ASC streximi exercitii 2

18 februarie 2019 (20)

a) Present the corresponding memory layout of the following segment (its structure and the corresponding memory ~~layouts~~ values, starting offset 0), justifying in context the reason for obtaining each of the stored values:

X dw -256, 256h

(*) -256 - in base 2, then in base 16

256 = 2^8 = 1 0000 0000

256 (word) = 1111 1111 0000 0000b = FF00h

in memory: 00 FF

(*) 256h

in memory: 56 02

00|FF|56|02|

y dw 256|-256, 256h & 256

| - or operation

We transform 256 in hexa

256 = 100h

-256 = 100h

How it would look in binary

256 = 0000 0001 0000 0000b

-256 = 1111 1111 0000 0000b

256|-256 = 1111 1111 0000 0000b = FF00h, in memory: 00|FF

256h = 0000 0010 0101 0110b

256 = 100h = 0000 0001 0000 0000b

256h & 256 = 0000 0000 0000 0000b = 00 00h in memory: 00, 00

so, all of it in memory: 00|FF|00|00|

7 db $-7, y-x

$ - the current address

$ - 7 = 0, in memory: 00
y - x = 4, in memory: 04
} in memory: |00|04|

db 'y'-'x', 'y-x'

In assembly, the db directive can be used to insert raw data into the program's data section without necessarily giving it a name, so the instruction is correct

'y'-'x': it makes a substraction of their ascii codes

'y'-'x' = 1, in memory: 01

'y-x': it writes the ascii code of each character

in memory: 'y'|'-'|'x'|

all of it in memory: |01|'y'|'-'|'x'|

a db 512>>2, -512<<2

512 >>2

512 in heva: 200h

200 h = 0010 0000 0000

>>2 = 0000 1000 0000 = 80h

in memory: 80 h

-512 = 200h = 1111 0000 0000b

on word: 1111 1111 0000 0000b

<<2: 1111 1100 0000 0000b = FC 00h

because it is declared as byte, we take only one byte:

in memory: 00 h

all of it in memory: |80|00|

b dw z-a, !(z-a)

z-a

offset of z: 8
offset of a: 14
} z-a = -6

-6 in binary as a word:

We use 2's complement

6 = 0000 0000 0000 0110b

we invert the bits and add 1 bit

1111 1111 1111 1001b +
1b

1111 1111 1111 1010b = FFFA

in memory: FA FF

!(z-a)

Apparently, the assembler does not use "!" to inverts bits but it makes it all 0

!(-6) = 0 = 00 00h

in memory: 00 00

all of it in memory: FA FF 00 00

c dd ($-b)+(d-$), $-2*cs+3

($-b)+(d-$)

$-b = 4
(d-$) = 4
} 4+4 = 8, in dword: 00 00 00 08h

in memory: 08 00 00 00

$-2*cs+3

POINTERS CAN'T BE USED FOR MULTIPLICATION

=> Syntax error

Only one dw is in memory, so d = c+4, from here we know d-$ = 4

d db -218, 128^(~128)

-218:

we transform 218 in hexa and determine its 2's complement

-218 = 26h

128^(~128)

^: or

~: complement of 1

128^(~128) = 128 or its complement of 1 = FF

all in memory: |26|FF|

e times 2 resw 6

it reserves in memory 6 words * two times

|00|00|00|00|00|00|00|00|00|00|00|00|00|00|00|00|
|00|00|00|00|00|00|00|00|

times 2 dd 1234h, 5678h

it declares 1234h, 5678h two times, in word types

1234h:

in memory: |34|12|00|00|

5678h

in memory: |78|56|00|00|

all of it in memory: |34|12|00|00|34|12|00|00|78|56|00|00|
|78|56|00|00|

NOT GOOD!

|34|12|00|00|78|56|00|00|34|12|00|00|78|56|00|00|

way better

b) The following ASM sequence is given:

```
mov bh, 7Fh ;
cmp bh, al  ;  not too relevant, ax is not changed
ror ah, 1   ;  0011 0000 | 0000 1010
sar ah, 7   ;
```

Write one single ASM instruction having the same effect (except flags). Justify.

