output channel) when they are detected during pass 2 of the assembly. When the assembly is complete the total number of errors is output at the end of the listing as well as on the console display.

D Invalid digit error. A constant contains a digit which is illegal for the specified radix.

E External symbol usage error. An external symbol is being used in an expression or as the operand of an EQU or DEFL pseudo-op.

F* Symbol table full. There are too many symbols for the size of symbol table specified (see 7.12.2). This is an abort error.

I Invalid operand error. There is an invalid operand or combination of operands for the given opcode.

L Label error. A label or symbol contains an invalid character. This error can result from a wrongly formed expression.

M Multiple definition error. The same label was defined more than once.

N No label error. An EQU or DEFL pseudo-op has no label.

O Opcode error. An invalid opcode exists in the opcode field of the source statement.

P PSECT error. The PSECT pseudo-op exists more than once in the same program.

Q Unmatched quotes error. An expression has an odd number of quotes in it.

R Range error. An operand exists which is out of the allowed range for the given opcode, for example a relative jump with a displacement of greater than 128.

S Syntax error. An expression is wrongly formed (e.g. has unmatched parentheses or includes a label of more than six characters).

T Truncation of input error. The source statement exceeded 127 characters in length and was truncated.

U Undefined symbol error. A symbol used in an expression is undefined. In single-pass operation forward references and global symbols will be flagged with this error.

V Overflow error. A constant has a value greater than can be represented by a sixteen-bit number, e.g. 99999D.

W* Macro option error. A MACR or ENDM pseudo-op was encountered without the macro (M) option being specified. This is an abort error.

X* Memory mapping error. The symbol table limits are illegal; the lower limit must be greater than or equal to 300H and the upper limit must be greater than the lower limit (see Appendix 6). This is an abort error.

* These errors are abort errors. The assembly will be aborted and the error message printed on the console display.

7.12 Advanced operations

7.12.1 Assembling from devices other than disk.

During pass one of the assembly, the symbol table and external references are defined, the name of the module is defined and the external symbol linked-list is built. At the end of the first pass the source-input device must be "rewound" to allow the source code to be read again. This is performed automatically with disks, a new line on the console indicating that the second pass has commenced, but most other devices must be rewound manually. In this case the assembler outputs READY PASS2? to the console display and waits for any key to be pressed. This allows the user to perform what actions are necessary to rewind the source input device

and then press a key. The second pass then commences in the normal way.

7.12.2 Changing the symbol table size.

Each symbol in the assembler's symbol table occupies 9 bytes. If the system is in auto-mapping mode (see Appendix 6) then the symbol table size defaults to one-quarter of the total RAM size less 768 bytes. In the case of a 32K system this gives room for (8192-768)/9=824 symbols. If this is insufficient then the symbol tables limits may be redefined to allow up to approximately 2900 symbols in a 32K system. To do this, auto-mapping mode must be disabled (see Appendix 6) whereupon the assembler will request SYMBOL TABLE LIMITS? after the options have been entered. The lower limit must be greater than or equal to 300H and the upper limit less than the memory size minus 5K. For a 32K system the largest symbol table size is obtained by entering 300,6C00 cr.
Note that if a macro-handler or cross assembler has been loaded then the symbol table must not be enlarged in this way.

7.12.3 Using one output device with the assembler.

Normally, if both object code and assembly listing are required then these are output concurrently during pass 2 of the assembly. However, if only a single output device (e.g. paper tape punch) is available then it is necessary to output the object code at a different time from the assembly listing. This is easily accomplished by running the assembler twice, the first time specifying option K (suppress listing, i.e. object only) and the second time specifying options P N S (pass 2 only, suppress object output, include symbol table).

7.12.4 Single pass operation.

The assembler can be used as a single-pass assembler under the following restrictions:

1. No GLOBAL symbols are defined.
2. No forward symbol references occur.
3. The NAME pseudo-op is not used.

This can be useful to assemble data tables and certain types of programs. To use this mode, specify options P and R in addition to the normal ones required.

