EXAM 3

62 | 61
48 | 48
14 | 13

February 8, 2018

I.

II min nr of bits

i) 61

61 = 3Dh = 0011 1101b

the min nr of bits necessary for rep 61 is 6 : 11 1101b

$61 \in [0, 2^6] \Leftrightarrow 61 \in [0, 64]$

ii) -62

-62 = 2's(62)

62 = 3Eh = 0011 1110b $\Rightarrow$ -62 = 1100 0010b

$-62 \in [-2^{6}-1, 2^6]$ $\Rightarrow$ the min nr. of bits = 7

(formula for signed : $[-2^{n-1}-1, 2^{n-1}]$)

iii) 130

130 = 82h = 1000 0010b

min nr. of bits for rep. 130 is 8 : 1000 0010b

$130 \in [0, 2^8-1]$

iv) -129

-129 = 2's(129)

129 = 81h = 1000 0001b ~~$\Rightarrow$ -129 = 0111 1111b~~

= 0000 0000 1000 0001 $\Rightarrow$ -129 = 1111 1111 0111 1111b

$\Rightarrow$ the min nr. of bits for rep -129 is 9

$-129 \in [-2^8-1, 2^8]$

b) xor ah, ah ; ~~ah ← 0~~ puts in the high part 0's

cwde , ~~ax → eax~~ convert AX → EAX

add ebx, ~~ebx~~ ; ~~adds to eax, ebx~~ ebx ← ebx + al

mov al, [ebx] ; al ← [ebx + al]

} (=) xlat (translates the byte AL into a corresponding byte from a translation table defined)

[DS : EBX] - address of translation table
explicit

III.

a) mov ax, 1000h

mov bl, 1000b + 10b

div bl

ax = 1000h = 4096d (signed and unsigned)

1000b = 8d
10b = 2d
⇒ 1000b + 10b = 10d = 0Ah (signed and unsigned)

div bl ⇒ AX : BL = 4096 : 10 = 409 r 6

~~⇒ AX = 409 = 0199h (quotient)~~

~~DX = 6 = 0006h (remainder)~~

~~For this case, there is not overflow.~~ Overflow for division

⇒ AL = 409 = 0199h (quotient)

AH = 6 = 0006h (remainder) ; but 0199h doesn't fit a byte so there will be an overflow marked by a runtime error (the program will stop after the instr ~~mov~~ div bl)

256 | 16
1536 | 3
256
4096

409
256
153 : 16 = 9
144
= 9

130 : 16 = 7
112
18

130 : 16 = 8
128
2

b) mov ah, 0beh
mov al, 0deh
add ah, al

ah = 0beh = 1011 1100b
al = 0deh = 1101 1110b

add ah, al

```
  1111 1
  1011 1100 +
  1101 1110
-------------
1 1001 1010
```

- there's a digit outside of the binary configuration of the final result (a transport digit) ⇒ CF = 1
- since, in the signed interpretation, there is an addition of two negative numbers, and the result is still a negative one ⇒ no overflow or s.i. ⇒ OF = 0

c) mov ax, 1001h
mov bx, 1111b
imul bl

ax = 1001h = 4096 + 1 = 4097d (→ al = 1)
bx = 1111b = 15d 000Fh
bl = ~~F0~~ 0Fh = ~~-1~~ 15 (signed and unsigned)
imul bl ; al * bl = ~~1 * (-1) = -1 ⇒ result in AX = FFFFh~~
= 01 * 15 = 15 ⇒ result in AX = 000Fh
~~OF = CF = 0 (b*b = b)~~

OF = CF = 0 (b * b = b)

d) mov dh, 62h
mov ch, 200
sub dh, ch

62h = 96 + 2 = 98d (signed and unsigned) 0110 0010
200 = c8h = 1100 1000b = 200 (unsigned), -56 (signed)

200 : 16 = 12
128 8
72 8

sub dh, eh

CF=1
0110 0010 −
1100 1000
0000 1010

→ fa the subtraction, there was needed a borrow
so CF = 1
→ pos − neg = pos (no overflow on signed int) →
⇒ OF = 0

In assembly language, an overflow means that a result of a performed operation didn't fit in the admissible size representation or that the result doesn't belong to the specified interval or that the resulted operation is incorrect from the ~~point of view~~ mathematical point of view.

