ZAPP

Z80 Assembly Programming Package

Features:

Comprehensive Text Editor

Single Pass High Speed Assembler

Single and Multi Step Monitor with Breakpoints

Symbolic Disassembler

Hex Memory Editor

File Manager

The Complete Machine Code Development Package
For The Amstrad CPC 464

HEWSON CONSULTANTS

ZAPP

Z80 Assembly Programming Package for the Amstrad CPC 464

© Hewson Consultants Ltd. 1985

ZAPP

Z80 Assembly Programming Package

Now you can develop professional quality machine code software on your Amstrad CPC464 using this fast and versatile assembly language programming package.

Features include:

* Editor – create, rearrange and modify your source code mnemonics quickly and easily using the comprehensive text editor.

* Assembler – all the usual assembler facilities plus temporary labels, some arithmetic operators and limited forward reference all at a single pass for high speed operation.

* Monitor – complete "front panel" controls plus single and multiple stepping and breakpoints.

* Disassembler – symbolic disassembly with external labels for full analysis of unknown object code.

* Hex Editor – hex and ASCII line-by-line dump with edit facilities.

* File Manager – full file handling facilities including append source file and assembly from file.

ZAPP was written by Keith Prosser of Hewson Consultants who are leading experts on the use of the Z80 microprocessor in home computer systems and publishers of many programs for the Amstrad CPC464 and the Spectrum Plus.

USER MANUAL

Version 0.2

PREFACE

ZAPP is designed to help you get your assembly language programs working. In addition to containing facilities to create, edit and assemble source programs. ZAPP has a powerful monitor with disassembler, single-stepping and 'front-panel' capability.

As with all powerful software tools it will take you some time to master all the facilities which ZAPP provides. To help you learn these facilities as quickly as possible this manual is divided into three sections.

Section A gives an overview of ZAPP, showing you how to enter, assemble and execute a short program and how to save the source and the object code. The section is designed to be a simple, readable account and should be read by beginners and experienced users alike.

Section B is the main description of ZAPP. In this section all the facilities of the program are defined and described in detail. Beginners should read this section carefully but experienced users may prefer to skim through it before reading section C.

Section C defines all the special words used in ZAPP and lists a summary of all commands. Most users will wish to keep the manual open at this section when working on the computer.

STOP PRESS!

DISC USERS

ZAPP will now run on the DDI-1 disc drive, the file handling commands being automatically re-directed to the disc.

Command	Effect
l tape	Turn on the tape system.
RUN"	Load and run ZAPP from tape.
*bye	Return to Basic.
l disc	Turn on the disc system.
SAVE "ZAPP"	Save ZAPP bootstrap program on disc.
SAVE "ZAPPc", b,32315, 10200	Save ZAPP main program on disc.

Disc users have the following additional command at their disposal

*era f

This command erases file f from disc.

CONTENTS

Section A

SYSTEM OVERVIEW

1.1 Background Material
1.2 Getting Started
1.3 Using the Editor
1.4 Making Changes
1.5 Using the Assembler
1.6 Using the Monitor
1.7 Saving Source Programs and Assembled Code

Section B

MAIN DESCRIPTION

1.0 STARTING UP

2.0 EDITING THE SOURCE FILE
2.1 The Autolister
2.2 Inserting Lines
2.3 Changing the Current Line
2.4 Editor Commands

3.0 THE ASSEMBLER
3.1 Instruction Format
3.2 Directives
3.3 Numbers
3.4 Symbols
3.4.1 Forward References
3.4.2 Permanent Symbols
3.4.3 Temporary Symbols
3.5 Operators and Offsets
3.6 Listings
3.7 End Report

3.8 Errors
3.8.1 Error Codes
3.8.2 Chaining Errors
3.9 Direct Assembly (Command ***dir**)
3.10 Assembly From File
(Commands ***asf, *asf+** and directive **file**)
3.11 Remote Assembly
4.0 THE MONITOR
4.1 Display Format
4.2 Monitor Commands
4.3 Performing Instructions
4.4 Modifying Registers
4.5 Breakpoints
4.6 Monitor Messages
5.0 THE DISASSEMBLER
6.0 THE HEX MEMORY EDITOR
7.0 FILE MANAGEMENT
7.1 Loading/Appending Source Programs
7.2 Saving Source Programs
7.3 Saving Assembled Code
7.4 Loading Code