7.12.5 Obtaining a symbol table on its own.

A symbol table on its own can be obtained by specifying the options N K S. However, if there are any errors in the program the error lines will be sent to the console display and this has the side-effect of sending the

symbol table to the console display as well, usually with the unfortunate consequence of scrolling the errors off the screen! If there is any doubt that there may be errors, it is probably better to specify options N K. The symbol table can subsequently be obtained by typing E :TB cr and responding to the prompt TO? with the name of the destination device. If this is the screen then it is best to specify VP: as the symbol table may be large and will contain form-feed characters.
This facility is also useful if you forget to call for a symbol table and don't want to run the assembler again. Note that executing :TB if the RAM contents have been changed, or if the assembler had not previously been run, will produce "garbage" and you may need to reset the system.

7.12.6 Assembling several source modules together.

After the assembler prompt SOURCE FROM? you may specify several disk filenames separated by commas or spaces, in which case each will be read in turn and assembled as one module. The END statement must only occur in the last file. The system will automatically "rewind" to the start of the first file for the second pass. This can be useful in allowing standard lists of EQUates or subroutines to be stored as separate source files and incorporated only at assembly time, leaving a more managable file for editing. The default conditions governing the devices and filename extensions are the same as given for the PIP COPY command (see 5.5).

8. The Relocating Linking Loader

8.1 Introduction

The resident loader will load into RAM, and link together, both relocatable and non-relocatable object code files produced by the assembler. It allows separately assembled modules to be linked together and to be relocated to any address within the user's RAM memory. This enables a program designer to utilise a modular approach to software development. For example, a large program can be created as a collection of relatively short individual modules. These modules can be separately assembled and debugged and then combined into a complete program only when loaded. In many cases this approach can significantly reduce the amount of assembly and debug time required during program development.

The loader automatically links global symbols which provide communication between program modules. A global symbol is a symbol which is defined within one program module but can also be referenced by other program modules. As object files are loaded, a table containing global symbol references and definitions is built up. At the end of each module, the loader resolves all references to global symbols defined in that module. The symbol table can be printed to list all global symbols and their load addresses. The number of object modules which can be loaded by the loader is limited only by the amount of RAM available for the modules and the symbol table.

The beginning and ending addresses of each program module are displayed as it is loaded. The execute address (as optionally defined in the assembler END statement) is also printed for the first module loaded. The loader execute command (E) can be used to start execution at this address automatically.

The loader allows loading of both relocatable and non-relocatable modules. A non-relocatable module (defined with the assembler PSECT ABS pseudo-op) will always be loaded at the starting address defined by the ORG pseudo-op during assembly. Relocatable modules are loaded at an address which is the sum of the origin address of the module and the address immediately following the end of the previously loaded module. In general, relocatable modules should have an origin address of zero, achieved by omitting the ORG statement in the source code or by specifying ORG 0.
Note that the end address of a module is taken to be the last address whose contents are defined. If a module ends with an area of reserved storage (DEFS pseudo-op) then the loader will fail to take this into account and will overwrite the storage area with the next module (assuming it has an origin address of zero). This problem may be overcome by explicitly defining the last byte of the storage area using the DEFB pseudo-op (the actual data value is unimportant).

8.2 Executing the loader

To enter the loader from the system monitor the user types the load command L with one or more operands. If no operands are specified, the absolute loader in the monitor is executed instead. This will only load programs not requiring relocation or linkage to other modules (see 4.7.7).

```
.L aaaa,bbbbt
LOAD FROM? file1,file2,file3.....cr
```

The value aaaa is the offset address for the first module (the actual load address in the event of the first module being relocatable and having an origin of zero). The optional value bbbb is the start address of the loader's symbol table. The symbol table storage is allocated downward in memory from this address; if bbbb is omitted then the symbol table origin is determined by the memory size (see 8.4).
After the LOAD FROM? prompt the filenames of the object code modules should be listed, separated by commas or spaces, in the order in which they are required to be loaded. They may be a mixture of relocatable and non-relocatable modules. The extension defaults to .O, and the device to DK0:. If the device is changed within the list then this becomes the default device for the rest of the list. For example DK1:FILE1 FILE2 FILE3 DK0:FILE4 results in files 1,2 and 3 being loaded from disk 1 and file 4 from disk 0.

