ULTRAVIOLET

Ultraviolet is a 2-pass assembler written specially for the Sinclair ZX Spectrum. It is supplied in two different versions for use in either 16K or 48K machines. Please make sure that you are using the correct version for your machine. The 48K version of Ultraviolet can be used at the same time as the 48K version of Infrared, the complementary disassembler program from ACS Software. In this case, please note that Ultraviolet should be loaded before Infrared.

To load Ultraviolet, simply use LOAD"". (The 16K version loads in three sections, so be patient.) To prepare the assembler for use, press any key to clear the screen and low memory. The assembler will remain in high memory ready for use. It is not possible here to provide an account of how to program in Z80 machine code. If you want to learn, we would suggest that you acquire a copy of "Understanding Your ZX81 ROM" by Dr. Ian Logan, published by Melbourne House Publishers Ltd.

Writing machine code programs with Ultraviolet

To use Ultraviolet, first reserve space for the machine code that it will produce. This is probably best done by putting a dummy REM statement at the beginning of the program. The REM statement should contain enough space to take the machine code. In this case, the first address available to accept machine code will normally be 23760. Normal Z80 mnemonics are recognised written in REM statements in *lower case* exactly as they appear in the back of the Sinclair ZX Spectrum Basic programming book. The exception is that ret c, ret p, ret m and ret z must always be followed by a comma. When using relative jumps, the direction must be indicated with a sign e.g. jr -3 or jr nc,+7. Note that all numbers are entered as decimal, so use for example rst 16 rather than rst 10h. (True hex enthusiasts may frown at this, but we believe that if humans want to use decimal, then they should not be forced to use hexadecimal simply to make life easier for a computer. Don't worry though, the assembler will list the hex code for you!) The mnemonics should be preceded by "REM go" and terminated by "REM finish". The first instruction should be the pseudo-instruction org (see below). Apart from the go and finish statements which should be on separate lines, multiple statements per line are allowed provided that they are separated by semicolons.

Remember when writing the mnemonics that the positions of the commas, spaces and brackets are very important. For example, out (255),a will be recognized correctly, but out(255),a or out(255)a will not. (In this respect note that some of the mnemonics listed at the back of the Sinclair ZX81 BASIC book are incorrect so make sure that you are using the Spectrum book.)

The pseudo-instructions

Ultraviolet supports certain instructions which, although not official Z80 instructions can be included in machine code programs to make life a bit easier. They are listed here:

org This instruction tells the assembler where to start assembling code. For example, org 23760 will initiate assembly at the first free byte in your REM statement at the top of your program. For obvious reasons the first instruction in your program must be an org. You can use org as often as you like to make the assembler skip to a new assembly address. Sometimes you might want to write

a program which is designed to operate in the region of memory around address 60000 (or around address 30000 in the 16K Spectrum). This could present a problem because this is where the assembler itself lives. Assembling code here would overwrite the assembler and probably cause a crash. You can get round this problem by using a special form of org. If you were to write org 23760 (60000) this would result in code being assembled at 23760 but written so that it would operate correctly when relocated to 60000. All you need to do now is to use the Spectrum SAVE"" CODE 23760,x and LOAD""CODE 60000,x facility to relocate the code (or better still use ldir). Once again, remember that the spaces and brackets are important.

defb This allows a byte at the current assembly address to be set to a defined value. This can be useful when forcing an error message with rst 8. For example rest 8;defb 1 will give the error message "2 Variable not found".

defw This is similar to defb but sets a word (2 bytes) rather than a single byte. Its argument is therefore a number between 0 and 65535. The number is inserted into memory least significant byte first.

defs This allows a string of ASCII characters to be inserted into memory at the current assembly position. For example the name of this program could be stored with defs Ultraviolet. This can be useful for storing text within a program.

equ This is used for specifying labels and for linking sections of code together (see below).