Section C

REFERENCE SECTION

1.0 DEFINITIONS
2.0 "*" COMMAND SUMMARY
3.0 EDITING COMMANDS
4.0 ASSEMBLER DIRECTIVES
5.0 ERROR CODES
6.0 MONITOR COMMANDS

SECTION A

SYSTEM OVERVIEW

1.1 Background Material

It is assumed that you know a little about Z80 assembly language. If you know nothing at all about it, then you'll need a book to teach you, although ZAPP is the perfect system for learning on. There are several books available.

The standard reference is "How to Program the Z80" by Rodnay Zaks, published by Sybex and available through Radio Shack (i.e. Tandy Stores), ISBN No. 0-89588-057-1. It contains a great deal of information about the hardware organisation of the microprocessor as well as listing full details of the instruction set. The beginner might find it rather formidable because it runs to more than 600 pages.

A rather more readable account is contained in "Z80 and 8080 Assembly Language Programming" by Kathe Spracklen, published by Hayden, ISBN No. 0-8104-5167-0. The book starts at a more elementary level and covers the more important software aspects and ignores the hardware almost entirely.

1.2 Getting Started

Load ZAPP in the usual way. In the top left hand corner is a red/blue marker. At the bottom of the screen is the ZAPP message and the input prompt C> !. The C prompt indicates that you are in the "command/edit" mode (call it "C mode" for convenience).

At the moment you do not have a source program in the machine. The block is the "eof marker", marking the End Of the source File.

1.3 Using the Editor

To enter a program could hardly be easier. Simply type in an assembly language statement, for example:

C> **ld a,101**

You must leave a space between **ld** and **a** and press the ENTER key at the end of the line. Do not put any other spaces in the line.

If you type an invalid instruction then you must correct it because it will not be accepted. To see this type in

C> **ld cc,202**

The line will not be accepted when the ENTER key is pressed. Instead the C will change to R (for Re-enter) and a query will appear at the beginning of the assembly statement. Use the RIGHT arrow key to move the cursor past the first **c**, delete the letter using the DELETE key and replace it by a **b** so as to generate the corrected line:

R> **ld bc,202**

Press ENTER and the line will be accepted.

Enter the rest of this routine:

C> **ld a,101**
C> **ld bc,202**
C> **ld de,303**
C> **call fred**
C> **ret**
C> **fred:ld hl,40**
C> **ret**

Notice that everything is neatly "tabbed" in the listing for you and the lines are numbered, except where the **fred** label has been used. Notice also that the red/blue marker moves down as you enter each line. The marker identifies

the "current line" and each new line is inserted above the "current line".

1.4 Making Changes

You may use the UP, DOWN, LEFT and RIGHT arrow keys to move the current line marker up and down the listing. The line can then be edited using the alphanumeric keys to insert characters, the LEFT and RIGHT arrow keys to move the pointer and the DEL key to delete a character. The COPY key can then be used to copy the current line to the bottom of the screen.

To DELETE a line in the program make it current by using the UP or DOWN arrow keys and press DELETE followed by CLR.

Suppose that in the line

fred:ld hl,40

you wanted to change the number to **404.** In such a small program you could very easily use the ARROW keys to make it current, but for large programs (ZAPP can handle over 3000 lines), there are two convenient methods available.

The first method to make a line number current is by entering an equals sign (=) followed by the line number. Try entering

C> =6

and notice that the line is made current. =0 is very useful for getting to the end of a long program as it always makes the last line in the file (the eof line) current.

The second method is to ask ZAPP to search for a particular sequence of characters by entering a query sign (?)

followed by the sequence required. For example to make the line

fred:ld hl,40

current you could type any of these:

C> ?fred:
C> ?hl
C> ?40 etc.

A search always starts at the line below the current line and pressing ENTER repeats the search. Try typing

C> ?ld

and press ENTER several times to see the effect.