8.3 Loader commands

1. #Tcr Display the loader symbol table.

2. #Ecr Start execution at the execute address defined by the END pseudo-op of the first module loaded.

3. #cr Load the next object code module. If there are no more modules this command has no effect.

4. #. Exit the loader and return to the system monitor.

Note: # is the loader prompt character.

8.4 Loader symbol table

In the linking process the loader builds a symbol table of global references between program modules and resolves these references as each individual module is loaded. Eleven bytes are used for each entry in this table. If the symbol table start address is not specified as the second operand of the loader command, and the system is in auto-mapping mode (see appendix 6), then the start address is at the top of the first quarter of the RAM memory (i.e. 1FFFH in a 32K system). The symbol table grows downwards from this address until it reaches 0000 or until it meets the program being loaded. If the program being loaded resides above 2000H then there is space for about 743 symbols.

CAUTION: If the system is not in auto-mapping mode (for example, to allow the assembler's symbol table limits to be changed) then the loader's symbol table will default to the top of RAM minus 512 bytes. This address is in the area reserved for the disk operating system so the symbol table start address MUST be specified in the loader command whenever the system is not in auto-mapping mode. The specified address must not exceed 6C52.

8.5 Loader error messages

ERROR 1 Checksum error.

ERROR 2 Double definition of a global symbol.

ERROR 3 Attempt to overwrite loader symbol table.

ERROR 4 Attempt to load outside of available memory. The linking loader cannot be used to load into the VDU RAM at A000H or the scratchpad RAM at FC00H.

ERROR 5 Symbol table full. This is caused by the symbol table reaching the bottom of RAM (0000H).

Errors 3,4 and 5 are fatal and return control to the system monitor.

In the case of an ERROR 2 the second definition of the symbol is ignored.

If an ERROR 1 (checksum error) occurs in a data record, the address of the next location above where the last byte of the record was stored is printed on the console display. If a checksum error occurs in a non-data record (external symbol, internal symbol, relocating information, end-of-file or module definition) then no address is printed. Normally, if a checksum error occurs the load should be restarted from scratch. A genuine checksum error when loading from disk is a very unlikely occurrence; if ERROR 1 is obtained it should give rise to suspicion that the file being loaded is not in Intel format.

9. Examples of use

The following example illustrates the steps required to develop and run a very simple program.

9.1 Check disk contents

```
.E :PIcr
*Lcr
    Unused   1963.SECTORS
*esc
.
```

9.2 Enter program using the editor

```
.E :EDcr
FROM? cr
TO? TESTcr
>Tcr
<PTXT     EQU  0E3C7H    ;ROUTINE TO PRINT TEXTcr
<RENTRY   EQU  0E11DH    ;MONITOR RE-ENTRY ADDRESScr
<ETX      EQU  3         ;TEXT TERMINATORcr
<;cr
<TEST:    LD   HL,TEXT   ;POINT HL TO TEXT STRINGcr
<         LD   E,1       ;CHANNEL NO.  FOR CONSOLEcr
<         CALL PTXT      ;PRINT THE TEXTcr
<         JP   RENTRY    ;EXIT PROGRAMcr
<;cr
<TEXT:    DEFM 'TEST MESSAGE'cr
<         DEFB ETXcr
<;cr
<         END  TESTcr
<cr
>B Qcr
>esc
.
```

9.3 Assemble program

```
.E :AScr
SOURCE FROM? TESTcr
LISTING TO? cr
OBJECT TO? cr
OPTIONS?Scr

ERRORS = 0000
.
```

9.4 Load and run program

```
.L 0cr
LOAD FROM? TESTcr
BEG ADDR 0000
EXECUTE  0000
END ADDR 0017
UNDEF SYM 00
#Ecr
TEST MESSAGE
.
```