Labels

Labels can be used to refer to memory locations whose addresses will not be known until assembly is complete. Labels can be any length with the following restrictions: the first character must be a capital letter and the characters ")" and "+" must not be used. For example, "Here", "Any-where" and "Z3/@$(" are valid labels but "this label", "Place(2)" and "Local+1" are not. The labels are simply written into the program like any other instruction. Suppose you had the label "AcS". This can be referred to in the following ways:
ld a,(AcS);ld hl,(AcS); ld hl,AcS;ld de, AcS;call AcS;jp nz,AcS;jr c,AcS etc.

Writing self-modifying code

Suppose that you have the following piece of code:

….;Johnny;ld hl,(23700);….

You might want your program to alter the address from which hl is loaded. In other words you might want to change the two bytes in memory one byte after "Johnny". This can be done with the following code:

….ld hl,30000;ld (Johnny+1),hl;….

When this part of the program is executed, it rewrites the program to give

….;Johnny;ld hl, (30000);….

Note that only +1 and +2 are allowed in this type of label reference.

Explicit reference to labels

Sometimes, particularly if you have a 16K Spectrum, you might want to write a machine code program in several sections and then join them together later. This is fine if you do not try to refer to labels outside the section of code that you are currently writing. If you do, then the label will not be found and an error will occur. You can get round this problem by using equ which tells the

assembler the address of a label. For example equ 23780 Fred will allow the label “Fred” to be used even though the address of “Fred” is not within the section of code being assembled.

A final word on labels. The assembler stores the names and addresses of labels in a table. The space allocated for this table is limited. When it is used up, no more labels will be recognised and reference to further labels will give an error. The number of labels that you can use clearly depends on their length. You can have about 100 5-character labels but fewer longer ones, so be frugal and concise!

Comments

Comments can be included in your programs by preceding them with an exclamation mark. During assembly, the first 32 characters of a comment will appear on the screen or printer. If you want to say more, use several consecutive comments.

Running the assembler

The 48K version of the assembler is run with RANDOMIZE USR 60000 (RANDOMIZE USR 27500 for the 16K version). This statement can be added to the REM statements so that assembly can be simply started with RUN. In this case a typical listing might look like this:

```
 10 REM-------space for machine code-------
 20 REM go
 30 REM org 23760 (54321);equ 30000 Other
 40 REM !Start of first section
 50 REM Start;ld hl,(Other);inc hl;ld a,(hl);cp 240;jr z,-6
 60 REM sub 34;ret z,;ld hl,(Start+1);dec hl;ld (Start+1),hl
 70 REM jr Start
 80 REM !Start of second section
 90 REM org 21000
100 REM defs This is text;ld hl,31000;ld a,(hl);cp 84;ret nz
110 REM rst 8;defs 10
120 REM !End of 2 useless sections
130 REM finish
140 RANDOMIZE USR 60000  or  RANDOMIZE USR 27500
```

When run, the assembly address of each instruction, its hex code(!) and the first 13 characters of the mnemonic will appear on the screen. Each instruction is assembled twice so that the destination addresses of forward jumps can be calculated. Assembly stops when the screen is full. Pressing the space key will abort the assembly, pressing “p” during the second pass will copy the listing to the printer and pressing any other key will continue the assembly.

Errors

If go, finish or org are faulty, then an error will be given before assembly starts. If an error is detected during assembly, then assembly will stop with two error messages. The first is a flashing error message which tells you the line number, statement number and type of the instruction where the error was detected. This should let you locate the offender quite easily. The second error message is one of the Sinclair messages. There are three possible. If you enter a wrong number,

i.e. >255 for a single byte or >65535 for a 2-byte number, the assembler will repeatedly subtract either 256 or 65536 from it until it gets a sensible number that it can use. If it can not get a sensible result (e.g. >129 for a relative jump) then error “B integer out of range” will be given. If you refer to a label which does not exist or if you have filled up the label table, the error given will be “2 Variable not found”. All other errors (e.g. mistyped instructions) will give the error “Q Parameter error”. Note that in certain rare cases the assembler will correct you. For example ld r,8 will be assembled as ld r,a since this is the only instruction of this type in the instruction set. However, do not regard the assembler as infallible and be sure to put the semicolons in the correct places. If you have any problems or need any advice, please contact ACS Software.