The overflow for unsigned numbers is marked by the CF when there appears a carry outside the resulted binary config.

ex - -
subtraction (borrowed)

add
OF → neg + neg = pos
pos + pos = neg

sub
CF → neg − pos = pos
pos − neg = neg

OF = CF for mult (but there's no overflow result = 2 * sizeofop)

for overflow div ⇒ runtime error.

EXAM 4

February 9, 2018

I Overflow concept

Overflow rep a situation (condition) in which the result didn't fit in the reserved size space or the res. doesn't belong to the admis. rep interval or the operation is a math. nonsense in that particular interpretation (signed/unsigned)

- Addition (CF=1: a+b > interval, digit outside, OF=1 (neg+neg=pos, pos+pos=neg)
- Subtraction (~~set~~ CF=1 (needs a borrow digit, OF=1 (pos-neg=neg, neg-pos=pos)
- Multiplication (No overflow, but CF=OF=0 : b*b=b, CF=OF=1 b*b=W)
- Division (res don't fit on quotient ⇒ "Run-time error" by OS "providing one of: "Div overflow", "~~Zero division~~", "Div by zero", "zero divide"

→ The prog. • should be careful about ~~the~~ allocating the right size for every explicit data, by using a suitable data type very time
• checking the flags after very execution (CF, OF)

→ The solving of these ~~not~~ situations mean changing the data type, so that is respecting the correct size for very

II. a1 db '256, -256' | 32 35 36 '-' 32 35 36

→ in mem there are placed the ascii codes for each char in a byte

⇒ memory : 32 35 36 '-' 32 35 36

'2' = 50 = 32h

'5' = 53 = 35h

'6' = 54 = 36h

a2 dw 256, 256h | 00 01 56 02

256 = ~~0010~~ 0100h ⇒ memory: ~~00 10~~ 00 01

256h = 0256h ⇒ memory : 56 02

a3 dw $-a2 | 04 00

$-a2 = 10-6 = 4$ (nr. of bytes between) $= 0004h$

⇒ memory : 04 00

a4 d equ -254/4

→ nothing will be shown in memory (equ- directive for initializing with constants without allocating memory space)

a5 db 128>>1, -128<<1 | 40 00

~~(doesn't fit a byte)~~

128 = 1000 0000 ⇒ 128 >>1 = 0100 0000 = 40h ⇒ memory 40

-128 = 2's(128) = ~~1000 (on a word~~ 1000 0000 ⇒ -128 <<1 = 0000 0000 ⇒ mem : 00

a6 dw a2-a5, ~(a2-a5) | FA FF 05 00

a2 - a5 = 8 - 14 = -6

-6 = 2's(6)

4 = 0000 0000 0000 ~~0100~~ 0110 ⇒ -4 = 1111 1111 1111 ~~1001~~ 1010 = FFFAh ⇒

⇒ memory : FA FF

~(a2-a5) = ~(-6) = ~(1111 1111 1111 ~~1001~~ 1010) = 0000 0000 0000 ~~0110~~ 0101 = 0005h ⇒

⇒ memory : 05 00

a7 dd [a2], !a2 - syntax error (not computable at assembly time a ~~not~~ var's addr not comp. at ass time)

~~!a2 = !(addr of a2 ≠ 0) = 0~~, but not ~~shown in mem. since syntax error~~

! → scalars

a8 dd 256h ^ 256, 256256h | 56 03 00 00 56 62 25 00

256h = 0000 0256h = 0000 0000 0000 0000 0000 0010 0101 0110

256 = 0000 0100h = 0000 0000 0000 0000 0000 0001 0000 0000 } ⇒ 256h ^ 256

= 0000 0000 0000 0000 0000 0011 0101 0110 = 00000356h ⇒

memory: 56 03 00 00

256256h = 00256256h ⇒ memory: 56 62 25 00

a9 dd ($-a8)+(a10-$) | 0C 00 00 00

$-a8 = 8 (nr. of bytes...)
a10-$ = ~~-8~~ +4(a10>$)
($-a8)+(a10-$) → will be a dword ⇒ 8 bytes