1.5 Using the Assembler

Having entered the program the command

C> *asm

assembles it. Assuming that no errors are found the message

FROM: 04BEh (1214)
TO: 04CDh (1229) (16 bytes)

is given, telling you the code was assembled at address 04CD (1214 in decimal) and that it occupies 16 bytes.

If an error is detected assembly stops and a flashing ? is displayed. There are ways described in Section B to correct the program and to continue assembling or you may press ESC, correct the error using the editor and then reassemble from the beginning.

1.6 Using the Monitor

It is possible of course to write a program in assembly language which is free of assembly errors but which does not

function as the programmer intended when it is executed. These kinds of errors are often called "bugs" and they can sometimes be extremely difficult to identify.

For this reason ZAPP includes a powerful single step and multi step monitor which will either step through a machine code program one instruction at a time or execute a group of instructions in one step if required. At each step the monitor displays the current status of the Z80 registers.

To invoke the monitor enter the command:

C> ***mon**

The prompt changes to M> to indicate the monitor is active. The monitor begins at the first instruction i.e. **ld a,101** and prints it in disassembled and hex form. Below this is printed the values in the registers and flags. The format is:

```
instruction                    address hexbytes
    xAFx    xBCx    xDEx    xHLx    flags
    xIXx    xIYx    xSPx    (SP)
```

(SP) is the top item on the stack.

Pressing ENTER causes the instruction to be performed. If you have the routine listed in Section 1.2 in your machine, assemble it, enter the monitor and press ENTER three times, observing the values in the registers changing as each instruction is executed. Note that the numbers in the disassembly are in decimal, but the registers are displayed in hex.

You should now have the instruction **"call fred"** displayed. Press ENTER again. The instruction **"fred:ld hl,404"** is displayed because pressing ENTER for a **"call"** or **"rst"** instruction causes the called routine to be executed one step at a time just as if it were part of the main program.

Alternatively pressing ENTER + CAPS SHIFT will cause the monitor to execute all the steps in the called routine at one go. This is very useful for skipping over routines you know you can trust.

Press ENTER twice more. Performing the final **"ret"** returns you to C mode with the message:

EXIT OK

indicating all is well.

1.7 Saving Source Programs and Assembled Code

Having written and checked the routine the next job is to save it. You may save your source programs or the assembled code.

To save the source program the command is:

C> ***ssr filename**

You can save the object code program using the command:

C> ***scd filename**

You may use any filename you choose up to 16 characters, not including a comma or semicolon.

There are many more facilities in ZAPP described in the next section. You will find that ZAPP is of enormous help in developing machine code programs, whether you are an experienced programmer or a complete novice.

SECTION B

Main Description

1.0 STARTING UP

Load ZAPP in the normal way. At the top of the screen is the EOF marker. At the bottom is the ZAPP message and the prompt:

C> !

The letter in the prompt (C in this case) indicates the mode in which ZAPP is operating. C indicates "command/ edit" mode, or C mode. This mode is used for editing the source program, to access the other modes and to enter "*" commands.

2.0 EDITING THE SOURCE FILE

The editor maintains the source program as a list of lines. At any time there is a "current line", indicated by a red/blue block, where editing takes place. The editor is controlled by the following keys:

COPY	Modify current line
LEFT ARROW	Move current line pointer up about one screenful.
RIGHT ARROW	Move current line pointer down one screenful.
DOWN ARROW	Move current line pointer down one line.
UP ARROW	Move current line pointer up one line.
DELETE then CLR	Delete current line.

The above keys only have the indicated effect if they are the first key pressed in a line, i.e. when the "!" is displayed.

The LEFT and RIGHT ARROW and DELETE keys are also used to correct a line in the editing area at the bottom of the screen. Pressing CLR clears the line.

2.1 The Autolister

The autolister lists part of the source program containing the current line whenever the source is edited. Pressing ENTER on its own also forces an autolisting.

2.2 Inserting Lines

Valid assembly instructions (i.e. those not containing a syntax error) are inserted into the source program immediately above the current line. If an error is detected in the line then it is not accepted and must be corrected.

2.3 Changing the Current Line

This may be done using the ARROW keys or particular lines may be made current:

by LINE NUMBER

Type the line number (given in the listings) preceded by a "=", e.g.

