CHAMP USER'S MANUAL

Champ is designed to run on the Commodore 64, BBC Micro model B, and Sinclair Spectrum 48K.

It comprises an assembler for 6502/6510 or Z80 assembly language, a program editor, and a monitor/debugger/disassembler. These facilities make Champ a powerful aid to the assembly language programmer.

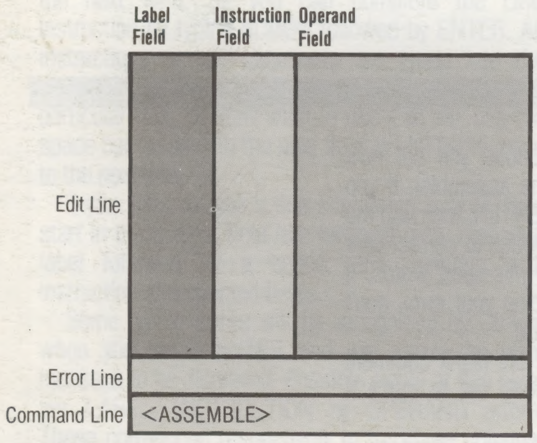
LOADING CHAMP

BBC Model B — CHAIN“”
C64 — Hold down [SHIFT] and hit [RUN/STOP]
Spectrum 48K — LOAD“”

Champ will auto-run when loading is complete, so, having issued the LOAD command, you need do nothing until the screen clears and displays the copyright message. Stop the tape, remove it, and replace it with a blank data tape if you intend to save program files from Champ.

In addition to the copyright message on the screen, you will see a message about Champ's location in memory; this is important data, so make a note of it all now, even if you're not sure what it's for. When you've done that, hit [ESC] to run Champ. (This refers to the BBC Micro; the corresponding key presses for the Commodore 64 and Spectrum are listed in the Key Conventions panel.)

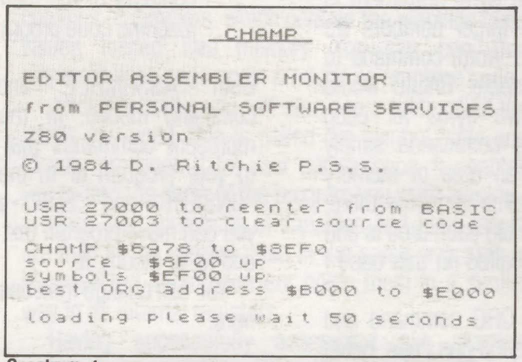
The screen should look like this:



At this point, the computer is waiting for you to type in an assembly language program, but don't do anything yet.

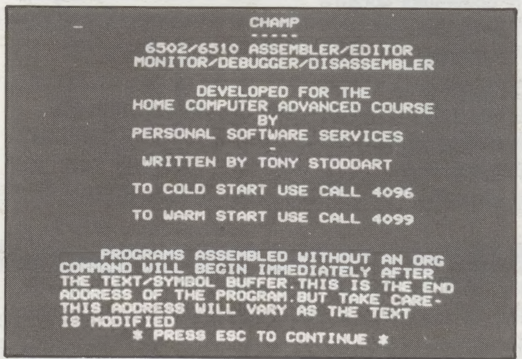
Have a look at the two example programs given overleaf, and choose the one for your computer. (The 6502 program for the BBC and Commodore 64 machines, the Z80 for the Spectrum). The two programs are equivalent.

The first thing to notice is that there are a lot of semi-colons (;) about. These may look strange, but are simply the equivalent of REM in BASIC. In machine code programming lines beginning with ';' are known as COMMENTS, and they are extremely important if you want to understand something you may have written weeks ago. You can put anything you like inside a comment without affecting the program. The comment must begin on a new line. We have put the equivalent BASIC program lines in comments so that you can see how the machine code instructions can be made to operate in the same way as BASIC.



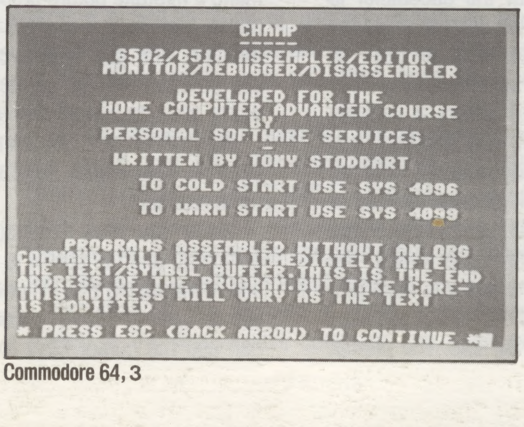
```
CHAMP

EDITOR ASSEMBLER MONITOR
from PERSONAL SOFTWARE SERVICES
Z80 version
© 1984 D. Ritchie P.S.S.

USR 27000 to reenter from BASIC
USR 27003 to clear source code

CHAMP $6978 to $8EF0
source   $8F00 up
symbols  $EF00 up
best ORG address $B000 to $E000

loading please wait 50 seconds
```

Spectrum, 1



```
CHAMP
-----
6502/6510 ASSEMBLER/EDITOR
MONITOR/DEBUGGER/DISASSEMBLER

DEVELOPED FOR THE
HOME COMPUTER ADVANCED COURSE
BY
PERSONAL SOFTWARE SERVICES
-
WRITTEN BY TONY STODDART

TO COLD START USE CALL 4096

TO WARM START USE CALL 4099

PROGRAMS ASSEMBLED WITHOUT AN ORG
COMMAND WILL BEGIN IMMEDIATELY AFTER
THE TEXT/SYMBOL BUFFER.THIS IS THE END
ADDRESS OF THE PROGRAM.BUT TAKE CARE-
THIS ADDRESS WILL VARY AS THE TEXT
IS MODIFIED
* PRESS ESC TO CONTINUE *
```

