**String constants. Memory layout and using them in data transfer instructions.**

When initializing a memory area with string type constants (sizeof > 1), the data type used in definition (dw, dd, dq) does only the reservation of the required space, the “filling” order of that memory area being the order in which the characters (bytes) appear in that string constant:

a6 dd '123', '345','abcd'    ; 3 doublewords are defined, their contents being

31 32 33 00|33 34 35 00|61 62 63 64|

a6 dd '1234' ; 31 32 33 34

a6 dd '12345' ; 31 32 33 34|35 00 00 00|

a7 dw '23','45' 32 33 34 35| - 2 words = 1 doubleword

a7 dw '2345' - 2 words - 32 33|34 35|

a7 dw '23456' - 3 words - 32 33|34 35|36 00|

'…' = "…"

a8 dw '1', '2', '3'    - 3 words - 31 00 32 00|33 00

a9 dw '123'    - 2 words - 31 32|33 00

The following definitions provide the same memory configuration:

dd 'ninechars' ; doubleword string constant

dd 'nine','char','s' ; 3 doublewords

db 'ninechars',0,0,0 ; “filling” memory area by a bytes sequence

A character constant with more than one byte will be arranged with little-endian order in mind: if you code

mov eax,'abcd'

then the constant generated is not 0x61626364, but 0x64636261, so that if you were then to store the value into memory, it would read abcd rather than dcba. This is also the sense of character constants understood by the Pentium's CPUID instruction.

….but if you code a data definition like a7 dd '2345' the corresponding memory layout will be NO little-endian representation, but |32 33 34 35|

So, comparatively and in short:

a7 dd '2345' ; |32 33 34 35|

a8 dd 12345678h ; |78 56 34 12|

…….

mov eax, '2345' 🡪 EAX = '5432' = 35 34 33 32

mov ebx, [a7] 🡪 EBX = '5432' = 35 34 33 32

DIFFER from the behavior of the numeric constant when they appear in a data transfer instruction:

mov ecx, 12345678h ; ECX = 12345678h

mov edx, [a8] 🡪 EDX = 12345678h

In the case when DB is used as a data definition directive it is normal that the bytes order given in the constant to be also kept in memory in similar way (little-endian representation being applied only to data types bigger than a byte!), so this case doesn’t need an extra analysis.