⇒ ($-a8)+(a10-$) = 12 = ~~0004h ⇒ memory 0400~~
0008 0008h ⇒ memory
0C 00 00 00 00

a10 dw -255, 256 | 01 FF 00 01

-255 = 2's (255)

255 = 0000 0000 1111 1111 ⇒ -255 = 1111 1111 0000 0001 = FF01h ⇒ memory: 01 FF
(2^8 - 1)

256 = 2^8 = 0000 0001 0000 0000 = 0100h ⇒ memory 00 01

a11 resb 6 | 00 00 00 00 00 00
→ reserves 6 bytes ⇒ memory

a12 times 4 dw 256 | 00 01 00 01 00 01 00 01

256 = 0100h ⇒ memory: ~~00 10~~ 00 01

a13 dw times 4,-128 ; syntax error

~~a14~~

times 2 resw 2 | 00 00 00 00 00 00 00 00
→ res 2*2 words

times 2 dd 12345678h | 78 56 34 12 78 56 34 12

12345678h ⇒ memory: 78 56 34 12

MEMORY LAYOUT

32 35 36 ' ' - 32 35 36 | 00 01 56 02 | 04 00 | 40 00 | F9

a1 a2 a3 a5

FF 06 00 | 56 03 00 00 56 62 25 00 | 04 00 00 00 | 01

a6 a8 a9 a10

FF 00 01 | 00 00 00 00 00 00 | 00 01 00 01 00 01 00

a11

01 | 00 00 00 00 00 00 00 00 | 78 56 34 12 78 56 34 12

a12

EBX = 0034C000

II a)

1) lea ebx, [ebx+6] 0034C006

2) lea ebx, [bx+6] → unsigned extension 0000C006

3) lea bx, [bx+6] 0034C006

4) lea bx, [ebx+6] 0034C006

5) mov ebx, ebx+6 – (SE)

6) mov ebx, [ebx+6] ; tries to load in ebx the content from the address ebx+6 (memory violation error)

7) movzx ebx, [ebx+6] – (SE ?)

8) movzx ebx, [bx+6] → unsigned ex – (SE) ([bx+6] - offset on 32 bits)

9) add bx, 6 ; movzx ebx, bx 0000C006

10) mov size [ebx], dword [bx+6] – (SE)

11) add ebx, 6 0034C006

12) add bx, 6 0034C006

13) push size [ebx+6] ; pop ebx (SE)

14) xchg ebx, [ebx+6] ; tries to put in ebx the content from address ebx+6 (memory violation error)

I modifies entire ebx (refers to entire ebx)

1), 3), 4), 11), 12)

II modifies only bx and the rest = 0

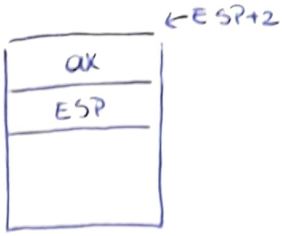
2), 9)

III memory violation error

6), 14)

IV syntax error

5), 7), 8), 10), 13)



b) dd

push ax

mov ax,[esp+2] ; mov ax,ax

stosw ; mov edi, ax ; take ax from the stack and stores it in edi

lea edi,[edi-2] ; mov edi, ax

add esp,4

stosw ; loads in EDI → AX

EDI ← EDI + 2

EXAM 2 January 28, 2019

I - using 2's complement (def.)