BBC Micro, 2



```
CHAMP
-----
6502/6510 ASSEMBLER/EDITOR
MONITOR/DEBUGGER/DISASSEMBLER

DEVELOPED FOR THE
HOME COMPUTER ADVANCED COURSE
BY
PERSONAL SOFTWARE SERVICES
-
WRITTEN BY TONY STODDART

TO COLD START USE SYS 4096

TO WARM START USE SYS 4099

PROGRAMS ASSEMBLED WITHOUT AN ORG
COMMAND WILL BEGIN IMMEDIATELY AFTER
THE TEXT/SYMBOL BUFFER.THIS IS THE END
ADDRESS OF THE PROGRAM.BUT TAKE CARE-
THIS ADDRESS WILL VARY AS THE TEXT
IS MODIFIED
* PRESS ESC (BACK ARROW) TO CONTINUE *
```

Commodore 64, 3

Opening display 1,2,3
The opening display of Champ shows the warm start and cold start addresses of the Champ program. If you quit to BASIC and want to return to Champ you should use PRINT USR (addr), SYS (addr) or CALL (addr), where addr is the appropriate warm or cold start address. (A cold start is what you get after switching your machine off; it clears Champ's buffers of any source code you have entered. A warm start restarts Champ as it was when you quit it.)

Any line that does not start with a ';' is an assembly language statement. The first one is ORG $CØØØ. This is not a machine code instruction; it tells the assembler where to put your program when it turns your assembly language into machine code. ORG is short for ORIGIN; $CØØØ is a hexadecimal numeral (signified by the '$' sign), equivalent to 49,152 decimal. In BASIC, you don't need to worry about where your program is because the interpreter looks after all that for you. Now, you have the whole of the computer under your control, and that means that you decide where you want your program to go.

You also have to tell the assembler where to put your variables. Once again, BASIC does all this for you, but in machine code you have to do it yourself. For large machine code programs it is usually a good idea to put all your variables together in one block, but for small programs it's all right to put them next to the appropriate part of the program for clarity. The next few assembly language statements in our example tell the assembler that you want to use two variables, I and J, and that they will be found at the beginning of the program, right after the ORIGIN address. Because we are not going to use numbers larger than 255, we need only one byte for each of the two variables, so the DB (Define Byte) statement is used. If we had wanted to use larger numbers we could have used the DW (Define Word) command to reserve two bytes for each variable. BASIC would automatically have used up five bytes for each variable. These storage-defining commands simply tell the assembler not to use an area of memory because you're going to store some variables there. They also tell the assembler how big each area is and what the areas are going to be called (in this case I and J).

The storage commands, the ORG command and the COMMENT are all called pseudo-ops ('ops' being short for 'operations') because the assembler doesn't generate any machine code from them; they are just there for your (and the assembler's) attention.

All the other assembly language statements will be translated by the assembler into executable machine code instructions — that is, instruction codes that will cause the microprocessor to do something. Among these instructions are: Load A Register; Test a Flag; and Jump or Branch to a new address. These make up the rest of the example program listing.

If you're not too clear about any of the three types of pseudo-ops (COMMENT, ORG or storage) then please re-read this section. It's not at all difficult, once you get the hang of it, but do take it at your own pace.

Now that we have covered the different types of statement understood by the assembler, you can enter the example program. Please refer to the relevant Hands On section below.

Operating Modes

<ASSEMBLE> mode
: is used after you have typed in an assembly language program, in order to assemble it into machine code

<INSERT> mode
: is what you use to type in an assembly language program

<EDIT> mode
: enables you to modify an existing assembly language program

<DEBUG> mode
: allows the inspection or modification of the contents of the memory, or the execution of a machine code program

Both <ASSEMBLE> and <DEBUG> modes are command modes. In these modes various keys represent commands that make something happen to your program or to memory. On the other hand, <INSERT> and <EDIT> are text modes; with these you can move program text around on the screen, and add to, or modify, it.

You can change from one mode to another as shown here.

$$\begin{tikzcd}
\text{<EDIT>} \arrow[r, shift left, "\text{[RET]}"] \arrow[d, shift right, "\text{[ESC]}"'] & \text{<INSERT>} \arrow[l, shift left] \\
\text{<ASSEMBLE>} \arrow[u, shift right] \arrow[d, shift right, "\text{[M]}"'] & \\
\text{<DEBUG>} \arrow[u, shift right, "\text{[A]}"'] &
\end{tikzcd}$$

KEY CONVENTIONS

A letter (or letters) enclosed in square brackets, thus [A], means 'the key carrying this symbol'. Special keys referred to in this Manual are:

KEY	MEANING
[RET]	[Return key]; [Enter] on Spectrum
[ESC]	[Escape key]; [←] on C64; [Caps Shift + 1] on Spectrum
[CRSRR]	[Cursor Right]; [Caps Shift + 8] on Spectrum
[CRSRL]	[Cursor Left]; [Caps Shift + 5] on Spectrum
[↑]	[Cursor Up]; [Caps Shift + 7] on Spectrum
[↓]	[Cursor Down]; [Caps Shift + 6] on Spectrum
[CTRL]	[Control Key]; [Caps Shift] on Spectrum
[SP]	[Space Bar]

Hands on — Spectrum

The <ASSEMBLE> prompt on the bottom line of the screen display is telling you that the assembler is waiting for you to do something. We want to enter our first program, so tell Champ you want to EDIT by pressing CAPS SHIFT and 1 simultaneously. The prompt changes to <EDIT> and shows a flashing underscore at the cursor position. The first thing you need to put in is a comment to say what the program does, so type a semi-colon, together with the title of the program and any other information you think might be useful. Press ENTER before you reach the right-hand side of the screen. The line you have just typed will move up one place and the cursor will start flashing at the beginning of the new line. The prompt will now show <INSERT>, because you are now inserting new information into the assembler. If you make a mistake before pressing ENTER, then use the cursor keys (which operate as normal) to correct your mistake, and just type over any mis-spelt words. If you pressed ENTER before noticing your error, don't worry; you can correct it in a minute.

As you can see, you can also use blank comment lines to space your program listing out to make it more readable.