9.5 Edit the program

```
E :EDcr
FROM? TESTcr
TO? cr
>Zcr
0001 0013 0001
>L10 Vcr
0010 TEXT:   DEFM 'TEST MESSAGE'
>C /'/'ANOTHER / Vcr
0010 TEXT:   DEFM 'ANOTHER TEST MESSAGE'
>Zcr
0014 0014 0014
>esc
.
```

9.6 Delete and rename files

```
.E :PIcr
*L TESTcr
DK0:TEST  .S     3 SECTORS
DK0:TEST  .O     1 SECTORS
DK0:TEST  .L     8 SECTORS
DK0:TEST  .N     3 SECTORS
    Unused   1948 SECTORS
*D TEST.O,.Lcr
CONFIRM? Y
*R TEST.P=TEST.Scr
*R TEST.S=TEST.Ncr
*esc
.
```

APPENDIX 1

System Error Messages

The following error messages may be issued by the Operating System. They appear on the screen in inverse video and the system aborts the current task and transfers control to PIP (see section 5).

FILE NOT FOUND	An attempt was made to read, delete or rename a file not present on the specified disk.
FILE ALREADY EXISTS	An attempt was made to create (by writing or renaming) a file with the same name as one already present on the specified disk. Note that incorrect use of the wildcard character (?) in the Rename command can result in this error, even if no file of the given name existed prior to the command being issued.
ILLEGAL FILENAME	An attempt was made to create (by writing or renaming) a file whose name includes an illegal character, i.e. a control character, a question mark, DEL, or a character with bit 7 set.
DISK FULL	There is insufficient room on the disk to store the file.
DIRECTORY FULL	The specified disk's directory (index of files) is full. This is only likely to occur before the disk itself is full if there are many short files or if the disk has been heavily used and its files are very fragmented. In the latter case a cure can be effected by copying the entire disk contents onto a fresh disk using "MFT" or "BACKUP" (see Handbook part 3).
DIRECTORY ERROR	The disk's directory is invalid. This error may occur either when reading the directory from a disk (in which case the disk itself is faulty) or when re-writing the directory to a disk. The latter is most likely caused by one of the following: 1. The disk became non-ready whilst there was an open write file, and an attempt was subsequently made to close the file. This is often caused by a time-out whilst using the editor. 2. A user's program corrupted the disk directory area in RAM.
ILLEGAL TRACK NUMBER	The system attempted to access a track number greater than 76. Most commonly caused by trying to read an un-initialised disk or by reading a file which was never closed (i.e.

	is "open-ended").
DEVICE DOES NOT EXIST	The specified device mnemonic was not found in either the resident (PROM) or user (RAM) mnemonic tables.
NO OPEN FILE ON CHANNEL n	A program tried to write a character to a disk file after it was closed, or to one which had never been opened. Can be caused by failing to set bit 3 of register E when the first byte is written (see Appendix 4).
DISK FAIL -	Disk errors:
NOT READY	The disk is inserted incorrectly (e.g. upside down) or the gate is not properly shut.
WRITE PROTECTED	An attempt was made to write, rename or delete a file on a disk with a write-protect notch. Cover the notch with a foil label if necessary.
TRACK/SECTOR NOT FOUND	The disk controller is unable to locate the required sector. Usually means that the disk is severely worn or damaged (either physically or magnetically) and needs to be replaced.
CRC ERROR	There is a persistent cyclic-redundancy-check error. May indicate a worn or damaged disk but very occasional CRC errors are to be expected. An abnormally high occurrence of CRC errors may indicate adverse environmental conditions or poor quality disks.
LOST DATA	Data was lost on reading or writing. Can be caused by a severely damaged disk or by something slowing the execution speed of the processor, such as bus requests or non-maskable interrupts.

Persistent disk faults may be caused by a faulty disk drive, disk controller (CO10/2) or power supplier. They should be checked by substitution.

APPENDIX 2

System Diagnostic Fault Codes

Part 1 of this Handbook gives a full description of the self-test system and of the diagnostic fault codes. This is intended only as a quick reference list.

