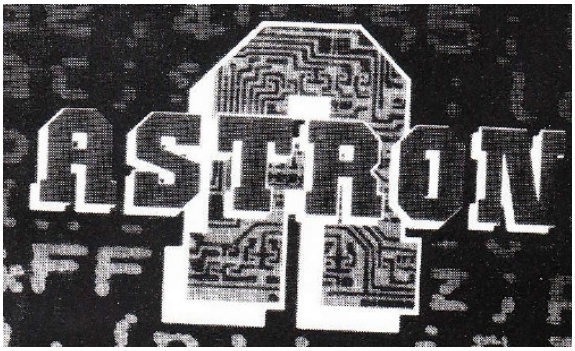

MACHINE CODE ASSEMBER

ASTRON

Astron is a Z80 Assembler with many unique features, user friendly for the beginner yet having the power and versatility required by the advanced programmer. Both versions feature a toolkit utility which, after a short time, you will wonder how you managed without!

LOADING
Load"" The program is in two parts and will auto-run giving a Menu. RUN will always give the Menu and also incidentally set all Border, Paper and Ink colours to "normal". Ramtop will be 55972 (16K 27390) but you can of course CLEAR lower to protect Object Code.

ENTERING SOURCE CODE
Source code as standard Zilog mnemonics is entered into REM statements. Multi statement lines use semi-colons as separators. Numbers can be written as decimal or hex (hex preceded by &). Relative jumps can be + or - but index displacements must always be positive.

LABELS
Labels are substitutes for any numeric value 0-65535. Examination of the sample program will reveal their usefulness in saving tedious calculation and making source code easy to modify and understand. Most labels represent an address and are assigned by writing the label into the code. Other values can be assigned to a label using 'equate', e.g. equ 16384 DFILE. Throughout the program any reference to DFILE will be substituted with 16384 e.g. 1d h1, DFILE.

Apart from the first character, which must be a capital letter, labels can be a mix of characters and any length. Small and capital letters are not distinguished between e.g. LABEL and Label are identical but LABELS LABS LAB1 LAB3 are all unique.

ASTRON used two types of label, temporary and permanent. Those described so far are temporary and at each assembly are erased and reassigned to their old or to new values. Permanent labels of which there are 26 will always consist of a per cent sign followed by a capital letter e.g. %A, %B and are designed specifically to deal with amalgamating sections of code assembled separately (see below).

PSEUDO OP-CODES
These non Zilog codes are designed as programming aids.

"Start" and "end" must be placed in REM statements as the first and last codes of your program so that the assembler knows its operational limits.

"Org" (shortened ORIGIN) must follow "Start" on the next line and be followed by the first address of the object code. "Org" can be used more than once in a program to assemble code into different parts of memory.

Sometimes you may wish your code to occupy a space in memory which during assembly is already used e.g. by your source code or ASTRON. If for instance you want your code to run at 24000 you have to do it in two stages using a special org instruction consisting of the assembly address (say 40000) followed by a comma and then the RUN address i.e. Org 40000,24000. Code will now be assembled at 40000 but will crash if you run it there. It must then be transferred to 24000 in order to run correctly either by saving to cassette and loading to 24000, or using a few lines of Basic transfer program.

"datab" (data byte) is used to insert one or more bytes (0-255) into the code. Leave spaces only between the numbers e.g. Datab 80 81 36 0 254.

"dataw" (data word) is used exactly as Datab but each two-byte number (0-65535) takes two bytes of code e.g. dataw 0 4 256 3473 65535 will code as 0,0,4,0,0,1,145,13,255,255.

"defm" (define message) can be followed by a string of any length, which will be converted into its ASC11 code, e.g. Defm Words will code as 87,111,114,100,115. Colour control characters may be included.

"defs" (define space) will reserve the number of bytes specified by skipping over them during assembly, and is useful for reserving space for program generated data.

All data input can, of course, be labelled e.g. M1; defm. This is a message; SPACE; defs 20.

COMMENTS
Any string preceded by an apostrophe (CHR$ 39) is a comment for in-program documentation and will be listed but otherwise ignored by the assembler. They can be any length and contain colour control characters but must not, for obvious reasons, include a semi-colon except to terminate.

ASSEMBLY
To assemble your program RUN for Menu and press A. Assembly is in two passes. The screen will be blank during the first pass and a display of addresses, codes and your programs will be given on the second pass.

Error messages will be given for incorrect syntax, oversize relative jumps, duplications of labels etc., during either pass and assembly will stop.

During the second pass a number of options are available, obtained by pressing the following keys:
Q - will switch off the screen display increasing assembly speed appreciably.
W - restores the screen display.
I - will give continuous scrolling.

U - will restore normal scroll.
P - sends the output to the printer.
O - restores screen display from the printer.
Assembly may be stopped at any time with the SPACE key.