7F = 0111 1111

cmp bh, al - fictive substraction (bh - al)
- if the sign bit of al = 1 => CF = 1
else => CF = 0

! shifts does not transmit from ah to al

ror - puts the CF value on the first position in al

sar ah, 7 -> shift arithmetic right (completează cu sign bit)

=> in ah will be the sign bit of al

equivalent instructions:

```
movsx ax, al ; or
cbw
```

23 January 2019

Present the correct memory layout of the following segment, justifying

```
a1 db '256'
```

It will store in memory the Ascii code of each bit

in memory: |'2'|'5'|'6'|

a2 dw 256, 256h

The first 256 is in decimal, so we transform it in hexa

256 = 100h, as a word: 01 00

in memory: |00|01|

256h: as a word: 02 56

in memory: |56|02|

all the declaration in memory: 00|01|56|02

a3 dw $+a2

~~$ is the current adress~~

~~$ = 7~~

~~$+a2 = 7+3 = 10 = Ah, as a word: 00 0A~~

~~in memory: 0A 00~~ WRONG!

addition with pointers is not valid

=> SYNTAX ERROR

a4 equ -256/4

/ - unsigned div

equ declares a constant => a4 is not in memory

a5 dw 256 >> 1, 256 << 1

256 = 100h = 01 00

= 0000 0001 0000 0000b

>>2: 0000 0000 1000 0000b = 00 80 it takes only one byte

=> in memory: 80

<<2: 0000 0010 0000 0000 = 02 00

=> in memory: 00

all of it in memory: |80|00|

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

a5-a2 = 7-3 = 4

in word: 00 04

in memory: |04|00|

!(a5-a2) = !(4) = 0

in memory: 00 00

all of it in memory: |04|00|00|00|

a7 dw [a2], ~a2

SYNTAX ERROR

[a2] is incorrect

~ operator may only be applied to scalar values

a8 dd 256h^256, 256256h

256h = 00100101 0110

256 = 100h = 0001 0000 0000

256h^256 = 00110101 0110

= 356h

in doubleword: 00 00 03 56

in memory: 56|03|00|00

256256h

in double: 00 25 62 56

in mem: 56 62 25 00

all: 56|03|00|00|56|62|25|00|

a9 dd $-a9

$ is the current offset

$-a9 = 0

in double, in memory: |00|00|00|00|

a10 db 256, -255

256 = 100 h = 0100 in bytes

in memory: it takes a single byte: 00

-255:

|-255| = 255 = FFh = 1111 1111

we determine its 2's complement

1111 1111

0000 0000 +

1

0000 0001 = 01 h

in memory: 01

all of it in memory: 1001011

a11 dw 256h-256

256 = 100h

256h - 256 = 256h - 100h = 156h

in word: 0156

in memory: 56|01

a12 dw 256-256h

256-256h = 100h - 256h = -156h

+|156| = 156

we determine its 2's complement

156h = 0000 0001 0101 0110b

~: 1111 1110 1010 1001b +

1

1111 1110 1010 1010b = FEAAh

in memory: AA|FE

a13 dw -256

|-256| = 256 = 100h

= 0000 0001 0000 0000b

~: 1111 1110 1111 1111b +

1

1111 1111 0000 0000b = FF00

in memory: 00|FF

8

a14 dw -256h

|-256| = 256h

= 0000 0010 0101 0110

N: 1111 1101 1010 1001 +
1

1111 1101 1010 1010 = FDAAh

in mem: AA|FD

a15 db 2, 5, 6, 25, 6, 2, 56

2 = 2h, in memory: 02

5 = 5h, in memory: 05

6 = 6h, in memory: 06

25 = 19h, in memory: 19

6: in memory: 06

2: in memory: 02

56 = 38h, in memory: 38

memory: 02|05|06|19|06|02|38|

8 Februarie 2018

a) Which is the MINIMUM number of bits necessary for representing:

i) 61

61 = 3Dh = 0011 1101b

the minimum number of bits for representing 61 is 6:
111101b

ii) -62

|-62| = 111110b

The number is negative, so it has to be represented in signed, so we need one more bit to be the sign bit