When you have finished typing the introductory comment, and the cursor is at the beginning of a new line, try pressing ENTER once more. You will find that you go back to <EDIT> mode. In <EDIT> mode, you can use the cursor keys to scroll the listing up and

```
;               CHAMP EXAMPLE PROGRAM
;               6502 VERSION
;
;               ORG $CØØØ
;
;               VARIABLES
;
I               DB    Ø
J               DB    Ø
;
;               PROGRAM
;
; 1Ø FOR I = 1ØØ TO 1 STEP-1
                LDX   #$64
NEXTI           STX   I
; 2Ø FOR J = 255 TO 1 STEP-1
                LDX   #$FF
NEXTJ           STX   J
; 3Ø NEXT J
                LDX   J
                DEX
                BNE   NEXTJ
; 4Ø NEXT I
                LDX   I
                DEX
                BNE   NEXTI
; 5Ø RETURN: REM TO BASIC OR CHAMP
                RTS
```

down and move the cursor through any line you may want to change. Correct any mistakes you may have made (but don't press ENTER) and move the cursor back down to the bottom of the text. Now press ENTER again and you should once more have the cursor on a blank line with text above it and nothing below. If not, use ENTER to toggle between <EDIT> and <INSERT>, and use the cursor keys to get you to the correct position at the bottom of the text.

If you now type a space without a semi-colon, the cursor will skip to the second coloured field (column), because Champ 'knows' that if you type a space here, you don't want anything in the first field. Now type in ORG followed by a space. Once again, when Champ gets the space, it knows that it has to skip to the next field. So you can complete the ORG instruction by typing $CØØØ followed by ENTER. All instructions except comments are typed into the assembler in this way; when you've finished using a particular field (or don't want to use it at all), use the space bar to move to the next field or ENTER to move to the next line.

If you want to type a line containing a label, then start in the LABEL field (the leftmost one), type your label followed by a space, and continue with instruction and operand fields.

Some typing errors will be recognised by Champ when you press ENTER and will cause an error message to be displayed. Possible errors at this point are LABEL, INSTRUCTION or OPERAND errors. These correspond respectively to the three coloured fields in the assembler display, so if you get an error message you should find your mistake in the field referred to. Use the cursor keys to go back and correct the mistake when you've found it.

When you've typed the listing in, press ENTER one last time to return to <EDIT> mode, then use the cursor keys to run through the listing to ensure that it looks like the example. When you are satisfied, press CAPS SHIFT and 1 to return to <ASSEMBLE> mode and SAVE your text, using the S command. (This can be loaded back at any time using the J command.) This is a good habit, as it is easy to lose what you are doing when using machine code since you can't use SHIFT/BREAK to stop a runaway program.

Having SAVEd your listing, you can now assemble it. Type A (for assemble), and Champ will display ASSEMBLE=> on the prompt line. Type in 3 as your assembly option — this tells the assembler what sort of listing you want and is explained more fully elsewhere in the manual. Press ENTER at this point and assembly should commence.

If all is well, Champ will print a version of your listing with some extra numbers on the left-hand side. The leftmost column of numbers shows the addresses to which each instruction has been assembled. They may look a little odd because they are expressed in hexadecimal notation. Notice that the addresses do not increase after comment lines: this is because comments do not produce any machine code. This is reflected in the second two columns of numbers, which contain the machine code values loaded into memory. You can see that comment lines once again produce no machine code.

After the listing you will also see a table of all the labels you used: this is called the symbol table and Champ produces it for your convenience. It enables you to find the parts of the program you want quickly.

Note that addresses of variables and jump labels are held in the symbol table in the same way; this is because the microprocessor holds them in a similar way in its internal registers.

Having successfully assembled our example program, we want to examine the program as it is in memory. Enter the monitor part of Champ by pressing M. The screen should now display the <DEBUG> prompt.

The start address of your program is not $CØØØ, but $CØØ2, because of the space that you reserved for two variables. So type Q (for disassemble), followed by CØØ2 (you don't need a $ sign in <DEBUG> mode).

When you press ENTER, you will see a listing similar to the code you entered, but without the comments, variables, labels and pseudo-ops. You will remember that these produce no machine code. (If you now press any key, the Disassemble feature will display a further block of memory; it will do this repeatedly, until you press CAPS SHIFT/1. To return to the disassembly, you will then need to type QCØØ2.)

If this listing doesn't look like the machine code you typed into the assembler, then type CAPS SHIFT/1, return to <ASSEMBLE> mode by typing A ENTER and reassemble the program, checking that you use the correct option and that when you finish the assembler says that it found no errors.

And now for the moment of truth! If the listing printed by the disassemble command looks correct,

<EDIT> MODE COMMANDS

In <EDIT> mode, source text is displayed with the cursor on the Edit Line, and <EDIT> on the Command Line. Text on the Edit Line can be overwritten or deleted (using [DEL] or [SP]). [RET] causes the Edit Line contents to be checked for syntax and format. An error message will appear if the line is faulty, and the text will remain on the Edit Line. If the line is acceptable, it will be entered into the source text, and mode will change from <EDIT> to <INSERT>. [RET] toggles these two modes, while [ESC] toggles <EDIT> and <ASSEMBLE> modes.

The following keys can be used to move the source text on the screen, assuming the text on the Edit Line is correct. If a line is edited, and if the edited text is valid, then any of the following keys has the effect of entering the new line into the source text without changing the mode.

KEY	EFFECT
[↑]	Moves one line up the text.
[↓]	Moves one line down the text.
[CTRL]+[U]	Moves the screen text up one page.
[CTRL]+[D]	Moves the screen text down one page.
[CTRL]+[T]	Moves to the top of source text.
[CTRL]+[B]	Moves to the bottom of source text.
[CTRL]+[Z]	Deletes the Edit Line contents.
[ESC]	Enters <ASSEMBLE> mode.
[RET]	Enters <INSERT> mode.

N.B. The text movement keys have the same effects when used in <ASSEMBLE> mode, but they then do not require [CTRL] to be pressed. Thus [U] in <ASSEMBLE> mode moves the screen text up one page.

<INSERT> MODE COMMANDS

It is in this mode that you actually type your Assembly language program into the Assembler. The Command Line shows <INSERT>, and a flashing cursor appears on the Edit Line. The Edit Line (and the whole screen) is divided into three coloured columns, corresponding to the Label, Instruction, and Operand Fields of an Assembly language program:

Label Field
A label is any alphanumeric string of up to six characters. There must be a letter in the first position of the Field. A label does not require a colon (or any other character) as delimiter.