SHORTAGE OF SPACE

If you run short of space the following remedies are available:

(1) If a small space is required you could:
   (a) Delete comments.
   (b) Use the toolkit to remove spaces in all 1d inc dec add sub call instructions. They will still assemble correctly.
   (c) Overwrite the Editor/Toolkit code (48K only) but, of course, don't try to use it!

If a much larges space is required you will have to write your program in two or more sections. The problem is that any reference made to addresses in Part 1, must be known to the assembler while assembling Part 2.

PROCEED AS FOLLOWS

1. While writing Part 1 use P/Labels for any addresses or subroutines needed by Part 2.
2. Use the Menu to save source code mnemonics etc. (Option 5). Also, of course, save your assembled object code.
3. (48K) Use the Toolkit to delete all Part 1 source code. You're now ready to write Part 2 of your program. P/Labels are still safe in Line 1.
3. (16K) In addition to saving your program, save the P/Labels separately (Option P). Now load the first program on tape (the Menu Basic) which will destroy Part 1. Then load the P/Labels (LOAD "" CODE) and you're ready to write Part 2.

If while writing Part 2 you wish to back to Part 1 to make alterations this can easily be done (save Part 2 first!) provided you save P/Labels and then transfer them into Part 2.

If you run out of P/Labels you should note the addresses of any cross reference labels in Part 1 and use "equate" in Part 2 so that the assembler is informed of the addresses of such labels during its Part 2 assembly. e.g. equ 25763 RAND placed once anywhere in Part 2 will enable use of the label RAND to refer to the subroutine RAND in Part 1.

EXAMPLE PROGRAM

```
100 REM start
102 REM org 40000
104 REM 'A silly program
106 REM 'labelled %S
108 REM LENG ; nop
110 REM equ &4000 DFILE
112 REM MESSAGE; datab 13 13 13; defm Hello there
114 REM %S ; call LENGTH
116 REM ld hl , DFILE; ld a , (LENG) , ld b , a
118 REM LOOP ; push bc
120 REM ld a , (hl) ; call PRINT
122 REM 'PRINT is an established subroutine
124 REM pop bc ; d inz LOOP
126 REM ret
```

```
128 REM LENGTH; ld a , LEN; ld (LENG) , a ; ret
130 REM equ 38478 PRINT
132 REM equ 14 LEN
134 REM end
```

16K VERSION ONLY

There is insufficient room in memory for the Assembler and Toolkit together. Only the Basic, followed by the Assembler will load. The Toolkit, which follows next on the tape is designed to directly overwrite and replace the Assembler if and when needed, using Option L of the Menu. After using Toolkit, the Assembler can be reloaded in the same way. Even so, space is at a premium and the following hints will aid its efficient use.

1. Multi statement lines are more economical than a statement per line.
2. Use P/Labels or keep your labels as brief as possible.
3. Use the print buffer for object code.
4. Omit spaces in source code but be careful of ambiguity. (cpl is not cp l)!

48K VERSION ONLY

These additional options are provided on the Menu:

8 - 8-Bit Peek lists addresses and their contents from an input address.

6 - 16-Bit Peek gives the 16-Bit contents of two consecutive addresses.

P - 16-Bit Poke enables you to poke a word into two consecutive addresses.

V - View labels gives an 'independent' list of normal labels and their addresses. Lines of question marks or 'garbage' which follow gives an indication of how much space is left for labels. This option is especially useful when assembling in mode Q (no screen output).

TOOLKIT

The more program you write, especially with the 48K version, the more useful you will find the Toolkit in organising your program. H for HELP is the key for both the Basic Menu selection and the Toolkit Menu. The space key or STOP as input will take you out of Toolkit.

KEY

A ALTER - searches for and then replaces a string of characters. You specify both the original and the new string, and the range of line numbers where the changes are to be made. See also VERIFY below.

B BYTES - tells you how much free memory space is available for your program.

C COPY - duplicates lines of your program. You specify the range of lines to be copied and where they are to be placed.

D DELETE - deletes a block of lines from your program. You specify the range of lines to be deleted.

F FIND - finds a string of characters. All lines containing the specified string are listed to the screen.

H HELP - gives a menu of the options available along with a brief description of the Function.

M MOVE - ask for COPY, except that the original lines are DELETED thus effectively moving lines from one part of your program to another.

R RENUMBER - renumbers lines of program from and to input line numbers. Note that this option is not suitable for Basic as GOTO's etc are not catered for.

S SEQUENCE	- gives auto line number complete with REMs specifically designed for source code entry. To exit from this mode delete the line number and REM and then input STOP.
V VERIFY	- When ON, will list all lines containing the string replaced by ALTER. When OFF, no listing is produced, just the total number of changes made in confirmation of successful amendments or otherwise.

NOTES