# Macro

Macro is a predefined set of instructions. It is useful when the same set of code must be utilized numerous times to save coding time.

Macro must be defined before using. Location of macro should be at the beginning of source file before data and code sections.

There are two key types of macros:
- single-line macros
- multi-line macros.

## Single-Line Macros

Single-line macros are defined using the **%define** directive.

```
%define  mulby4(x)      shl x, 2
```

To invoke this macro, use the macro name with argument

```
mulby4(rax)
```

# Macro

## Multi-Line Macros

Multiple-line macros are defined using the following format:

**%macro** <name> <number of arguments>

; body of macro

**%endmacro**

The arguments can be referred to within the body by %<number> i.e., %1, %2,...
Label within the body must be preceded by %%

## Example:

```
%macro      abs  1
            cmp      %1, 0          ;compare argument with zero
            jge      %%done         ;if argument >= 0 goto done:
            neg      %1             ;else negate the argument
%%done:
%endmacro
```

# Macro

**Example:**

```
qvar    dq          4


mov     eax, -3
abs     eax

abs     qword [qVar]
```

**Macro expansion:**

Assembler contains macro processor that processes macro before assembling the source file. Macros are expanded by substituting macro body and arguments at location where macro name is placed.

# Macro

```
%macro aver 3
        mov eax, 0
        mov ecx, dword [%2] ; length
        mov r12, 0
        lea rbx, [%1]
%%sumLoop:
        add eax, dword [rbx+r12*4] ; get list[n]
        inc r12
        loop %%sumLoop
        cdq
        idiv dword [%2]
        mov dword [%3], eax
%endmacro
section .data
        EXIT_SUCCESS equ 0 ; success code
        SYS_exit equ 60 ; code for terminate
section .data
        list1 dd 4, 5, 2, -3, 1
        len1 dd 5
        ave1 dd 0
        list2 dd 2, 6, 3, -2, 1, 8, 19
        len2 dd 7
        ave2 dd 0
section .text
global _start
_start:
        aver list1, len1, ave1 ; 1st, data set 1
        aver list2, len2, ave2 ; 2nd, data set 2
        mov rax, SYS_exit ; exit
        mov rdi, EXIT_SUCCESS ; success
        syscall
                                        1,1           All
```

# Macro

```
%macro aver 3
        mov eax, 0
        mov ecx, dword [%2] ; length
        mov r12, 0
        lea rbx, [%1]
%%sumLoop:
        add eax, dword [rbx+r12*4] ; get list[n]
        inc r12
        loop %%sumLoop
        cdq
        idiv dword [%2]
        mov dword [%3], eax
%endmacro
section .data
        EXIT_SUCCESS equ 0 ; success code
        SYS_exit equ 60 ; code for terminate
section .data
        list1 dd 4, 5, 2, -3, 1
        len1 dd 5
        ave1 dd 0
        list2 dd 2, 6, 3, -2, 1, 8, 19
        len2 dd 7
        ave2 dd 0
section .text
global _start
_start:
        aver list1, len1, ave1 ; 1st, data set 1
        aver list2, len2, ave2 ; 2nd, data set 2
        mov rax, SYS_exit ; exit
        mov rdi, EXIT_SUCCESS ; success
        syscall
                                1,1           All
```

```
 1                                  %line 14+1 macro.s
 2                                  [section .data]
 3                                   EXIT_SUCCESS equ 0
 4                                   SYS_exit equ 60
 5                                  [section .data]
 6 00000000 040000000500000002-      list1 dd 4, 5, 2, -3, 1
 7 00000000 000000FDFFFFFF0100-
 8 00000000 0000
 9 00000014 05000000                 len1 dd 5
10 00000018 00000000                 ave1 dd 0
11 0000001C 020000000600000003-      list2 dd 2, 6, 3, -2, 1, 8, 19
12 0000001C 000000FEFFFFFF0100-
13 0000001C 000008000000130000-
14 0000001C 00
15 00000038 07000000                 len2 dd 7
16 0000003C 00000000                 ave2 dd 0
17                                  [section .text]
18                                  [global _start]
19                                  _start:
20 00000000 B800000000               mov eax, 0
21                                  %line 27+0 macro.s
22 00000005 8B0C25[00000000]         mov ecx, dword [len1]
23 0000000C 49C7C400000000           mov r12, 0
24 00000013 488D1C25[00000000]       lea rbx, [list1]
25                                  ..@4.sumLoop:
26 0000001B 420304A3                 add eax, dword [rbx+r12*4]
27 0000001F 49FFC4                   inc r12
28 00000022 E2F5                     loop ..@4.sumLoop
29 00000024 99                       cdq
30 00000025 F73C25[00000000]         idiv dword [len1]
31 0000002C 890425[00000000]         mov dword [ave1], eax
32                                  %line 28+1 macro.s
33 00000033 B800000000               mov eax, 0
34                                  %line 28+0 macro.s
35 00000038 8B0C25[00000000]         mov ecx, dword [len2]
36 0000003F 49C7C400000000           mov r12, 0
37 00000046 488D1C25[00000000]       lea rbx, [list2]
38                                  ..@5.sumLoop:
39 0000004E 420304A3                 add eax, dword [rbx+r12*4]
40 00000052 49FFC4                   inc r12
41 00000055 E2F5                     loop ..@5.sumLoop
42 00000057 99                       cdq
43 00000058 F73C25[00000000]         idiv dword [len2]
44 0000005F 890425[00000000]         mov dword [ave2], eax
45                                  %line 29+1 macro.s
46 00000066 48C7C03C000000           mov rax, SYS_exit
47 0000006D 48C7C700000000           mov rdi, EXIT_SUCCESS
48 00000074 0F05                     syscall
(END)
```

# Macro

**Macro vs. Function**

| Functions | Macros |
| --- | --- |
| Locate within .text section | Locate before .data & .text section |
| Invoke by name | Invoke by name |
| Arguments are passed via registers, stack or variable | Arguments are specified |
| Do not expand | Expand every time they are invoked |
| Based on linkage & return | Based on code substitution |