C> **=100**

Line 0 is conventionally the last line in the file (i.e. the eof line), so

C> **=0**

is a quick way of getting to the end of the program.

by CONTEXT

Type the item to be located preceded by a **?**, e.g.

C> **?string**

The search begins with the line BELOW the current line for any line containing the string. If the string is not found the current line is not changed. The search may be repeated as often as required by pressing ENTER.

2.4 Editor Commands

*del n1,n2 – delete lines n1 to n2 inclusive (n2=0 for end of file).

*eof – print highest address used by source file.

*new [n] – delete source program from memory and reset base of source area to n.

3.0 THE ASSEMBLER

3.1 Instruction Format

Instructions take the form:

[LABEL:] MNEMONIC [;COMMENT]

or ; COMMENT

The items in square brackets are optional. The ZAPP assembler recognises the standard Z80 instruction mnemonics, with the exception that ex af,af' must be written without the quote, i.e. ex af,af. ZAPP recognises the short forms **add s, adc s** and **sbc s** for **add a,s, adc a,s** and **sbc a,s.**

3.2 Directives

In addition to the Z80 instructions ZAPP also recognises the following directives:

defb n – assemble as the byte value n

defw nn – assemble as the double byte nn (low byte first)

org nn	– assemble subsequent instructions at address nn and above
defm "string	– assemble as the character codes for the characters in the string specified
list	– enable listing (DEFAULT)
nlst	– disable listing
prnt	– send listing to printer
scrn	– send listing to screen (DEFAULT)
base	– set base address for saving code
label:equ nn	– set symbol equivalent to number nn
file filename	– assemble from file

3.3 Numbers

Numbers may be written in decimal, hex, character code or represented by a symbolic name (symbol). Hex numbers must be preceded by an ampersand or begin with a digit and end with a 'h', e.g.

&7FFE or 0DF10h

Most character codes can be represented by their associated character preceded by a double quote, e.g.

ld a,"a is equivalent to ld a,65

Index register offsets must be decimal in the range –128 to +127.

3.4 Symbols

Symbols (or labels) are mnemonic names given to numbers to make the program easier to understand and write. A symbol is defined, (i.e. made equivalent to a number) by using it as a label. In the instruction

label:equ nn

the symbol used as the label becomes the same number nn.

In any other instruction the label symbol becomes equivalent to the assembled address of the instruction, e.g.

```
43          org 9000h
        fred:ld a,100
```

the symbol **"fred"** is made equivalent to the value 9000h.

3.4.1 Forward References

Symbols may appear as operands before they are defined, e.g.:

```
37          inc a
38          jr nz,jim
39          ld b,45
        jim:ld c,45
```

In this case the use of **"jim"** in the jr instruction (line 38) precedes the definition of jim in the line **jim:ld c,45** (line 40). The use of a symbol before it is defined is called a "forward reference". Forward references may be used freely in ZAPP, except that they may not take an offset, or be used where 8 bit data is required, except for relative addresses.

3.4.2 Permanent Symbols

Symbols like **"fred"** and **"jim"** above are called "permanent symbols", because they stand for the same value throughout a program. They may be defined once only. Such symbols must begin with a letter and may consist of up to six characters. Comma, semicolon and space may not be used in symbols.

3.4.3 Temporary Symbols

There are ten "temporary" symbols, represented by the digits 0-9. They are very useful because they save the need

to invent unique names for trivial loops, etc and because they save space in the symbol area. They may be redefined any number of times in one program, only the most recent use of the symbol being active.

For example:

```
    2:inc  hl                First:  inc  hl
72    cp   (hl)              72      cp   (hl)
73    jr   nz,:2             73      jr   nz,First
74    ld   a,(de)      =     74      ld   a,(de)
    2:cp   (hl)              Second: cp   (hl)
76    inc  hl                76      inc  hl
77    jr   z,:2              77      jr   z,Second
```

Note that temporary symbols when used as operands are preceded by a colon to distinguish them from integers.