Instruction Field
Instructions are Assembly language mnemonics as in MOS Tech 6502 and Zilog Z80 specifications. They may be two, three, or four letters long, starting in the first position of the Field.

Operand Field
Operands may be hex constants (which must be preceded by $), labels, symbols, or expressions comprising two operands separated by + or –. Decimal, octal, and binary constants are not permitted. Operand formats for the various addressing modes are as specified by MOS Tech and Zilog.

Text entry in <INSERT> is subject to Field Formatting: This means it is impossible for you to type a seven-character label, or a five-character instruction. Typing an extra character, or hitting [SPACE], causes the cursor to skip to the first position of the next Field.

The [CRSRR], [CRSRL], and [DEL] keys act as normal in <INSERT> mode — subject to Field Formatting — but the delete key acts on the cursor character rather than on the character to the left of the cursor.

When you hit [RET] in <INSERT> mode, the contents of the Edit Line are checked for syntax and format; if an error is found, then a message appears on the Error Line. If no error is found, then the contents of the Edit Line enter the source text, and the Edit Line is cleared for the entry of a new line. Hitting [RET] when the Edit Line is blank toggles between <EDIT> mode and <INSERT> mode.

<DEBUG> MODE COMMANDS

This mode combines the following functions:
Memory Monitor — allows you to inspect and alter the contents of memory.
Hex Disassembler — allows you to interpret the contents of memory as machine code to be converted back into Assembly language.
Debugger — allows you to execute machine code programs and trap errors.

<DEBUG> is a command mode, but the Command Line/Edit Line/Field Format display of the other modes is not used: the screen is a blank page showing only the prompt and a cursor. In this mode all constants are hex constants without the '$' prefix, although the 'H' command supports decimal constants.

ABBREVIATIONS	
addr	any hex address
saddr	start address of a block of memory
faddr	finish address of a block of memory (=1+ address of last byte of block)
daddr	destination address in hex
hx	a hex value (hx<= FF)
regname	CPU register name (see below)
expr	any arithmetic expression in one or two operands; operands may be decimal constants, '$' – prefixed hex constants, or legal symbols; operators are '+' or '–'
bystr	a string of hex byte values separated by spaces
chstr	a string of characters (exactly as it appears, no separators)

COMMAND	EFFECT
@ addr	Memory from the given address onwards is displayed one byte at a time, in hex and ASCII equivalent. Hit [RET] to advance to next byte, hit [ESC] to return to command level, or type a hex constant to replace the existing content of the byte
A	Return to <ASSEMBLE> mode
D addr	Memory from the given address onwards is displayed in screen pages; hit any key to continue, or [ESC] to return to command level
F saddr faddr hx	Every byte between saddr and faddr is filled with hx
M daddr saddr faddr	The block of memory between saddr and faddr is copied to the block starting at daddr
Q addr	Memory from addr onwards is disassembled; hit [RET] to continue, and [ESC] to return to command level
G addr	The code starting at addr is executed (returnable)
C addr	Execute from addr (non-returnable)
Bn=addr	A breakpoint number n, (between 1 and 8) is set at addr, to cause a break in execution of any program which accesses the contents of addr as an instruction; press [C] [RET] to continue from breakpoint
En	Eliminates breakpoint n
T	Displays the addresses of all the breakpoints
R regname	Displays the contents of a CPU register and accepts a new value (similar to the function of '@' above)
J addr	Executes the code from addr onwards, one instruction at a time, giving a full register display. Hit [J] to continue, [ESC] to return to the command level
H expr	Displays the decimal, hex, and binary value of expr
S bystr	Searches the memory from $0000 onwards for every occurrence of bystr. The word 'searching' is displayed while the program is searching, and the address is displayed when bystr is found. Hit [RET] to continue the search, or [ESC] to return to command level
Nchstr	As 'S' above
W	Load, Save, and Verify machine code to tape; see BASIC panel

CPU Register Na
Abbreviations

6502

A = Accumulator; X, Y = X, Y registers; P= Status register; SP = Stack Pointer

A = Ac
Status r
H . . . E r
Pointer; I

<ASSEMBLE> MO

COMMAND FORMATS

Find =>string [RET]
Searches the Assembly language program from the start of the program for the first occurrence of the given string.

Next =>string [RET]
Searches the Assembly language program for the next occurrence of the given string. The search begins from the end of the program line currently on the Edit Line.

Find =>[RET] and
Next => [RET]
As above, but this searches for the string defined in last 'F' or 'N' command. While a search is preceeding, the message 'searching' appears on the Error Line. If the search is successful, the line containing the string being searched for appears on the Edit Line. If the search is unsuccessful, the last line of the program appears on the Edit Line.

Load => Save => Verify =>
These must all be followed by a filename; double quotes are not needed, but the filename must be legal for the user's machine.

Print =>expression [RET]
This prints the hex value of the given expression on the Error Line. eg.
Print =>$F8-$C1 **$37**
Symbols already defined in source text can be used in expressions; but only one Operator (+ or –) is allowed per expression.

Quit =>[Y]
This quits Champ and returns control to the BASIC system only if [Y] follows the prompt;
any other response aborts the command.

[M]
Enter <DEBUG> mode. Return from there to <ASSEMBLE> mode by pressing [A] [RET].

[ESC]
Toggle <EDIT> and <ASSEMBLE> modes.

Assemble =>(option number) [RET]
This assembles the source text in one of a variety of ways, depending upon which numerical option is chosen:

COMM

KEY	PR
[F]	Fir
[N]	Ne
[L]	Lo
[W]	Sa
[V]	Ve
[P]	Pri
[Q]	Qu
[M]	
[ESC]	
[A]	As =

To abort
do not er
just hit
prompt.

Spect

KEY	
[J]	
[S]	
[sym.sh	
[R]	

= X, Y
er; SP
guage
f the
rence
guage
rence
earch
f the
ne Edit
es for
or 'N'
ch is
ssage
Error
essful,
string
rs on
rch is
of the
t Line.
=>
d by a
re not
ust be
e.
of the
Error
7
ed in
ed in
one
ed per
eturns
n only
s the
Return
BLE>
T].
and
[RET]
text in
ways,
nerical

Register Name Abbreviations

Z80

A = Accumulator; F = Flag/Status register; H,L,B,C,D,E = H . . . E registers; SP = Stack Pointer; IX, IY = IX, IY registers.