- 4 reguli (dii $\overline{100..0}$ - signed, $\overline{0...0}$ - signed, $\overline{100..0}$ = abc ⇒ ? - abc

$\overline{00..0}$ = abc ⇒ ? abc)

$2^m - V$ - 2's complement

- addition, subtraction the same for signed/unsigned
- multiplication (imul), division (idiv)
- 0 - considered a positive nr (if cons. its signed ~~is~~ interp. the sign bit is 0 which translates it as a positive number)

II.

a1 db '256' | ~~50 53 54~~ 32 35 36 '1' = 49 (offset = 00400 1000)

'256' - string - stored as '2' '5' '6' ⇒ memory ~~50 53 54~~

50 = 32h 53 = 35h 54 = 36h

a2 dw 256, 256h | ~~00 01~~ 00 02 (offset = 00400 1003)

256 = ~~0100~~ 0100h ⇒ memory: ~~00 01~~ 00

256h = 0256h ⇒ memory: 56 02 1/2

a3 dw $ + a2 ; synthax error (pointer addition not possible)
(possib. mem. violation error

a4 equ -256/4 | nothing shown in memory, equ is a directive used for
a4 = -64 ; initialising a variable with a constant without
; allocating memory space

a5 db 256 >> 1, 256 << 1 | 80 00 ✓

256 = 2^8 = 1 | 0000 0000

256 >> 1 = 1000 0000 = 80h ⇒ memory: 80

256 << 1 = 0000 0000 = 00h ⇒ memory 00

a6 dw a5-a2, !(a5-a2) | 02 00 00 00

a5-a2 = 8-6 = 2 (nr. of bytes between a5 and a2) ⇒ memory: 02 00
0002

!(a5-a2) = !(2) = 0 ⇒ memory 00 00

a7 dw [a2], wa2 ; syntax error, not determinable at assembly time

a8 dd 256h ^ 256, 256256h | 56 03 00 00 56 62 25 00

256h = 0000 0256h = 0000 0000 0000 0000 0010 0101 0110

256 = 0000 0010h = 0000 0000 0000 0000 0000 0000 0000 0010 ⇒

⇒ 256h ^ 256 = 0000 0000 0000 0000 0010 0101 0110 = 00003256h

⇒ memory: 56 03 00 00

256256h = 00 25 62 56h ⇒ memory = 56 62 25 00

a9 dd $-a9 | 0000 0000

$-a9 = 0 (current offset)

a10 db 256, -255 | 00 01

256 = 2^8 = 1|0000 0000 = 00h (ou a byte)

-255 = 2's(255) -255 = 2's(255) = 1 (256-255)

255 = 1111 1111 = FFh

a11 dw 256h-256 | 56 00

256h = 2·256 + 5·16 + 6 = 512 + 80 + 6 = 598d

598 - 256 = 342d = 1·256 + 5·16 + 6·1 = 0156h ⇒ memory: 56 00

a12 dw 256-256h | AA FE

256h = 598 d

256 - 598 = -342

-342 = 2's(342)

342 = 0156h = 0000 0001 0101 0110 ⇒ 2's(342) = 1111 1110 1010 1010

= FEAA ⇒ memory : AA FE

a13 dw -256 | 00 FF

-256 = 2's(256)

256 = 0000 0001 0000 0000 ⇒ 2's(256) = 1111 1111 0000 0000 = FF00h

⇒ memory : 00 FF

a14 dw -256h | AA FD

-256h = - (0000 0010 0101 0110) = - 598
512 80 6

-598 = 2's (598)

598 = 0000 0010 0101 0110 ⇒ 2's(598) = 1111 1101 1010 1010

= FDAA ⇒ memory AA FD

342 : 16 = 21 (2)

342 : 16 = 2 (22)

22 : 16 = 1 (6)

1 : 16 = 0 (1)