To make a forward reference with a temporary symbol add an "f" as a suffix. e.g.:

```
    3:cp   (ix–1)
48    jr   z,:3f
49    dec  hl
    3:cp   (hl)
```

The symbol :3f refers to the label 3: on the cp (hl) instruction. Using the forward reference means the label on the 3:cp (ix–1) instruction is no longer accessible.

3.5 Operators and Offsets

The addition and subtraction operators (+ and –) may be used freely in ZAPP except in an operand which is a forward reference. The following instruction for example is illegal if **fred** is a forward reference.

```
    jr    fred + 10
```

The most significant byte and least significant byte operators are > and < respectively, i.e.

>1234h = 12h

<1234h = 34h

3.6 Listings

The listing will normally be sent to the screen or printer as set by ***prnt** and ***scrn** commands. Switching the listing off using the **nlst** directive greatly increases assembly speed. The directive **list** switches listing back on.

The listing displays the instruction, the assembly address and the hex bytes that make up the machine code instruction. However for relative branches the absolute address is printed in brackets, e.g.

```
13      jr      1000h          1003 18<1000>
```

For forward references the assembled bytes are not known and such bytes are indicated by an asterisk (*) after the address.

3.7 End Report

At the end of assembly any forward references left undefined are printed. This is followed by the report:

```
BASE: xxxxh (ddddd)
TOP:  xxxxh (ddddd) (ddddd bytes)
```

The **BASE** address is set by the ***asm** command or **base** directive.

The **TOP** address is the last byte assembled. Unless the org directive is used to plant code outside these addresses then these addresses are the lowest and highest used by the assembled code. The **BASE** and **TOP** addresses are used by many commands as the default values if parameters are omitted.

3.8 Errors

If an error is detected at assembly time then assembly stops with a flashing ? and a two digit hex code. At the bottom of the screen an X> input prompt is displayed, and the following keys may be used:

COPY – correct and retry line.
ENTER – continue with next line
ESC – abandon assembly

Some errors return immediately to the C prompt as the assembled code may be corrupt even after correcting the error.

If any errors remain uncorrected after assembly then the message:

ERROR(S) IN ASSEMBLY

is given, with the End Report.

3.8.1 Error Codes

The following two digit codes are given with a flashing ? to indicate an error.

?00 – no statement with label
?E0 – no label defined for this temporary symbol
?E1 – label is a register name etc.
?E2 – permanent label already defined
?F1 – error in first operand
?F2 – error in second operand
?77 – unrecognised instruction
?DD – invalid operands.

The DD error can be generated in the following cases:

1) an illegal register or number (e.g. sbc, de,bc or im 4)

2) a forward reference for 8 bit data
3) a forward reference with an offset

?FD – relative address range exceeded.
?FE – chaining error
?FF – chaining error

3.8.2 Chaining Errors

ZAPP is a "one pass" assembler, i.e. it works through the source file once only, creating all the object code as it goes. This makes it faster than other assemblers but it means that "chaining errors" can sometimes occur. ZAPP chains together forward references to symbols and then returns to fill them when the symbol is defined. If at some stage ZAPP cannot complete the chain (because for example it needs to store a relative address greater than FF in a one byte location) then a chaining error occurs. In the following example a chaining error will occur if there are more than FF bytes in the object code between the **"jp tom"** and the **"jr tom"** instructions.

```
      jp      tom
      " " " " " " "
      " " " " " " "
      jr      tom
      " " " " " " "
      " " " " " " "
tom:  ld      a.100
```

The best cure in this case is to use a second symbol and equivalence it to the main symbol when it is defined, e.g.:

```
      jp      tom
      " " " " " " "
      " " " " " " "
      jr      tom1: do this only if jr tom causes a chaining
      " " " " " " "                 error.
      " " " " " " "
```

```
tom  : ld    a,100
tom1: equ    tom
```

3.9 Direct Assembly (Command *dir)

In direct assembly mode assembly instructions entered at the keyboard are assembled immediately. Direct assembly mode is indicated by the prompt letter D. To exit from Direct mode type the directive **"end"** or press ESC.

3.10 Assembly From File (Commands *asf, *asf+ and directive file)

ZAPP has the ability to assemble source programs saved on disc or tape. This allows very large source files to be assembled. You may not use the **'file'** assembly directive within source programs that are themselves assembled from file.