MBLE> MODE COMMANDS

KEY	PROMPT	FUNCTION
[F]	Find =>	Find a string
[N]	Next =>	Find a string
[L]	Load =>	Load a source file
[W]	Save =>	Save a source file
[V]	Verify =>	Verify a source file
[P]	Print =>	Print value of expression
[Q]	Quit =>	Quit to BASIC
[M]		Enter <DEBUG> mode
[ESC]		Enter <EDIT> mode
[A]	Assemble =>	Assemble program

To abort any of these commands, do not enter a command operand, just hit [RET] in response to the prompt.

Spectrum variations

KEY	PROMPT
[J]	Load =>
[S]	Save =>;
[sym.shift] + [R]	Verify =>

INSTRUCTION FORMATS

6502

INSTRUCTION		ADDRESSING MODE
LDA	#$D4	Immediate
LDA	$3C	Zero Page (Direct)
LDA	$A290	Absolute (Direct)
LDA	$31FE,X	Absolute Indexed
LDA	$7B,X	Zero Page Indexed
LDA	($2A,X)	Pre-Indexed (Indirect)
LDA	($2A),Y	Post-Indexed (Indirect)
CLC		Implied

Z80

INSTRUCTION		ADDRESSING MODE
LD	A,B	Register (Direct)
LD	A,$9F	Immediate
LD	($ED46),A	Absolute (Direct)
LD	A,(HL)	Register (Indirect)
LD	A,(IY+d)	Indexed (Indirect)
CCF		Implied

ASSEMBLY LANGUAGE FORMATS

PSEUDO-OP-CODES	MEANING
ORG addr	origin: assemble machine code in memory from addr onwards. The program line with ORG on it cannot take a label.
EQU	equate: set the symbol in the Label Field equal to the constant, symbol, or expression in the Operand Field.
DB const/chstr	define byte(s): load this location, and as many following as required, with the value(s) of const or chrstr
DW const/symb	define word: load this location with the lo-byte, and the next location with the hi-byte of the operand
DS const/symb	define storage: add the value of the operand to the location address of this instruction

Abbreviations:

addr	a $-prefixed hex address
const	a $-prefixed hex constant; as an operand of DB, const must be a single-byte value. A string of constants, such as (DB const [SP] const [SP] const . . . etc) is valid
chstr	a string of characters enclosed in single quotes (e.g. 'AB3%9K10')
symb	any valid symbolic operand

LINKING MACHINE CODE AND BASIC

Once you're familiar with both Champ and Assembly language programming, you'll probably want to be able to call special-purpose machine-code routines from BASIC programs, rather than write entire programs in machine code. The easiest way of doing this is:

1) Using Champ, develop the Assembly language routine until it works.
2) From <ASSEMBLE> mode, SAVE the Assembly language routine to tape for future reference.
3) Assemble the routine into memory, choosing an ORG address near the top of User RAM (see your computer User Manual for Memory Map and advice).
4) From <DEBUG> mode, SAVE the block of memory containing your machine code to tape.
5) Quit Champ.
6) Write your BASIC program, starting with the instructions necessary to set the Top of User RAM pointers to an address safely below the ORG address of your routine. Follow those instructions in the program with a LOAD instruction that will load your machine code routine from tape to the location from which it was SAVEd (consult your User Manual).
7) Whenever you need to execute the machine-code routine in the BASIC program, use a CALL, SYS, or USR instruction with your routine's ORG address.
8) Save the BASIC program as usual.

If you exit from Champ to BASIC, and type LIST, you should see an example of this technique at work: When you LOAD Champ, you load only the short BASIC loader program which you see; when this is executed, it LOADS Champ itself as a machine code file into memory, then calls it as a machine code routine.

CHAMP ERROR MESSAGES

Error messages appear on the Error Line in all modes except <DEBUG>, which prints 'ERROR' at the current cursor position.

MESSAGE	MEANING
LABEL ERROR	A syntax or format error in the Label Field.
INSTRUCTION ERROR	A syntax or format error in the Instruction Field.
OPERAND ERROR	A syntax or format error in the Operand Field.
UNDEFINED LABEL	The Label or Symbol displayed on the Edit Line has not been assigned an address or a value.
JUMP OUT OF RANGE	The relative jump in the instruction on the Edit Line requires a displacement of more than 127 bytes forward or 128 bytes backward.
OVERFLOW	Assembling the instruction on the Edit Line into memory would overwrite CHAMP itself, or some protected memory, or would be out of range.
ERROR	The operand of a <DEBUG> command contains illegal symbols, or is too large a quantity, or is a bad address, etc.

ASSEMBLY OPTIONS

OPTION NUMBER	0	1	2	3	4	5	6	7
Display full list on screen	N	Y	N	Y	N	Y	N	Y
Load m/code into memory	N	N	Y	Y	N	N	Y	Y
Copy screen to printer	N	N	N	N	Y	Y	Y	Y
Verify labels, symbols & syntax	Y	Y	Y	Y	Y	Y	Y	Y
Display symbol table on screen	Y	Y	Y	Y	Y	Y	Y	Y

Y = This facility enabled
N = This facility disabled

E.g. **Assemble =>2 [RET]** causes the source text to be assembled with error-checking, and the resulting machine code to be loaded into memory as directed by the ORG pseudo-op-code. The symbol table is displayed on the screen, but no assembly listing appears on the screen, and there is no output to the printer.
Any option number can be preceded by 1, which gives a double-line display if the screen list facility is enabled.

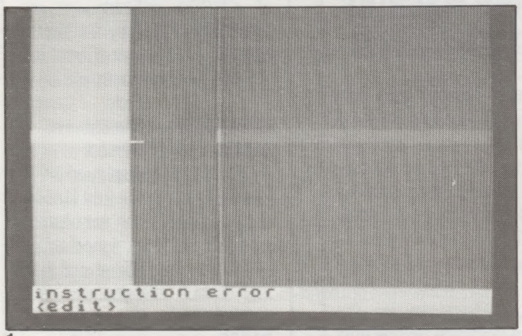
If an error is found during assembly, a message will appear on the Error Line, assembly will cease, and the screen will display the source text with the faulty instruction appearing on the Edit Line.

you can execute it in <DEBUG> mode by typing GCØØ2 ENTER. (In other words, G followed by the starting address.) If all is well, the <DEBUG> prompt will return almost immediately, telling you that your program has been executed.

