Constantes

```
mov     ax,200          ; decimal
mov     ax,0200         ; still decimal
mov     ax,0200d        ; explicitly decimal
mov     ax,0d200        ; also decimal
mov     ax,0c8h         ; hex
mov     ax,$0c8         ; hex again: the 0 is required
mov     ax,0xc8         ; hex yet again
mov     ax,0hc8         ; still hex
mov     ax,310q         ; octal
mov     ax,310o         ; octal again
mov     ax,0o310        ; octal yet again
mov     ax,0q310        ; octal yet again
mov     ax,11001000b    ; binary
mov     ax,1100_1000b   ; same binary constant
mov     ax,1100_1000y   ; same binary constant once more
mov     ax,0b1100_1000  ; same binary constant yet again
mov     ax,0y1100_1000  ; same binary constant yet again
```

Macros

Uma macro é um sinônimo para um grupo de instruções de uso repetitivo.

Embora assemblers diferentes tenham notações ligeiramente diferentes para definir macros, todos requerem as mesmas partes básicas em uma definição de macro:

1. Um cabeçalho de macro que dê o nome da macro que está sendo definida.
2. O texto que abrange o corpo da macro.
3. Uma pseudoinstrução que marca o final da definição (por exemplo, ENDM).

REF - https://www.nasm.us/doc/nasmdoc4.html#section-4.4.1

MACROS

Item	Chamada de macro	Chamada de procedimento
Quando a chamada é feita?	Durante montagem	Durante execução do programa
O corpo é inserido no programa-objeto em todos os lugares em que a chamada é feita?	Sim	Não
Uma instrução de chamada de procedimento é inserida no programa-objeto e executada mais tarde?	Não	Sim
Deve ser usada uma instrução de retorno após a conclusão da chamada?	Não	Sim
Quantas cópias do corpo aparecem no programa-objeto?	Uma por chamada de macro	Uma

```
1  section .text
2      global _start
3
4  _start:
5
6      %macro soma 2
7      mov rax, %1
8      mov rbx, %2
9      add rax, rbx
10     %endm
11
12
13     soma 100,5
14
15
16     mov rdi,rax
17     mov rax, 60
18     syscall
19
20
```

Single-Line MACROS

%define

```
%define soma(x,y)     y+x
mov     rax, soma(3,5)
```

Single-Line MACROS

%define

%undef - remove a macro

```
%define soma(x,y)    y+x
%undef soma
mov     rax, soma(3,5)
```

STRINGS

%defstr

```
section .text
    global _start

section .data

    msg db 0

_start:

   %defstr hello HELLO
   mov rax, hello
   mov [msg], rax

    mov rax, 1
    mov rdi, 1
    mov rsi, msg
    mov rdx, 5
    syscall
    mov rax, 60
    mov rdi, 0
    syscall

    mov rdi,rax
    mov rax, 60
    syscall
```

STRINGS

%strlen

```
1  section .bss
2  value resb 1
3
4  section .text
5      global _start
6
7  _start:
8     %defstr str1 HELLO
9     %strlen  charcnt str1
10
11     mov rax, charcnt
12     mov [value], rax
13
14     mov rdi, rax
15     mov rax, 60
16     syscall
17
```

STRINGS

%substr

```
1  section .bss
2  value resb 10
3
4  section .text
5      global _start
6
7  _start:
8     %defstr str1 HELLO
9     %substr  mychar str1 1,3
10
11     mov rax, mychar
12     mov [value], rax
13
14     mov rax, 1
15     mov rdi, 1
16     mov rsi, value
17     mov rdx, 10
18     syscall
19
20     mov rdi, 0
21     mov rax, 60
22     syscall
```

Conditional Assembly

Da mesma forma que o pré-processador C, o NASM permite que seções de um arquivo de origem sejam montadas apenas se certas condições forem atendidas.

```
%if<condition>
    ; some code which only appears if <condition> is met
%elif<condition2>
    ; only appears if <condition> is not met but <condition2> is
%else
    ; this appears if neither <condition> nor <condition2> was met
%endif
```

```
6
7   _start:
8       %defstr str1 HELLO
9       %substr  mychar str1 1,3
10
11       mov rax, mychar
12       mov [value], rax
13
14      %if mychar=="HEL"
15       mov rax, 1
16       mov rdi, 1
17       mov rsi, value
18       mov rdx, 10
19       syscall
20
21       mov rdi, 0
22       mov rax, 60
23       syscall
24      %else
25
26       mov rdi, 0
27       mov rax, 60
28       syscall
29
30      %endif
31
```