3.11 Remote Assembly

In remote assembly the code is not planted at the address where it will be executed. For example a routine can be developed in high RAM and finally be remotely assembled to be loaded and run in the RAM occupied by ZAPP. Note that code which has been remotely assembled should not normally be run or single stepped using ZAPP.

4.0 THE MONITOR

4.1 Display Format

The monitor displays instructions in disassembled and hex form, together with the address and the register values BEFORE the instruction is performed. The format is:

```
[LABEL:] instruction          address hex-bytes
xAFx    xBCx    xDEx    xHLx    cf zf pf sf
xIXx    xIYx    xSPx    (SP)
```

If there is a symbol defined as the address of the instruction it is printed as a label. 16 bit numbers are printed in decimal. Relative addresses are shown as the absolute address in hex thus:

djnz fred<xxxx>

The address and bytes are shown in hex.

xAFx etc. stands for the value of the indicated register pair in hex. cf, zf, pf and sf stand for the values of the carry, zero, parity and sign flags.

4.2 Monitor Commands

The monitor is controlled by the following commands and keys:

ENTER	– perform instruction
ENTER + CAPS SHIFT	– perform call/rst subroutine
no	– skip instruction
af nn	– set af register pair to nn
bc nn	– set bc register pair to nn
de nn	– set de register pair to nn
hl nn	– set hl register pair to nn
sp nn	– set stack pointer to nn
pc nn	– set program counter to nn
a nn	– set register a to nn
fc	– complement carry flag
fz	– complement zero flag
fp	– complement parity flag
fs	– complement sign flag
*hex nn	– display/edit a hex dump of memory
run	– run program

call nn	– call routine
brk [n,][addr]	– set/display breakpoints
brun	– break and run

4.3 Performing Instructions (ENTER or CAPS SHIFT + ENTER)

Pressing ENTER performs the instruction. **"call"** and **"rst"** instructions continue by single stepping through the called routine.

Pressing CAPS SHIFT + ENTER has the same effect as ENTER for instructions other than **call** or **rst.** For these instructions it causes the call to be performed, executing the routine before returning to the monitor. Note that in this case the return address is inside ZAPP and the routine should not use the return address internally.

4.4 Modifying Registers (a, af, bc, de, hl, sp, ix, iy, pc)

After modifying a register or flag the monitor redisplays the instruction and registers. If the number is omitted or invalid then the register is not changed.

4.5 Breakpoints (brk, run, brun)

Breakpoints are used to restart the single step monitor after the monitor "run" command. This is particularly useful for testing loops etc., for example:

```
        start: ld      c,"a
2              ld      b,26
            0: ld      a,c
4              call    & bb5a
5              inc     c
6              djnz    :0
         fini: ret
```

This routine prints the alphabet. Single stepping through is tedious and unnecessary once you are sure the loop works correctly. After going round the loop a few times, the command:

M> **brk fini**

sets a breakpoint at the address of the "ret" instruction. The command

M> **run**

runs the routine from the point reached up to the normal exit or a breakpoint. If a breakpoint is met single-stepping restarts.

Up to four breakpoints may be set, numbered 0, 1, 2 and 3. If no breakpoint number is specified breakpoint 0 is used. If no parameters are used with the command the current breakpoint addresses are displayed only.

The command **"brun"** means "break and run" and sets a breakpoint (number 3) at the address of the next instruction and then "runs" the routine.

4.6 Monitor Messages

The following messages are generated by the monitor:

1) SP WARNING

This indicates that more values have been popped from the stack than have been pushed onto it since the monitor was last entered.

This is a warning only and does not stop the monitor.

2) EXIT OK

This message indicates that the routine has terminated correctly.

3) BAD EXIT – STACK ERROR

This message indicates that the stack is not correct although the routine has terminated.

4) BAD OPCD AT xxxxh (ddddd)

This indicates that the instruction at the indicated address is not a valid Z80 instruction.

5.0 THE DISASSEMBLER

The disassembler is available using the command ***dis** in C mode. Starting at the specified address the disassembler begins disassembling the code, giving the instruction, address and hexbytes. The disassembly may be sent to the printer by using the ***prnt** command.