Fault Code	Probable cause
F0	UN26/31 self-test PROM checksum fail.
F1	UN26/31 RAM test fail.
F2	UN26/31 force CXXX function fail.
F3	Failure of UN27/13 RAM unit, or system bus.
F4	UN27/1 PROM unit fail.
F5	CO10/2 Disk Interface Unit or PS2/187G power supplier fail.
F6	GE8/505 VDU port fail.
F7	GE8/505 VDU RAM fail.
F8	CO4/6 I/O Unit parallel interface port fail.
F9	CO4/6 I/O Unit Z80-CTC fail.
FA	CO4/6 I/O Unit serial channel 1 fail.
FB	CO4/6 I/O Unit serial channel 2 fail.
CC	Bus or UN27/1 fault.
FD	UN27/1 PROM Unit checksum failure.
FE	System monitor entered directly, without a full self-test sequence having taken place.

APPENDIX 3

Object Code Definition

1. Each record of an object module begins with a delimiter (colon or dollar sign) and ends with carriage return, line feed. A colon (:) is used for data records and the end of file record. A dollar sign ($) is used for records containing relocating information and linking information. A standard Intel loader will ignore the records beginning with a dollar and will correctly load non-linkable programs that are not required to be relocated. All information is in ASCII.

2. Each record is identified by a "record type". The type appears in the 8th and 9th characters of the record (including the delimiter) and can take the following values:

   00 - data record
   01 - end-of-file
   02 - internal symbol
   03 - external symbol
   04 - relocation information
   05 - module definition

3. Data Record (type 00)

Character	Content
1	Colon (:) delimiter.
2-3	Number of data bytes in this record, expressed as ASCII-HEX (i.e. 1C signifies 28 bytes).
4-5	Most significant byte of the start address of the data (ASCII-HEX).
6-7	Least significant byte of the start address of the data (ASCII-HEX).
8-9	ASCII 00. This is the record type.
10-	Data bytes, each represented as ASCII-HEX.
Last two	Checksum of all bytes in the record (excluding delimiters), in ASCII-HEX. The checksum is calculated such that when each pair of ASCII characters in the record is converted to an eight-bit value, and these values (including the checksum) are summed, the result when evaluated modulus 256 is zero.

4. End-of-File Record (type 01)

Character	Content
1	Colon (:) delimiter.
2-3	ASCII 00.
4-5	Most significant byte of the execute address of the program as specified by the argument of the END pseudo-op in the program source code.
6-7	Least significant byte of the execute address.
8-9	ASCII 01 (record type).
10-11	Checksum.

5. Internal symbol record (type 02)

Character	Content
1	Dollar sign ($) delimiter.
2-7	Internal symbol name, left justified, blank filled.
8-9	ASCII 02 (record type).
10-13	Address of the internal symbol, ASCII HEX, most significant byte first.
14-15	Checksum (ASCII HEX). Note that the ASCII letters of the symbol name are taken in pairs and converted to 8-bit binary values as if they were HEX characters, without regard for errors. E.g. "AG" would be converted to 176 (decimal), being 10*16 + 16.

6. External symbol record (type 03)

Character	Content
1	Dollar sign ($) delimiter.
2-7	External symbol name, left justified, blank filled.
8-9	ASCII 03 (record type).
10-13	Address of last occurrence of external symbol in object module. This is the start of a linked list of the occurrences of the external symbol in the object module. The linked list is terminated by hex FFFF.
14-15	Checksum (ASCII HEX) calculated as for record type 02.

7. Relocation information record (type 04)

The addresses in the program which must be relocated are explicitly defined in these records. Up to 16 addresses may be defined in each record.

Character	Content
1	Dollar sign ($) delimiter.
2-3	Number of pairs of ASCII characters in the data part of the record, where two such pairs defines an address. This must always be an even number (ASCII-HEX).
4-7	ASCII 0000.
8-9	ASCII 04 (record type).
10-	Addresses which must be relocated, most significant byte first (ASCII-HEX).
Last two	Checksum (ASCII-HEX).

