Disasm.asm documentation:

**Functions:**

[address\_to\_hex 2](#_Toc152099261)

[convert\_to\_decimal 2](#_Toc152099262)

[write\_to\_line 2](#_Toc152099263)

[effective\_address 2](#_Toc152099264)

[find\_write\_register 2](#_Toc152099265)

[find\_write\_seg\_register 2](#_Toc152099266)

[end\_line 3](#_Toc152099267)

[read\_bytes 3](#_Toc152099268)

[get\_byte 3](#_Toc152099269)

[write\_to\_buff 3](#_Toc152099270)

[end\_line 3](#_Toc152099271)

[force\_write\_to\_file 3](#_Toc152099272)

double\_byte\_number

Take address and change it to human readable format. Furthermore, write the results to line array.

Used functions by this function:

# double\_byte\_number

# convert\_to\_decimal

Take the binary\_number variable and convert it to decimal text. Furthermore, write the results to line array.

Used functions by this function:

* write\_to\_line

# write\_to\_line

Take pointer (ptr\_) and write its contents to line array until the ‘$’ symbol is reached in the given word. This function also stores the length of the line.

# effective\_address

Adds these symbols to line array: first ‘[‘, then the specific register or register sum. Checks if the mod is not equal to 0. If yes, then it adds offset (“poslinkis”) with ‘+’ to the line. Lastly it adds ‘]’ symbol to the line.

Used functions by this function:

* write\_to\_line
* address\_to\_hex
* convert\_to\_decimal
* add\_plus
* add\_left\_bracket
* add\_right\_bracket

# find\_write\_register

Using the variables w\_, reg\_, mod\_ decodes the register to ASCII symbols. Furthermore, it writes the result to line. Only the register AX, BX, ax, bx, etc. is written to line. Only effective address is written in style [bx+si+1], [si], [05FA], etc. No commas or spaces are added!

Used functions by this function:

* write\_to\_line
* effective\_address

# find\_write\_seg\_register

Using the variable sr\_, decode segment register to ASCII symbols. Furthermore, write the result to line. No commas or spaces are added.

Used functions by this function:

* write\_to\_line

# end\_line

Use to add endl to line before writing it to output buffer.

# read\_bytes

Used to update byte\_ variable and/or next\_byte variable. Set second\_byte\_used to ‘1‘ to update both bytes. Return values: next\_byte\_available is set to ‘0’ if next\_byte is unavailable or ‘1’ if it is available; end\_of\_file\_reached to ‘1’ if end of file was reached or to ‘0’ If the end was not reached.

Used functions by this function:

* get\_byte

# get\_byte

Used to read byte from buffer. If buffer end is reached it reads file again and places new symbols in buffer. Returns new next\_byte

# write\_to\_buff

Takes the text from line buffer and writes it to *write\_buff*. When buffer does not have enough capacity left to save the line, the function outputs *write*\_*buff* content to output file, resets the *write\_buff* and saves the line to *write\_buff*.

# end\_line

Used to end line. The function adds *end of line* to the line and writes it to *write\_buff*.

Used functions by this function:

* write\_to\_buff

# force\_write\_to\_file

Used to end work. The function first ends the line and saves it to *write\_buff*. Then it saves the *write\_buff* to output file and resets *write\_index* to ‘0’.

Used functions by this function:

* end\_line
* write\_to\_file

# number\_to\_hex

Take variable binary\_number and convert it to hex. The binary number is one byte size.

Used functions by this function:

* convert\_half\_byte\_to\_HEX