6.0 THE HEX MEMORY EDITOR

The contents of the memory may be edited by the command *hex in C mode or from the monitor. The editor displays up to 8 bytes on a line, each line terminating at an address xxx7h or xxxfh. A dump of the characters for the codes between 32 and 127 is also given.

At the end of each line displayed the prompt H> is given, pressing ENTER displays 8 more bytes. Pressing ESC exits from the editor.

Typing a number after the H> prompt allows the memory to be altered beginning at that address. The address and current contents of the byte are displayed and you can alter the contents by pressing two hex digits. Pressing ENTER moves onto the next byte. Pressing UP-ARROW moves to the previous byte. Pressing ESC exits from the editor.

7.0 FILE MANAGEMENT

7.1 Loading/Appending Source Programs

The command *lsr f loads a source file, erasing any source in RAM. The command *asr f loads a source file and appends it – i.e. adds it to the end of a source in RAM.

The message

Not source format

is given if the loaded file does not appear to be a valid source program.

7.2 Saving Source Programs

The command

*ssr f

is used to save source programs.

7.3 Saving Assembled Code

The command

*scd f[,n1][,n2]]

saves the area of memory between the specified addresses. It is not possible to save code in auto-run form from ZAPP – you must return to BASIC and save it from there. n1 defaults to the BASE address, and n2 to the TOP address.

7.4 Loading Code

The command

```
*lcd f,n
```

loads the specified file at the address given.

Typing a number after the HD> prompt causes the memory to be altered beginning at that hexadecimal address and current contents of the byte are displayed and you can alter the contents by pressing two hex digits. Pressing ENTER moves onto the next byte. Pressing UP ARROW moves to the previous byte. Pressing ESC exits from the editor.

7.0 FILE MANAGEMENT

7.1 Loading/Appending Source Programs

The command *lst f loads a source file, erasing any source in RAM. The command *app f loads a source file and appends it to the end of a source in RAM.

The message

Not source format

is given if the loaded file does not appear to be a valid source program.

7.2 Saving Source Programs

The command

*sav f

is used to save source programs.

7.3 Saving Assembled Code

The command

*sav f[,a1][,a2]

saves the area of memory between the specified addresses. It is not possible to save code in auto-run form from ZAP - you must return to BASIC and save it from there. a1 defaults to the BASE address, and a2 to the TOP address.

SECTION C

QUICK REFERENCE SECTION

1.0 Definitions

This part of the manual is for instant reference to the facilities in ZAPP. Each command is described as follows:

1) The command and its format.
2) The use of the command.
3) The parameters and defaults.

The parameters are either numbers (m, n1 and n2) filenames (f) or line numbers (L1, L2). Filenames are used in saving, loading and veryifying commands. Filenames may be omitted for loading and verifying.

Numeric parameters may be written as a decimal, hex or symbol, with offset. Symbols must be defined. The form (nn) may also be used. In this case the number passed as a parameter to the command is the contents of the 16 bit location whose address is given by nn.

In the following description a numeric parameter in square brackets, i.e. [nn] is optional. The default value depends on the command in question.

The message

Parameter error

is given when the parameter passed is badly formed, an undefined symbol or out of range.

2.0 "*" COMMAND SUMMARY

*asf f[,[n1][,n2]] Assemble source file f planting code at address n1. (Default = eof + 128). If n2 given then code is generated suitable for relocation at that address.

*asf+ f	Assemble source file f, continuing from end of last assembly.
*asm [n1][,n2]	Assemble source program in memory. n1 and n2 as for *asf.
*asm+	Assemble source program in memory, continuing from end of last assembly.
*asr f	Append source file f to source program in memory.
*bye	Return to BASIC. (Re-enter ZAPP using CALL 36000).
*cat	Catalogue tape or disc.
*del L1,L2	Delete lines L1 to L2 inclusive (L2 = 0 for end of program).
*dir [nn]	Enter 'direct assembly' mode (default = TOP + 1).
*dis [nn]	Disassemble memory at nn (default = BASE).
*do [nn]	Call machine code routine at nn (default = BASE).
*eof	Give top address of end of source program
*lcd f,n	Load file f at address n
*lsr f	Load source program f
*mon [nn]	Enter monitor with PC = nn (default = BASE)
*new [nn]	Delete source program and reset base address of source area to nn (default = no change)
*num nn	Print value of nn in hex and decimal
*prnt	Direct output to printer