342 = 1 5 6

598 = 2 5 6

342

Q15 Ab 2,5,6,25,6,2,56 | 02 05 06 19 06 02 38

25 = 19 h

56 = 38 h

1 a) CF, AH, SF, OF, ZF

a1) mov ah, 129
mov bh, 9Fh
add ah, bh

ah = 129 = 81h = 1000 0001
bh = 9F = 144 + 15 = 169 = 1001 1111
129 + 169 = 298

```
  11 111
 1000 0001 +
 1001 1111
------------
1 0010 0000
```

the result doesn't fit a byte (there's a transp. digit) ⇒
⇒ CF = 1
SF = 0 (MSB = 0)
OF = 1 (2 ~~positiv~~ neg numbers - starting with 1' addition provide a positive result, which is math. impossible
ZF = 0 (the result ≠ 0)
AH = 20h

a2) mov ax, 128
sar al, 4
imul ah

ax = 128 = 2^7 = 0000 0000 (ah) 1000 0000 (al)

sar al, 7 ⇒ 1111 1111 (complete with the sign bit)
= FFh = 255 = -1

imul ah ⇒ al (-1) * ah (0) ⇒ AX = 0
CF = OF = 0 (no overflow b*b = b)
SF = 0 (0)
ZF = 0 (~~the result is~~ 0
zero flag is not set after division/multiplication)

a3) mov ~~ax~~, ax, 256

mov bx, -1

add ah, bh

$ax = 256 = 2^8 = \overbrace{0000\ 0001}^{ah}\ 0000\ 0000$

$bx = -1 = \overbrace{1111\ 1111}^{bh}\ 1111\ 1111$

$-1 = 2's(1) = 1111\ 1111\ 1111\ 1111$

```
 1111 1111
 0000 0001
 1111 1111
-----------
1|0000 0000
```

$\Rightarrow$

- CF = 1 (there's a transp digit outside the ~~reg~~ saving reg.)
- OF = 0 (no overflow on signed, $0 \in [-128, 127]$)
- SF = 0
- ZF = 1

a4) mov ah 128/2

mov bh, 90h >>3

sub ah, bh

$128 = 1000\ 0000$
$2 = 0000\ 0010$ } (1) $\Rightarrow$ ah = 1000 0010

$90h = 1001\ 0000 \xrightarrow{>>3} 0001\ 0010 \Rightarrow bh = 0001\ 0010$

sub ah, bh

```
 1000 0010 -
 0001 0010
-----------
 0001 0000
```

- CF = 0 (there it is needed to borrow a digit so CF will be set to 1)
- OF = 1 (~~no overflow on signed $16 \in [-128, 127]$~~ neg - pos = pos)
- SF = 0
- ZF = 0

b1) movzx, operand1, operand2 movzx eax, al ; puts in eax the unsigned extension of al

- for this instruction both operands should be of different sizes.

in the first (register) one it is loaded the unsigned-extended ~~one, so the first one must be~~ content of the second one (reg/memory ~~add~~ location)

b2) cbw ; the implicit operands are al, ax

- for this instruction both implicit operands are of different sizes,

the first one (al) - which is a byte - is converted to a word (ax).

~~(b3)~~ xlat ; the implicit operands are al, table.)

b3) IT DOESN'T EXIST ! ⇒ THERE CANNOT BE 2 OPERANDS FROM MEMORY because there's only 1 [Mode R/M] which specifies the type of operand (Register or memory ~~one~~ location) in the instruction ~~assembly stru~~ address computation.

b4) movsb ~~/ lodsb~~

- loads into EDI the content of ESI

b5) ~~cmp op1, op2~~ movsb

EXAMEN 1 February 18, 2019

A flag is an indicator of 1 bit representation. The EFLAGS register ~~is a 32~~ shows a synthetic overview of (the effects made by the instructions performed) the execution of the instructions. For the 80x86 microprocessor, the EFLAGS register (status register) is a 32 bit register, but only 9 of them are used. The flags are: CF, PF, AF, ZF, SF, TF, IF, DF, OF. Their clarification may be structured on 2 categories:

- flags that ~~cannot~~ are implicitly set by the processor (that show a previous effect): CF, PF, AF, ZF, SF, OF
- flags that can be explicitly set by the programmer (for a future effect): CF, IF, DF, TF. But there is no direct access to TF, for the fact that an accidental set of this one can determine the stop of the execution, ~~so~~ this is why TF is a debugger flag (and why there are debuggers even step by step)

- CF (Carry Flag) - is the transport flag. It is set to 1 if in the last Performed Operation, there was a transport digit outside the representation domain, and 0 otherwise
  - this flag marks the unsigned overflow

```
 1 1 1 1   0 1 1 0  +
 1 1 1 1   1 1 1 1
-------------------
1 1 1 1 0 1 1 0 1
```

the carry flag will be set to 1 since the result doesn't fit ~~a byte~~ 1 byte.