-62 = 1000010b => the minimum number of bits for

representing -62 is 7 bits

!. Are legătură cu interpretarea signed și unsigned

iii) 130

130 : 16 = 8 r. 2

8 : 16 = 0 r. 8

130 = 82h

130 = 1000 0010b

the minimum bits of representing 130 is 8

iv) -129

|-129| = 129

129 = 81h

81h = 1000 0001 b

-129 = 1011 1101b

the minimum bits of representing -129 is 9 bits, because we also need a sign bit

b) xor ah, ah; puts 0 in ah

code; puts 0 in the higher part of eax, bc. the most significant bit in ah is 0

add ebx, eax; puts in ebx the value of eax

mov al, [ebx]; al will be [ebx + al]

equiv: xlat

February 9 2018

e) Represent the corresponding memory layout, justify

a1 db '256-256'

it represents the ascii code of each bit

in memory: '2'|'5'|'6'|'-'|'2'|'5'|'6'

a2 dw 256, 256h

256 = 100h

in word: 0100

in memory: 00 01

256h

in word: 0256

in memory: 56 02

all: 00|01|56|02

a3 dw $-a2

$ - the current address

$-a2 = 11-7 = 4

in word: 00 04

in memory: 04|00

a4 equ -256/4

equ - it doesn't put a4 in the memory

as it was a4 db -256/4

+256/4 = 64

+64 = 40h = 0100 0000b

~: 1011 1111 +

1

1100 0000b = c0h

in memory: c0

same for a4 db -256//4

a5 db 128>>1, -128<<1

128 = 100h

128 = 0001 0000 0000b

>>1 = 0000 1000 0000b = 0010 h

it takes only one byte => in memory: 10

-128: we determine its 2's complement

~: 1110 1111 1111 b+

1

1101 0000 0000 = FF00h

in memory: 00

all: 10|00

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

a2-a5 = 7-13 = -6

|-6| = 6h =

in word: 00 06 = 0000 0000 0000 0110b

~: 1111 1111 1111 1001 +

1

1111 1111 1111 1010b = FFFAh

in memory: FA|FF

~a2-a5 = ~1111 1111 1111 1010

= 0000 0000 0000 0101 = 0005

in memory: 05|00

all: FA|FF|05|00

a7 dd [a23], !a2

[a23 - syntax error

!a2 = error: ! operator may only be applied to scalar values

a8 dd 256h^256, 256256h

256 = 100h

256h^256 = 256h^100h

= 0000 0010 0101 0110b^
  0000 0001 0000 0000b

= 0000 0011 0101 0110b = 0356h

on double: 00 00 03 56

in memory: 50|03|00|00

256256h

on double: 00 25 62 56

in memory: 56 62 25 00

all: 50|03|00|00|56|62|25|00

a9 dd ($-a8)+(a10-$)

$-a8 = 8

a10-$ = 4 because at a9 is declared a doubleword

($-a8) + (a10-$) = 8+4 = 12 = 0Ch, as a dould: 00 00 00 0C

in memory: 0C|00|00|00

a10 dw -255, 256

|-255| = 255 = FFh

its 2's complement:

 1111 1111

~: 0000 0000 +

 0000 0001 = 01

as word: FF 01, in mem: 01|FF

256 = 100h

in mem: 0000

all: 01|FF|00|01

a11 resb 6

reserves 6 bytes:

00|00|00|00|00|00

a12 times 4 dw 256

256 = 100h

as a word: 01 00

in memory: it is declared 4 times:

00|01|00|01|00|01|00|01|

a13 dw times 4 128

SYNTAX ERROR!

it should be a13 times 4 dw 128

times 2 resw 2

reserves 2 words 2 times

00|00|00|00|00|00|00|00

times 2 dd 12345678h

in memory: 78 56 34 12

declared two times:

78|56|34|12|78|56|34|12

28 January 2022

a) the same exam

a1 dd 'oabcdefgh', oabcdefgh

'oabcdefgh'

it doesn't matter that it's declared as double, it still puts the ascii of each bit

in memory: 'o'|'a'|'b'|'c'|'d'|'e'|'f'|'g'|'h'|

oabcdefgh

in memory: ef|cd|ab|00

all: 'o'|'a'|'b'|'c'|'d'|'e'|'f'|'g'|'h'|'ef|cd|ab|00|

a2 dw 'oabcdefgh', 3/6

'oabcdefgh':

in memory: 'o' 'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h'

