**Macros**

Writing a macro is another way of ensuring modular programming in assembly language.

* A macro is a sequence of instructions, assigned by a name and could be used anywhere in the program.
* In NASM, macros are defined with **%macro** and **%endmacro** directives.
* The macro begins with the %macro directive and ends with the %endmacro directive.

The Syntax for macro definition −

%macro macro\_name number\_of\_params

<macro body>

%endmacro

Where, *number\_of\_params* specifies the number parameters, *macro\_name*specifies the name of the macro.

The macro is invoked by using the macro name along with the necessary parameters. When you need to use some sequence of instructions many times in a program, you can put those instructions in a macro and use it instead of writing the instructions all the time.

; A macro with two parameters

; Implements the write system call

%macro write\_string 2

mov eax, 4

mov ebx, 1

mov ecx, %1

mov edx, %2

int 80h

%endmacro

section .text

global \_start ;must be declared for using gcc

\_start: ;tell linker entry point

write\_string msg1, len1

write\_string msg2, len2

write\_string msg3, len3

mov eax,1 ;system call number (sys\_exit)

int 0x80 ;call kernel

section .data

msg1 db 'Hello, programmers!',0xA,0xD

len1 equ $ - msg1

msg2 db 'Welcome to the world of,', 0xA,0xD

len2 equ $- msg2

msg3 db 'Linux assembly programming! '

len3 equ $- msg3

Macros are evaluated by NASM macroprocessor, and they do have a complex structure, as any macro instantiation can include other macros to be instantiated. A fine tuning of evaluation order is possible, because NASM provides slightly different versions of macro definition directives, namely

• %define for a deferred substitution. If macro body contains other macros, they will be expanded after the substitution.

• %xdefine performs substitutions when being defined. Then the resulting string will be used in substitutions.

• %assign is like %xdefine, but it also forces the evaluation of arithmetic expressions and throws an error if the computation result is not a number.

The key differences are that

• %define may change its value between instantiations if parts of it are redefined.

• %xdefine has other macros on which it directly depends glued to it after being defined.

• %assign forces evaluation and substitutes values. Where xdefine would have left you with the preprocessor symbol equal to 4+2+3, %assign will compute it and assign value 9 to it. We will use the