Assembly

AS.COM includes a text editor. To open a file, enter:

```
as bios
```

To assemble, press ^KO (control-K O). Press Enter to confirm BIOS.BIN as output filename, and the BIOS will be assembled. AS directly generates a binary output file, no linking is required.

If the assembler encounters an error, it will stop and leave the cursor where the error occurred. Press the space bar to continue.

To save and exit a file, press Escape. For a symbol table, press ^KT. Online help is available through ASHELP.COM. Source code for the assembler and editor is available on request.

Editing source code

The built-in editor is similar to the WordStar and Turbo Pascal editors. Tab stops are optimized for assembly language source. Most normal editing can be done with the usual function keys, you really don't need to remember too many functions.

Some useful commands:

^Backspace	Delete to end of line
^J	Search a word, starting from beginning of file (use ^QF for more options)
^L	Repeat search
^QA	Search and replace
^U	Move display such that cursor line is in the middle. Cursor location will lock if Scroll Lock is pressed.
^Y	Delete line.

File management: This editor can handle a main file (selected by command line, or by ^KM), and an include file at the same time. Both files together must be less than 64KB.

Escape	Save and exit (or return to main file if editing include file)
^KI	Return to previous include file.
^KL	Load include file
^KM	Load main file, or return to main file
^KQ	Abandon changes, return to main file or exit to DOS.
^KS	Save current file

Block commands. Please note that this editor always selects entire lines.

^KA	Select all
^KB	Mark beginning of selection
^KC	Copy selection
^KK	Mark end of selection
^KR	Read file
^KV	Move selection
^KW	Write selection to file
^KY	Delete selection

Keyboard Macros: Function and Alt key combinations can be programmed with macros.

^OL F1	Program macro for F1 - type in key sequence, then press ^Brk to complete. Press F1 to execute macro.

For additional details, please refer to ASHELP.

Assembler syntax

The AS syntax does not conform to the standard MASM syntax. Unlike MASM, AS does not try to second-guess the programmer about whether an operand is a variable or a constant, or what the data size is - what you write is what you get. Immediate operands are preceded by #. If the size of an operation can’t be inferred from a register name, add .b to select byte width. Examples:

```
mov    al,count          ;no .b necessary here
inc.b  count             ;default would be word
cmp.b  bytevar,#123      ;immediate operand
```

Register combinations (used rarely) are written bp_si, bp_di, bx_si, bx_di.

Symbol definitions must be at the beginning of the line, and not followed by a colon. Symbols are not case-sensitive.

```
hex    =     $1234             ;this is an equate, hex constant
dec    =     1234              ;decimal constant
bin    =     %1010             ;binary constant
char   =     “!”               ;character / string constant
pc     =     @                 ;program counter value of current segment
```

Expressions are evaluated strictly from left to right, using unsigned 16-bit operations. Parentheses are not supported.

|	Logical OR
&	Logical AND
+ .	Addition (can also use . for record references: base.offset)
-	Subtraction
/	Division
\	Modulo
!	Logical XOR (exclusive or)
[ ]	Memory reference (only needed for register indirect such as [si], and indirect jumps / calls)
~	Logical NOT

Assembler pseudo operations

AS supports the following pseudo operations:

cseg Switch to code segment / set origin. There is only one code segment.

```
cseg  $100              ;set file origin (default origin is 0)
cseg                    ;switch back to code segment
cseg  $fff0             ;advance to $fff0, fill with $ff’s
```

dseg Switch to data segment. The data segment can be used for unitialized variables.

struct Switch to structure segment. Similar to the data segment. Intended for the definition of data structures and stack frames.

s, sw Allocate bytes, words. In the data and structure segments, the current offset is incremented. In the code segment, the space is filled with zeroes, or an optional value.

```
s     2                 ;allocate two bytes
s     10,$ff            ;10 bytes, $ff fill
```

even Align to even address. If the current address is odd, a NOP is inserted.

even16 Align to paragraph (16 byte) boundary. Inserts $ff bytes.

b,w Emit byte / word constants. Use in the code segment only.

```
b     1,2,3,4
b     “Hello world !”,13,10,”$”
```

i486 Support 80486 instruction set. 486 support is rudimentary, but does include the invd and wbinvd instructions, and the a32 / d32 address size prefixes.

include Include source file. The main and include file must fit into 64KB together. Include files cannot be nested.

```
include pci
```

external Include binary file. Use this to include binary tables such as fonts.

```
external font.bin
```

biosdate Emit today’s date as mm/dd/yy.

bytesum Emit a checksum byte. This is calculated starting from the address given. The sum of all bytes, including the checksum byte, is zero.

```
bytesum start
```

#ifdef Start conditional assembly if symbol is defined

#ifndef Start conditional assembly if symbol is not defined

#else Alternate case (optional)

#endif End conditional assembly

Please note that these commands must always be at the start of the line, they cannot be indented. Conditional assembly can be nested.

```
#ifdef debug             ;if symbol debug exists then...
                         ;...assemble  this section
#else                    ;(else is optional)
                         ;otherwise assemble this section
#endif
```

32 bit code

AS was originally written in 1987, and does not include 32 bit support. However, it is quite easy to write 32 bit code even with a 16 bit assembler. Only a small portion (about 3%) of the code in tinyBIOS is 32 bit.

In virtually all cases, tinyBIOS is executing in a 16 bit segment. That is, the CPU assumes 16 bit data and address size unless an override prefix is used. An exception is the 32 bit PCI API, where a prefix is required to force 16 bit data or address size.

For data, the override prefix is $66. AS provides the prefix d32:

```
d32   mov ax,bx          ;mov eax,ebx
d32   add ax,#$5678      ;add eax,#$12345678
w     $1234              ;(need to add high word separately)
```

For addresses, the override prefix is $67. AS provides the prefix a32. 32 bit addresses are problematic as Intel in their infinite wisdom decided to change the encoding of index registers. Please refer to a x86 instruction format table for details.

```
a32 d32 movs             ;[ds:esi] -> [es:edi], 32 bit transfer
                         ;Note that a32 means that ecx rather than cx
                         ;is used for count.
a32   mov [bx],ax        ;The CPU will read this as mov [edi],ax !
a32   mov ax,$5678       ;mov ax,[$12345678]
```

```
w     $1234             ;(high address)
```