3/6

3 = 3h

as word: 00 03h = 0000 0000 0000 0011b

6 = 6h

as word: 00 06h = 0000 0000 0000 0110b

3/6 = 3 or 6

= 0000 0000 0000 0111 = 00 07

in memory: 0700

a3 dw $ - a2, a2 - a1

$ - the current address

$ - a2 = 10 = ~~10h~~ 0Ah

a2 - a1 = 12 = 0Ch

in memory: 0A|00|0C|00|

a4 db 129 >>1, -129 <<1

129 = 81h = 1000 0001b

>>2 = 01000000b = 40h

in memory: 40

-128|-129 = 81h = 1000 0001b

~ = 01111110 +

1

01111111b

<<1 = 11111110b = FEh

memory: |40|FE|

a5 dw a2 - a4, ~(a2 - a4)

a2 - a4 = -14

|-14| = 14 = Eh

in word: 00 0Eh = 0000 0000 0000 1110

~ = 1111 1111 1111 0001 +

1

1111 1111 1111 0010 = FF F2h

in memory: F2|FF

~(a2-a4)

here, a2-a4 gives a value, so the declaration works, in comparation with ~a2 which would not work

~(a2-a4) = ~ 1111 1111 1111 0010b = 0000 0000 0000 1101b

= 00 0Dh

in memory: 0D|00

all: F2|FF|0D|00|

a6 dd $+a2-1, !a2

$+a2-1 - SYNTAX ERROR

pointers can not be added

!a2 - error

'!' operator may only be applied to scalar values

a7 dd 256h^256, 256256h

I - am mai lucut

56|03|00|00|56|62|25|00

a8 dd ($-a7)+(a9-$), -256

$-a7 = 8

a9-$ = 8 because a8 has 8 bytes declared

8+8 = 16 = 10 h

as double: 00 00 00 10 in memory: 10|00|00|00|

-256:

|256| = 256 = 100h

as double: 00 00 01 00

= 0000 0000 0000 0000 0000 0001 0000 0000

~ = 1111 1111 1111 1111 1111 1110 1111 1111 + 1

= 1111 1111 1111 1111 1111 1111 0000 0000 = FF FF FF 00

in mem: 00|FF|FF|FF

all: 10 00 00 00 00 FF FF FF

a9 dw -255, -128

|255| = 255 = FF

= 0111 1111b

~ : 0000 0000+

1

: 0000 0001 - its 2's complement = 01

as word: FF 01, in memory: |01|FF|

|-128| = 80h = 1000 0000 b

~ = 0111 1111b+

1

= 1000 0000b = 80h

as word: FF80, in memory: 80 FF

all: |01|FF|80|FF|

a10 times 4 dw 128h, -128

128h as word: 01 28, in memory: 28 01

-128 as word: FF 80, in memory: 80 FF

they are declared 4 times

all: |28|01|80|FF|28|01|80|FF|28|01|80|FF|28|01|80|FF|

a11 db a3

a3 is the address.

a3 = 22 = 16h

in memory: 16h

a12 dw a3

a3 = 22 = 16h

as word: 00 16

in memory: 16 00

4 Februarie 2022

a1 dd -255, a5

-255:

2's complement: 256 - 255 = 1

as a double: FF FF 01

in memory: 01|FF|FF|FF

a5:

at a1 we have 2 doubles = 8 bytes

a2: 6 words = 12 bytes

a3: 2 bytes

a4: 4 bytes

so, a5 = 8+12+2+4 = 26 = 1Ah

as a double: 00 00 00 1A

in memory: 1A|00|00|00

all: 01 FF|FF|1A|00|00|00

a2 dw 1^2, 384, 516, 2^3, 485, 116

1^2: 0001b ^
0010b
0011 = 3h

in memory: 03|00

384 = 0011b 2
0100b
0000b

in memory: 00|00

516 = 0101b
0110b
0111b = 7h

in memory: 07|00

18

2 ^ 3 = 00101
        0011
        0001

in memory: 01|00

485 = 01000
      0101
      0100

in memory: 04|00

116 = 0001
      0110
      0111

in memory: 07|00

all: 03|00|00|00|07|00|01|00|04|00|07|00

a3 db ~((-1)^0CCh), 1^0CCh

-1: 256-1=255 = FFFFh

FF55h ^ 0CCh = 1111 1111 1111 1111
               0000 0000 1100 1100b
               1111 1111 0011 0011b

~ = 0000 0000 1100 1100 = 00CCh

in memory: 66

1^0CCh = 0000 0000 0000 0001b ^
         0000 0000 1100 1100b
         0000 0000 1100 1101b = CDh

in memory: CD
all: CC|CD

a4 dw $-$$, $$-a4

$-$$ = 22 = 16h

$$-a4 = -22

256-22 = 234 = 0EAh = FFEAh

all in memory: 16|00|EA|FF

05 dd 'oabcdefgh', -127<<1

'oabcdefgh'

in memory: 'o' 'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h'