You might like to enter and run the BASIC program in order to appreciate the difference in speed between the two languages. You could even modify the programs to put an extra loop around the outside of the two loops already present and use a stop-watch to calculate exactly how much faster machine code is. Be prepared to wait a long time for the BASIC!

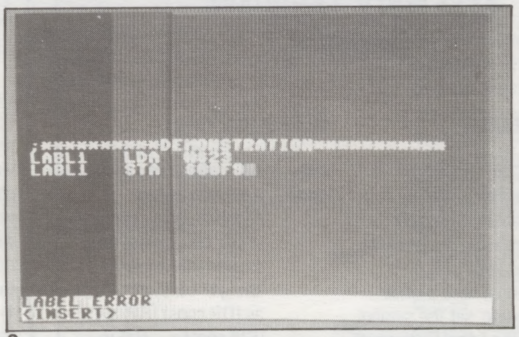
DISASSEMBLY LISTING — Z80

```
CØØ2    3E64      LD     A,$64
CØØ4    32ØØCØ    LD     ($CØØØ),A
CØØ7    3EFF      LD     A,$FF
CØØ9    32Ø1CØ    LD     ($CØØ1),A
CØØC    3AØ1CØ    LD     A,($CØØ1)
CØØF    3D        DEC    A
CØ1Ø    2ØF7      JR     NZ, $CØØ9
CØ12    3AØØCØ    LD     A,($CØØØ)
CØ15    3D        DEC    A
CØ16    2ØEC      JR     NZ,$CØØ4
CØ18    C9        RET
```



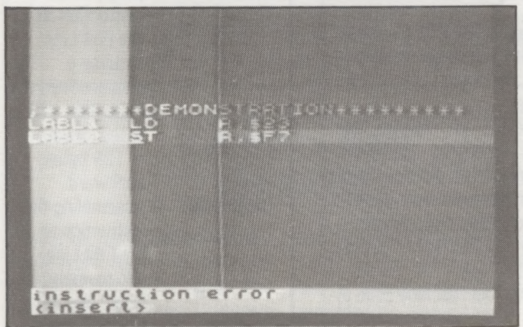
```
instruction error
<edit>
```

1



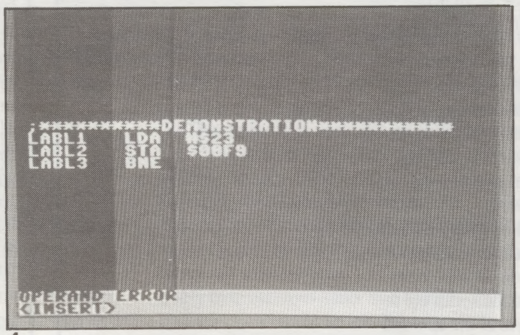
```
;**********DEMONSTRATION**********
LABL1    LDA   #$23
LABL1    STA   $00F9
LABEL ERROR
<INSERT>
```

2



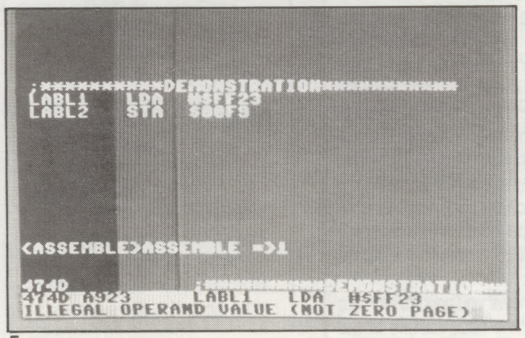
```
;*******DEMONSTRATION**********
LABL1    LD      A,$23
[illegible]    ST      A,$F7
instruction error
<insert>
```

3



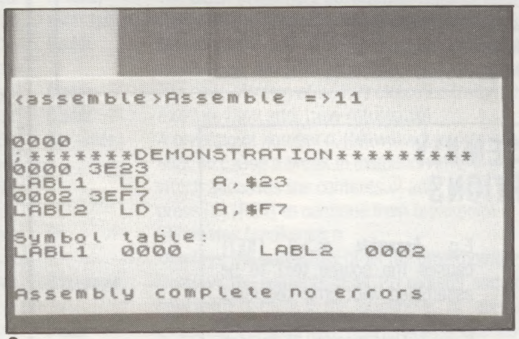
```
;**********DEMONSTRATION**********
LABL1    LDA   #$23
LABL2    STA   $00F9
LABL3    BNE
OPERAND ERROR
<INSERT>
```

4



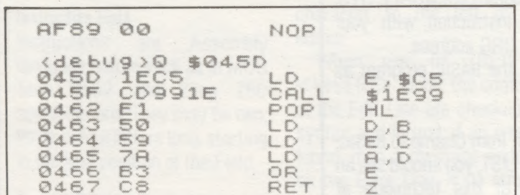
```
;**********DEMONSTRATION**********
LABL1    LDA   #$FF23
LABL2    STA   $00F9

<ASSEMBLE>ASSEMBLE =>1

474D
474D A923       ;**********DEMONSTRATIO
                LABL1    LDA   #$FF23
ILLEGAL OPERAND VALUE (NOT ZERO PAGE)
```

5



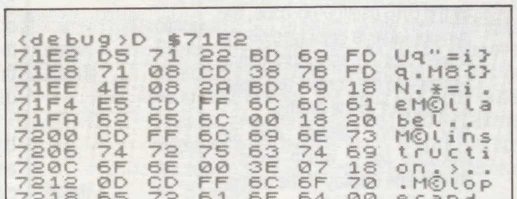
```
<assemble>Assemble =>11

0000
;*******DEMONSTRATION*********
0000 3E23
LABL1  LD      A,$23
0002 3EF7
LABL2  LD      A,$F7

Symbol table:
LABL1  0000       LABL2  0002

Assembly complete no errors
```

