Data type

At the level of 80x86 architecture (assembly language level), the memory can be accessed only by using the offset computation formula (base + index x scale + constant), without variables having associated a data type.

The task of data definition directives is not to specify an associated data type to a variable, but to generate the corresponding number of bytes for a named memory area following the little endian representation.

The general form of a data definition source line is:

① [name] data_type expression_list.

name = label for data referral

data_type = size of representation and its value will be the address of its first byte. One of the following:

db - byte data type
dw - word data type
dd - double data type
:

ex: a db 'a' - 1 byte
b dw 100 - 2 bytes
c dd 1000h - 4 bytes
d dq 'abcd' - 8 bytes
e dt 10 - 10 bytes.

② [name] allocation_type factor.

factor = number which shows how many times the allocation type/expression list is repeated.

allocation type = uninitialized data reservation directives.

Ex: resb = byte data type
resw = word data type
resd = double data type
resq = 8 byte data type
rest = 10 byte data type.

number resw 100h - reserves 256 words for "number" array

(11) [name] TIMES factor data-type expression_list.

TIMES directive allows repeated assembly of an instructions & data definition.

Ex: zin TIMES 80 db 'a'

↳ creates an array of 80 bytes every one of them being initialized with the ASCii code of 'a'.

Equ directives:

- allows assigning a numeric value or a string during assembly time to a label without allocating any memory or bytes generation.

syntax: index equ 1000h.

Conversions classification:

1. a) destructive: cbw, cwd, cwde, ~~cdq~~ movzx | movsx, mov Ah

Ex: mov AL, -1 ; AL = FF —cbw→ AX = FFFF =)
cbw ; => the content in Ah was destroyed in extending AL to Ax.

```
mov AL, 100   ! AH was overwritten with 0 destroying the
mov AH, 0     !    previous contents.
```

b) non-destructive : byte, word, dword, qword.

```
Ex: data segment                  !
      a db 5                      ! the only content that was modified
    code segment                  !    was the register AL.
    mov AL, byte[a]               !
```

2 a) signed : cbw, cwd, cwde, cdq, movsx.

```
mov AL, -2   ! AL = FE, being a negative number in the signed
cbw          !    interpretation (cbw) AX = FFFE, extended with the
             !                              sign.
```

b) unsigned : movzx, mov AH, 0, mov dh, 0.

```
mov AL, 5     ! AX = 0005, extended with 0 being placed
movzx AX, AL  !
```

3. a) by enlargement : all the destructive ones + word + dword + qword.

```
Ex: data segment:                 !
      a db 10                     ! takes a word from a's starting
    code segment.                 !    address
      mov AX, word[a]             !
```

b) by narrowing : byte, word, dword

```
Ex: d.b                           !
      a dd 1000h                  ! takes only a byte from a's sta
    c.b                           !    ting address, the least significa
      mov AL, byte[a]             !    ant
```

d) implicit v explicit conversion.

float -> integer ✗

integer -> float ✓ (implicit).

There are cases where the type of data doesn't need to be specified, like:

```
mov AL, 5 | AL only fits a byte, so specifying byte(5) is not needed.
```

but there are other situations where it is absolutely necessary to specify (otherwise syntax error: invalid operator).

```
mov [mem], 12
(i) div [mem]; (ii) mul [div]
push [mem], pop [mem].
```

Ex: mov [v], 0 - syntax error: size not specified.