-127 <<1

256 - 127 = 129 = 81h

as double: FF FF FF 81

FF81 = 1111 1111 0001 1000

<<1 = 1111 1110 0011 0000 = FE30h

so, the result: FF FF FE 30

memory: 'o' 'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 30 FE FF FF

06 dd -1<<1Fh, 11h>>11b

-1<<1Fh

-1:

256 - 1 = 255 = FFh

as a double: FF FF FF FFh

1Fh = 31

FF FF FF FF: from 32 bits of 1, 31 will be shifted,

so one bit of 1 will remain as the most significant

bit => 80 00 00 00

11h >> 11b

= 0001 0001 >> 3

= 0000 0010 = 02h

as double: 00 00 00 02

memory: 00 00 00 08 02 00 00 0e

07 times 4 dw a2, a2+1

a2 = 8, as word: 00 08

a2+1 = 9, as word: 00 09

declared 4 times:

in memory: 08 00 09 00 08 00 09 00 08 00 09 00 08 00 09 00

20

a8 dw !(a2-a1), !(a2-1)

a2-a1 = 7

!(a2-a1) = !7 = 0

in memory: 0010001

(a2-1)

SYNTAX ERROR: '!' operator may only be applied to scalar values

a9 dd ~(256/256h), ~256/256h

~(256/256h)

256/256h = 100h/256h, as double: 0000 0200 / 0000 0256

= 0000 0356

~: FF FF FC A9

~256/256

FF FF FF FE

in mem: A9 FC FF FF FF FE FF FF

code segment

lea eax, a2

SYNTAX ERROR - invalid combination of opcode and operands

ES lodsb

loads in al a byte from [Extra segment x ES:ESI]

movzx ebx, [ebx+6]

SYNTAX ERROR: operation size not specified

[ebx+6] should be a byte/word in order to be zero-extended in ebx

xchg ebx, [ebx+6]

Exchanges the value from ebx with the 32 bit value in the memory starting from [ebx+6]

push dword [ebx + esp]

Loads in the stack the value from the memory starting from ebx + esp in dword size

In this formula: esp - base, ebx - index

Pay attention, everything has to be mentioned

push dword [ebx + ebp]

Loads in the stack the value from the memory starting from ebx + ebp in dword size

mov ax, a+b

~~SYNTAX ERROR - pointers cannot be added~~ NO

adding two pointers: it's not a syntax error but it does not make sense

b) Detaliati efectele fiecărei surse de cod. O singură instrucțiune cu aceleași efect asupra stării

shl eax, 1 ; shifts the value from edx register to the left with 1 bit position

rcr edx, 1 ; rotates the bits from eax to the right the value of the CF will be added to the left hand side, then the bit will be kept in CF

times 2 sar edx, 16 ; sar instruction shifts the bits from edx with 16 positions, and the leftmost bits will be filled with the sign bit. This happens two times so the edx register will be filled with the sign bit

rcr eax, 1 ; same explication from rcr edx, 1

27 ianuarie 2023

I'm lazy and I don't have time so I will write only what I haven't done before

z db z, $-z

Pay attention!!

syntax error: dej format can only handle 16 or 32-byte relocations

p dd ($-y)*(x-$), y-x

too weird - it should be expression syntax error

p dd ($-y)*(x-$) works

a dw $-$$, $$-a

$ - the current offset = the offset of a

$$ - the offset from the start of segment = 0

$-$$ = the offset of a

$$-a = 0 - the offset of a = -a

b dw x+y-z+18, x-y+z-18

x+y-z+18 - is a scalar, because first we add 2 pointers => pt. and then we sub. 1 pointer => scalar

x-y+z-18 - pointer, because we sub. 2 pointers => scalar and we add 1 pointer => pointer

Normally the addition of two pointers works but it doesn't make sense so idk

f dw x+1, [x+1]

[x+1]:

SYNTAX ERROR: Expresion syntax error

K dw 1+2h+3h+a, a+0ah

both operands are pointers!

m dd 0ah+0bh, a+b

0ah+0bh -> 15h

a+b -> the addition of two pointers

30 ian. 2024

a db $$-$, h-11b

$$-$ -> 0-$ = -$

h-11b = the pointer h-3'

s dd a-start, start-start1 ; start si start1 def. in CS

Aici nu stiu