23 ian. 2019

2. a1 db '256'

* le consideră caractere, pe fiecare dintre ele separat => în memorie va fi '2'|'5'|'6'

a2 dw 256, 256h

* $256 = 2^8 = 1\ 0000\ 0000_b = 0100h$ => în memorie va fi 00|01
* 256h => în memorie va fi 56|02

a3 dw $ + a2

! adunarea de pointeri nu e validă

a4 equ -256/4

! nu ocupă memorie

a5 db 256 >> 1, 256 << 1

* >> este de fapt o împărțire cu 2^ nr. cu care se shiftează
  => 256 >> 1 = $2^8 : 2 = 2^7 = 1000\ 0000 = 80h$ => în memorie va fi 80
* << este de fapt o înmulțire cu 2^ nr. cu care se shiftează
  => 256 << 1 = $2^8 \cdot 2 = 2^9 = 0010\ 0000\ 0000$
  Suntem pe byte => se salvează doar ultimul octet => în mem. va fi 00

a6 dw a5 - a2, !(a5 - a2)

* a5 - a2 = 4 => în memorie va fi 04|00
* !(a5 - a2) = !4 = 0 => în memorie va fi 00|00

a7 dw [a2], \a2

! valoarea din a2 nu e determinabilă la momentul asamblării
! nu se pot face operații pe biți decât cu valori scalare

a8 dd 256h ^ 256, 256256h

* 256h ^ 256

256h = 0000 0010 0101 0110
256 = 0000 0001 0000 0000

=> 256h ^ 256 = 0000 0011 0101 0110 = 0356h pe word => 00 00 03 56h pe dword
=> în memorie va fi 56|03|00|00

* 256256h => în mem. va fi 56|62|25|00

a9 dd $-a9

* prima dată atribuie valoare, apoi crește contorul => va fi 0 => în mem. 00|00|00|00

a10 db 256, -255

* $256 = 2^8 = 1\ 0000\ 0000$, suntem pe byte deci se salvează numai ultimul octet => 00 în mem

* $-255 = -255 + 256 = 1$ => în mem. va fi 01

! pe byte: scăderea sau adunarea cu 256 nu schimbă ultimul octet

a11 dw 256h-256

$$= 0000\ 0010\ 0101\ 0110 - \\ 0000\ 0001\ 0000\ 0000 \\ \hline 0000\ 0001\ 0101\ 0110 = 0156h$$ => în memorie va fi 56|01

a12 dw 256-256h

$$= 0000\ 0001\ 0000\ 0000 - \\ 0000\ 0010\ 0101\ 0110 \\ \hline 1111\ 1110\ 1010\ 1010 = FE\ AA$$ => în memorie va fi AA|FE

a13 dw -256

(aici nu merge regula cu +256 pt. că suntem pe word)

$|-256| = 256 = 1\ 0000\ 0000$

$C_2(256) = 1111\ 1111\ 0000\ 0000 = FF\ 00$ => în memorie va fi 00|FF

↳ pe word

a14 dw -256h

$|-256h| = 256h = 0000\ 0010\ 0101\ 0110$

$C_2(256h) = 1111\ 1101\ 1010\ 1010 = FD\ AA$ => în mem. va fi AA|FD

a15 db 2,5,6,25,6,2,56 => în mem. va fi 02|05|06|19|06|02|38

$$\begin{array}{r|l} 25 & 16 \\ \hline 16 & 1 \\ \hline 9 & 0 \\ & 1 \end{array} \quad \begin{array}{l|l} 16 & \\ \hline 1 & 16 \\ \hline 0 & 0 \end{array} \Rightarrow 25 = 19h$$

$$\begin{array}{r|l} 56 & 16 \\ \hline 48 & 3 \\ \hline 8 & 3 \end{array} \quad \begin{array}{l|l} 16 & \\ \hline 3 & 16 \\ \hline 0 & \end{array} \Rightarrow 56 = 38h$$

a)

a1) * mov ah, 129

mută pe 129 în ah (fără semn)

* mov bh, 97h mută pe 159 în bh

97 = 1001 1111 b = 16·9 + 15 = 159 (fără semn)

* add ah, bh

adună 129 + 159 = 288

nu încape pe octet => se va salva 288 - 256 = 32 în ah

CF = 1 (nu a încăput pe octet fără semn)

signed: 159 = 159 - 256 = - 97

129 = 129 - 256 = -127

=> -97 + (-127) < -128 => OF = 1

ZF = 0 (rezultatul nu a fost 0)

SF = 0 (semnul a cu se salvează în ah)

a2) * mov ax, 128

mută în ax pe 128 (signed și unsigned) = $2^7$ = 0000 0000 1000 0000

* sar al, 7

shift aritmetic right => completează cu bitul de semn

=> în al va fi 1111111

* înmul ah => al * ah = -1 * 0 = 0 => în ax va fi 0

! ZF nu se setează la înmulțire și împărțire

SF = 0 (0 e considerat nr. pozitiv)

CF = OF = 0 (0 a încăput pe octet)

a3) * mov ax, 256 => pune în ax pe ah: 0000 0001 0000 0000 = 0100h

* mov bx, -1 => pune în bx pe bh: FF FFh

* add ah, bh => 01h + FFh = 0000 0001 +
1111 1111
10000 0000 => în ah va fi 00

SF = 0 (0 e considerat pozitiv)
ZF = 1 (a încăput în ah)
CF = 1 (depășire fără semn)
+ + (-) = + => OF = 0 (nu e unul dintre cazurile în care OF se setează)

a4) * mov ah, 128/2

128 = 1000 0000
2 = 0000 0010 } => 128/2 = 1000 0010

* mov bh, 90h >> 3

90h = 1001 0000 => 90h >> 3 = 0001 0010

* sub ah, bh

=> ah - bh = 1000 0010 -
0001 0010
0111 0000 => CF = 0
||
70h (-) - (+) = + => OF = 1
||
112 signed și unsigned

ZF = 0 SF = 0 (număr pozitiv

b) b1) movsx ax, al

b2) cbw

b3) nu există, nu pot fi ambii operanzi din memorie

b4) movsb

b5) movsb