8. Module definition record (type 05)

Character	Content
1	Dollar sign ($) delimiter.
2-7	Module name, ASCII, left justified, blank filled.
8-9	ASCII 05 (record type).
10-11	Flag byte (ASCII-HEX). When converted to binary, bit 0 has the following significance: Bit 0 - 0 for absolute module (non-relocatable). 1 for relocatable module.
12-13	Checksum (ASCII-HEX).

9. Example of external symbol linked list.

```
$LABEL 03212A5A
   .
   .
:092008001F346B32FFFF912AC95D
   .
   .
:05212800CC13200C7631
   .
   .
:00000001FF
```

APPENDIX 4

User Device Drivers & Mnemonic Table

The user may write I/O drivers (and define mnemonics for them) for additional devices, and maintain compatibility with the system I/O protocol, by observing the following:

1. Register E is the control register, register D is the data register.

2. Register E - control register:
   Bit 7=1 implies "immediate return" mode. That is, if data or device is ready, perform the I/O. When the I/O is done, clear bit 7. If data or device is not ready, return immediately and leave bit 7 set.
   Bit 7=0 implies "wait until done" mode. Wait until I/O is complete before returning from the driver.
   Bit 3=1 implies initialise the device and/or rewind the file (this bit is set the first time the driver is called). Bit 3 must be reset before returning from the driver.
   Other bits of register E should be left unchanged.

3. On input, the data is returned in registers A and D. Only registers A, F, D and bits 3 and 7 of register E may be altered. All other registers must be left unchanged.

4. On output, the data is provided in register D. Register A should be set equal to register D on return from the driver. Only registers A, F and bits 3 and 7 of register E may be altered. All other registers must be left unchanged.

The new driver may be given a mnemonic by adding an entry to the user mnemonic table. This begins at address FF33 and each entry consists of four bytes, the first pair is the mnemonic in ASCII (1st letter first) and the second pair the associated address (LS byte first). The table is terminated by an 80H in the 1st letter position. Each time the system is reset, this table is initialised to contain eight entries (ED, AS, PI, TB, DK, PT, VP & BB) with the terminator at FF53. A new mnemonic may be added at FF53 and the 80H terminator moved to FF57, but bear in mind that this will be deleted if the system is reset.
Note that mnemonics which represent a memory address at which a two-byte value is stored have bit 7 of the second character of the mnemonic set to 1.

The number of mnemonics which can be added in this way is limited only by the available RAM. The memory space from FF54 to FF8F is reserved for user mnemonics (FF54 upwards) and stack (FF8F downwards). For every mnemonic added, the available stack space is reduced by four bytes. The stack requirements of system programs are such that only two additional mnemonics can safely be added.

APPENDIX 5

Resident Device Drivers

TTID Mnemonic TK:. Read a character from the terminal keyboard.
Calling address: 0E689H.
Immediate return mode is implemented.
Bit 7 of the keyboard data is masked out and set to 0.

CONIN The default console input driver (no mnemonic).
Immediate return mode is implemented.
This driver behaves the same as TK: except that certain system housekeeping functions are performed. In particular, the disk drive ready lines are sampled to determine whether a disk has been removed from the drive.

VDUOUT Mnemonic TT:. Send a character to the VDU screen.
Calling address: 0E6F9H.
Immediate return mode is not implemented.
The driver responds to various control characters as follows:

BACKSPACE (08H)	Move cursor one position to the left.
HORIZONTAL TAB (09H)	Move cursor one position to the right.
LINE FEED (0AH)	Move cursor down one line. Scroll screen if cursor was on bottom line.
VERTICAL TAB (0BH)	Move cursor up one line.
FORM FEED (0CH)	Home cursor (move to top left-hand corner).
CARRIAGE RETURN (0DH)	Move cursor to the beginning of the current line.
SHIFT OUT (0EH)	Subsequent characters stored with bit 7 set (inverse video).
SHIFT IN (0FH)	Subsequent characters stored with bit 7 cleared (normal video).
CANCEL (18H)	Clear screen from current cursor position to end.
ESCAPE (1BH)	Next letter output is displayed as a control character or graphics (line-drawing) symbol (see Handbook part 1).