*scd f[,[n1][,n2]]	Save memory from n1 to n2 inclusive (defaults n1 = BASE, n2 = TOP)
*sps nn	Set stack pointer for running or monitoring code routines
*ssr f	Save source program
*syms	Display permanent symbols and their values
*call [nn]	Call routine at nn (default = BASE). (Disabled if errors or source program changed).

3.0 EDITING COMMANDS

The editor maintains the source program as a list of lines. At any time there is a "current line", indicated by a red block, where editing takes place. The editor is controlled by the following keys:

COPY	Modify current line
LEFT ARROW	Move current line pointer up about one screenful.
RIGHT ARROW	Move current line pointer down about one screenful.
DOWN ARROW	Move current line pointer down one line.
UP ARROW	Move current line pointer up one line.
DELETE the CLR	Delete current line.

The above keys only have the indicated effect if they are the first key pressed in a line, i.e. when the "!" is displayed. The LEFT and RIGHT ARROW and DELETE keys are used in the middle of a line to correct the line.

Pressing CLR clears the line.

Editor Commands:

*del n1,n2 – delete lines n1 to n2 inclusive (n2 = 0 for end of file).

*eof – print highest address used by source file.

*new[n] – delete source program from memory, resetting start address of memory.

4.0 ASSEMBLER DIRECTIVES

In addition to the Z80 instructions ZAPP also recognise the following directives:

defb n	– assemble as the byte value n
defw nn	– assemble as the double byte nn (low byte first)
org nn	– assemble subsequent instructions at address nn and above
defm "string	– assemble as the character codes for the characters in the string specified
list	– enable listing (DEFAULT)
nlst	– disable listing
prnt	– send listing to printer
scrn	– send listing to screen (DEFAULT)
base	– set base address for saving code
label:equ nn	– set symbol equivalent to number nn
file filename	– assemble from file

5.0 ERROR CODES

?00 – no statement with label
?E0 – no label defined for this temporary symbol
?E1 – label is a register name etc.
?E2 – permanent label already defined
?F1 – error in first operand
?F2 – error in second operand
?77 – unrecognised instruction
?DD – invalid operands.
?FD – relative address range exceeded.
?FE – chaining error
?FF – chaining error

6.0 MONITOR COMMANDS

ENTER – perform instruction
ENTER +
CAPS SHIFT – perform call/rst subroutine
no – skip instruction
af nn – set af register pair to nn
bc nn – set bc register pair to nn
de nn – set de register pair to nn
hl nn – set hl register pair to nn
sp nn – set stack printer to nn
pc nn – set program counter to nn
a nn – set register a to nn
fc – complement carry flag
fz – complement zero flag
fp – complement parity flag

fs	– complement sign flag
*hex nn	– display/edit a hex dump of memory
run	– run program
call nn	– call routine
brk [n,][addr]	– set/display breakpoints
brun	– break and run

Printed by Powage Press, Aspley Guise, Milton Keynes MK17 8HF.

ZAPP

Z80 Assembly Programming Package

Now you can develop professional quality machine code software on your Amstrad CPC464 using this fast and versatile assembly language programming package.

Features include:

* Editor – create, rearrange and modify your source code mnemonics quickly and easily using the comprehensive text editor.

* Assembler – all the usual assembler facilities plus temporary labels, some arthimetic operators and limited forward reference all at a single pass for high speed operation.

* Monitor – complete "front panel" controls plus single and multiple stepping and breakpoints.

* Disassembler – symbolic disassembly with external labels for full analysis of unknown object code.

* Hex Editor – hex and ASCII line-by-line dump with edit facilities.

* File Manager – full cassette handling facilities including append source file and assembly from file.

ZAPP was written by Keith Prosser of Hewson Consultants who are leading experts on the use of the Z80 microprocessor in home computer systems and publishers of many programs for the Amstrad CPC464 and the Spectrum Plus.