6



```
AF89 00          NOP

<debug>Q $045D
045D 1EC5        LD     E,$C5
045F CD991E      CALL   $1E99
0462 E1          POP    HL
0463 50          LD     D,B
0464 59          LD     E,C
0465 7A          LD     A,D
0466 B3          OR     E
0467 C8          RET    Z
0468 1B          DEC    DE
0469 C3B503      JP     $03B5
046C CF          RST    8
046D 0A          LD     A,(BC)
046E 89          ADC    A,C
046F 02          LD     (BC),A
0470 D0          RET    NC
0471 12          LD     (DE),A
0472 86          ADD    A,(HL)
0473 89          ADC    A,C
0474 0A          LD     A,(BC)
0475 97          SUB    A
```

7



```
<debug>D $71E2
71E2 D5 71 22 BD 69 FD Uq"=i}
71E8 71 08 CD 38 7B FD q.M8{}
71EE 4E 08 2A BD 69 18 N.*=i.
71F4 E5 CD FF 6C 6C 61 eM©lla
71FA 62 65 6C 00 18 20 bel.. 
7200 CD FF 6C 69 6E 73 M©lins
7206 74 72 75 63 74 69 tructi
720C 6F 6E 00 3E 07 18 on.>..
7212 0D CD FF 6C 6F 70 .M©lop
7218 65 72 61 6E 64 00 erand.
721E 3E 0D FD 77 10 CD >.}w.M
7224 38 72 4F FD BE 0E 8rO}>.
722A C0 CD CE 6D 01 F6 @MNm.v
7230 FF 09 CD B5 70 0E ©.M5p.
7236 00 C9 CD FF 6C 20 .IM©l 
723C 65 72 72 6F 72 00 error.
7242 21 00 02 11 80 00 !.....
7248 CD B5 03 AF C9 FE M5./I~
724E 02 20 31 CB E0 CD . 1K£M
7254 2F 70 1B 0C 0C FE /p...~
725A 24 28 84 FE 30 38 $(.~08
```

8

Getting out of <EDIT> mode 1
When the Edit Line is blank in <EDIT> mode on the Spectrum, ENTER should toggle <INSERT> mode, but may cause an error message. Type a semi-colon (;), thus turning the line into a comment. ENTER will then toggle <INSERT> mode, and you can delete the entire spurious line from <EDIT> or <ASSEMBLE> mode.

Label error, 2
The second occurrence of LABL1 in this example is detected as a label error or erased.

Instruction error, 3
Champ is here being used on a Spectrum. However, the assembly language instruction ST does not exist in Z80 code, so an instruction error is flagged when the line is entered

Operand error, 4
Here Champ is being used on a Commodore 64. The 6502 assembly language instruction BNE requires an operand, so trying to enter the line without one causes an operand error.

Assembly error, 5
Not all errors are trapped at entry time. Here, the first line of code contains a logical error, which has been caught. Hitting any key will restore <ASSEMBLY> mode.

Completed assembly, 6
The assembly has been successful, as the message shows. Because the option chosen was 11, each line of assembly listing occupies two screen lines; option 1 is logically equivalent, but allows only one screen line per assembly line.

Disassembly function, 7
Q, the disassembly function in <DEBUG> mode, produces a block of lines of disassembly; hit ESCAPE to return to <DEBUG> and then hit any key to disassemble the next block.

Memory display function, 8
D, the memory display function in <DEBUG> mode shows the hex contents (and their ASCII equivalents) of a block of memory; hit ESCAPE or any key to return or continue, respectively. This is the Z80 display, which shows six bytes per line. The 6502 display shows eight.

Hands on — BBC Micro and Commodore 64
The <ASSEMBLE> prompt on the bottom line of the screen display is telling you that the assembler is waiting for you to do something. We want to enter our first program so tell Champ you want to EDIT by pressing ESCAPE on the BBC Micro, or ← on the Commodore 64. The prompt changes to <EDIT> and shows a flashing square at the cursor position. The first thing you need to put in is a comment to say what the program does, so type a semi-colon together with the title of the program and any other information you think might be useful. Press RETURN before you reach the right-hand side of the screen. The line you have just typed will move up one place and the cursor will start flashing at the beginning of the new line. The prompt will now show <INSERT>, because you are now inserting new information into the assembler. If you make a mistake before pressing RETURN, then use the cursor keys (which operate as normal) to correct your mistake, and just type over any mis-spelt words. If you pressed RETURN before noticing your error, don't worry; you can correct it in a minute.

As you can see, you can also use blank comment lines to space your program listing out to make it more readable.

```
;               CHAMP EXAMPLE PROGRAM
;               6502 VERSION
;
                ORG $CØØØ
;
;               VARIABLES
;
I               DB      Ø
J               DB      Ø
;
;               PROGRAM
;
; 1Ø FOR I = 1ØØ TO 1 STEP-1
                LDX     #$64
NEXTI           STX     I
; 2Ø FOR J = 255 TO 1 STEP-1
                LDX     #$FF
NEXTJ           STX     J
; 3Ø NEXT J
                LDX     J
                DEX
                BNE     NEXT J
; 4Ø NEXT I
                LDX     I
                DEX
                BNE     NEXT I
; 5Ø RETURN: REM TO BASIC OR CHAMP
                RTS
```

When you have finished typing the introductory comment, and the cursor is at the beginning of a new line, try typing RETURN once more. You will find that you go back to <EDIT> mode. In <EDIT> mode, you can use the cursor keys to scroll the listing up and down and move the cursor through any line you may want to change. Correct any mistakes you may have made (but don't press RETURN) and move the cursor back down to the bottom of the text. Now press RETURN again and you should once more have the cursor on a blank line with text above it and nothing below. If not, use RETURN to toggle between <EDIT> and <INSERT>, and use the cursor keys to get you to the correct position at the bottom of the text.