All other control characters are ignored. If control characters are output with bit 7 set, they are stored rather than being acted upon (resulting in the same displayed characters as otherwise obtained by prefixing a letter with ESCAPE).

CONOUT The default console output driver (no mnemonic).
Immediate return mode is not implemented.
This driver behaves the same as TT: except that certain system housekeeping functions are performed and the following additional control characters are recognised:

PRINT (10H) This alternately enables and disables the printer. When enabled, everything which is sent to the console output channel also appears on the printer.

F1 (1FH) This unconditionally disables the printer and enables the VDU display (normal condition).

F2 (1EH) This disables the VDU display. If the printer is enabled, characters sent to the console output channel appear on the printer only. If the printer is not enabled, characters sent to the console output channel are "ditched".

F3 (1DH) This terminates "paging" mode (see F4)

F4 (1CH) This selects "paging" mode. The system pauses after 20 lines have been sent to the console output and waits for a keypress. If SPACE is pressed, one more line is displayed. If Q or F3 is pressed, paging is terminated. If any other character is pressed, the system displays 20 more lines.

VPAGDR Mnemonic VP:. Send a character to VDU, pause when 20 lines have been output ("video pager").
Immediate return mode is not implemented.
This driver behaves the same as TT: except that after 20 line-feeds have been output it waits for a key press before continuing. If SPACE is pressed, one more line is displayed. If Q is pressed, paging is terminated. If any other character is pressed, one more page (20 lines) is displayed. This driver also ignores form-feed characters (even after paging has been disabled) making it useful when copying to screen an assembly listing or other file containing form-feeds.

PRINT Mnemonic PT:. Send a character to the parallel printer.
This driver utilises the 8255A parallel interface device on the CO4/6 unit. Immediate return mode is implemented.

APPENDIX 6

Memory Mapping Options

After a system reset, the main memory of the system is "partitioned" into three main areas:

1. Symbol Table area (normally 0000-1FFF).
   This area is used both by the assembler and by the linking loader to store their symbol tables (and by the assembler as scratchpad memory). The asembler uses memory from 0000 upwards whereas the loader uses from 1FFF downwards.

2. Overlay area (normally 2000-6C52).
   This area is intended for assembler overlays, particularly cross-assemblers and macro-handlers.

3. Disk Directory and Buffer area (6C53-7FFF).
   This area is reserved for use of the operating system.

The total area from 0000 to 6C52 is available for the Text Editor, utility programs and user programs.
The default memory partitioning (so-called AUTO MAPPING MODE) is usually suitable for all normal purposes. Occasionally, however, it may be necessary to override this to provide, for example, more space for the assembler's symbol table. AUTO MAPPING MODE may be disabled by changing the value stored at memory location FF06 to anything other than 01:

```
.M FF06cr
FF06 01 00^
FF06 00 .
.
```

The main consequences of this change are as follows:

1. When the assembler is executed it requests SYMBOL TABLE LIMITS? (see 7.12.2).

2. The linking loader's symbol table defaults to the top of RAM and must therefore be entered explicitly as the second operand of the L command (see 8.4).

APPENDIX 7

Filename Extensions

The following extensions are commonly used but it should not be assumed that they always have the given meanings.

.A	"Acorn" cassette-format file.
.B	BASIC data file.
.C	Executable binary file.
.D	Documentation file (.L preferred).
.E	
.F	
.G	
.H	"Hewlett-Packard" format file.
.I	"I-format" file (for cassette).
.J	
.K	
.L	Listing file, intended to be printed.
.M	Music source file.
.N	"New" version of source file (editor output).
.O	Intel-format object code file.
.P	"Previous" version of a file.
.Q	
.R	
.S	Assembly language source file.
.T	Text processor source file.
.U	
.V	"Virgin" file, kept for archival purposes.
.W	
.X	
.Y	
.Z	Temporary file.