If you type a space without a semi-colon, the cursor will skip to the second coloured field (column), because Champ knows that if you type a space here, you don't want anything in the first field. Now type in ORG followed by a space. Once again, when Champ gets the space, it knows that it has to skip to the next field. So you can now complete the ORG instruction by typing $CØØØ followed by RETURN. All instructions except comments are typed into the assembler in this way; when you've finished using a particular field (or don't want to use it at all), use the space bar to move to the next field or RETURN to move to the next line.

If you want to type a line containing a label, then start in the LABEL field (the leftmost one), type your label followed by a space, and continue with instruction and operand fields.

Some typing errors will be recognised by Champ when you press RETURN and will cause an error message to be displayed. Possible errors at this point are LABEL, INSTRUCTION or OPERAND errors. These correspond respectively to the three coloured fields in the assembler display, so if you get an error message you should find your mistake in the field referred to in the error message. Use the cursor keys to go back and correct the mistake when you've found it.

When you've typed the listing in, press RETURN one last time to return to <EDIT> mode, then use the cursor keys to run through the listing to ensure that it looks like the example. When you are satisfied, press the ESCAPE key to return to <ASSEMBLE> mode and SAVE your text, by using the W command. (This can be loaded back at any time, using the L command.) This is a good habit, as it is easy to lose what you are doing when using machine code, since you can't use STOP or Control/C to stop a runaway program.

Having SAVEd your listing, you can now assemble it. Type A (for assemble), and CHAMP will display ASSEMBLE=> on the prompt line. Type in 3 as your assembly option — this tells the assembler what sort of listing you want and is explained more fully elsewhere in the manual. Press RETURN at this point and assembly should commence.

If all is well, Champ will print a version of your listing with some extra numbers on the left-hand side. The leftmost column of numbers shows the addresses to which each instruction has been assembled. They may look a little odd because they are expressed in hexadecimal notation. Notice that the addresses do not increase after comment lines: this is because comments do not produce any machine code. This is reflected in the second two columns of numbers, which contain the machine code values loaded into memory. You can see that comment lines once again produce no machine code.

After the listing you will also see a table of all the labels you used: this is called the symbol table and Champ produces it for your convenience. It enables you to find the parts of the program you want quickly.

Note that addresses of variables and jump labels are held in the symbol table in the same way; this is because the microprocessor holds them in a similar way in its internal registers.

Having successfully assembled our example program, we want to examine the program as it is in memory. Enter the monitor part of Champ by pressing M and RETURN. The screen should now display the DEBUG prompt.

The start address of your program is not $CØØØ, but $CØØ2, because of the space that you reserved for two variables. So type Q (for disassemble), followed by CØØ2 (you don't need a $ sign in <DEBUG> mode).

When you press RETURN, you will see a listing similar to the code you entered, but without the comments, variables, labels and pseudo-ops. You will remember that these produce no machine code. (If you now press any key, the Disassemble feature will display a further block of memory; it will do this repeatedly, until you press ESCAPE (BBC) or ← (C64). To return to the disassembly you will then need to type QCØØ2.)

If this listing doesn't look like the machine code you typed into the assembler then press ESCAPE, return to <ASSEMBLE> mode by typing A RETURN and reassemble the program, checking that you use

DISASSEMBLY LISTING – 65Ø2

```
CØØ2      A264        LDX     #$64
CØØ4      8EØØCØ      STX     $CØØØ
CØØ7      A2FF        LDX     #$FF
CØØ9      8EØ1CØ      STX     $CØØ1
CØØC      AEØ1CØ      LDX     $CØØ1
CØØF      CA          DEX
CØ1Ø      DØF9        BNE     $CØØ9
CØ12      AEØØCØ      LDX     $CØØØ
CØ15      CA          DEX
CØ16      DØEE        BNE     $CØØ4
CØ18      6Ø          RTS
```

the correct option and that when you finish the assembler says that it found no errors.

And now for the moment of truth! If the listing printed by the disassemble command looks correct, you can execute it in <DEBUG> mode by typing GCØØ2 RETURN. (In other words, G followed by the starting address.) If all is well, the <DEBUG> prompt will return almost immediately, telling you that your program has been executed.

You might like to enter and run the BASIC program in order to appreciate the difference in speed between the two languages. You could even modify the programs to put an extra loop around the outside of the two loops already present and use a stop-watch to calculate exactly how much faster machine code is. Be prepared to wait a long time for the BASIC!

Points to remember

LABELS must start with a letter, and must not be more than six alphanumeric characters long.

INSTRUCTION MNEMONICS must be standard 6502 or Z80: two, three, or four letters long.

OPERANDS must follow standard 6502 or Z80 formats. They can contain arithmetic expressions comprising symbols or hex constants and a '+' or '—' operator, and can fill, but not exceed, the entire operand field.

In <EDIT> mode you can change the text on the Edit Line, and you can move the entire text file up and down on the screen using the following keys (on the Spectrum, replace the control key by CAPS SHIFT).

KEY	EFFECT
[↑]	Moves the Edit Line up one line
[↓]	Moves the Edit Line down one line
[CTRL]+[T]	Moves to the top of the text
[CTRL]+[B]	Moves to the bottom of the text
[CTRL]+[U]	Moves text up one screen page
[CTRL]+[D]	Moves text down one screen page
[CTRL]+[Z]	Deletes the contents of the Edit Line

These keys without [CTRL] have the same effects in <ASSEMBLE> mode, but you cannot delete or otherwise modify your text in that mode.

If your program executes successfully, then the <DEBUG> prompt and cursor will return to the screen. The 'D' command can now be used to display the contents of the memory that the program should affect. If the results are successful, then you might want to SAVE the machine code (called the object code to distinguish it from the assembly language source code) to tape, using the 'W' command in <DEBUG> mode. Having done that, you might like to try altering some of the object code in memory using the '@' command, also in <DEBUG> mode. Once you've started to understand roughly what's going on in Champ, you should simply play around with any and every command or option that meets your eye — you can't damage anything and it's really the only way to learn.

Developed for Home Computer Advanced Course by Personal Software Services, 452 Stoney Stanton Rd, Coventry (0203) 81346.

Software	**— © PSS 1984**
Manual Text	**— © Orbis 1984**
Software Authors	**— D. Ritchie, T. Stoddard**
Manual Authors	**— B. R. Morris, C. D. McCausland**
Editor	**— D. Cohen**