- PF (Parity Flag) - it is set to 1 if the number of 1's in the binary representation of the Last Performed Operation is even, and set to 0 otherwise

ex: if the LPO = 11 00 11 00 ⇒ PF = 1

- ~~AF~~ AF (Auxiliary Flag) - it is set to 1 if in the last performed operation there was a transport digit from bit 3 to bit 4
  - if in an addition there was a carry from the low nibble to the high nibble or if in a subtraction there was a

borrowed from the high nibble to the low nibble

ex:
$$\begin{array}{r} {\scriptstyle 1}\ {\scriptstyle 1}\ \ {\scriptstyle 1} \\ 1000\ \ 1111 \\ 0000\ \ 1111 \\ \hline 1001\ \ 1010 \end{array}$$

- ZF (Zero Flag) - it is set to 1 if the result of the ALO = 0, and set to 0 otherwise

- SF (Sign Flag) - it stores the most sign bit of the ALO (sign bit)
  (1 - negative, 0 - positive)

- Trap Flag (TF) - is a debug flag
  - if set to 1 ⇒ stops after every execution
  - if set to 0 ⇒ let everything flow

- IF (Interruption Flag) - if set to 1 interrupts are allowed
  - critical section = section of code that cannot be interrupted
  - the time between the switch of IF = time critical section

- DF (Direction Flag) - ~~if set to 1~~ for operating with string instructions
  - if set to 1 ⇒ the string is parsed in descending order (end → start)
  - if set to 0 ⇒ the string is parsed in ascending order (start → end)

→ OF (Overflow Flag) - it marks the signed overflow
  - set to 1 if the ~~reps~~ binary rep. of the last prel op didn't fit the repr. domain, 0 otherwise

considered in signed interpretation

The instr that are most involved with the flag values are mul, idiv because these instructions

The conf. of EFLAGS can be modified by pushing the content onto the stack and restac the EFLAGS register (by popping the top of the stack)
POPF
The flags reacting to an overflow situation are CF, OF

CF - set to 1 if there's a transp. digit outside the rep. domain

OF - set to 1 if the res of LPO doesn't fit the internal rep cons. in signed interpretation or an operation is not providing a logical mathematical result

- when positive + positive = negative (0... + 0... = 1...) OF=1
- when neg + neg = positive (1... + 1... = 0) OF=1
- when ~~positive - negative~~ = neg - pos = pos
- when pos - neg = neg

There is more than one flag dealing with an overflow situation, because there are two interpretations of every representation (unsigned and signed) so there are 2 possible results which can be ~~accessed~~ interpreted based on how the following instructions may be reffering to. There are 2 diff int of the admissible internal repr., even though there are a certain nr. of values ~~in an n~~ for a data type (byte, word, ...) but since on unsigned the int $[0, 2^n - 1]$, n- nr of bits and in signed $[-2^{n-1}, 2^{n-1}]$ the overflow situation is different for every interpretation, so we need 2 flags.

- instructions operating with flags (CF, OF, AF, DF)

```
mov AH, 200 ; AH = C8
mov BH, 200 ; AH = C8
add AH, BH  ; in AH it will be tried to be add 200+200 = 400, but this
            ; doesn't fit the a byte, so instead AH = 144, which is an
            ; incorrect result generated because of the overflow, so in this
            ; case CF = 1

mov AH, 1010 0000b
mov BH, 1010 0000b
add AH, BH ; AH = 0100 0000 ; in this case, in signed interpretation, it can
           ; be seen that 2 negative nrs provide a portive
           ; result, which is mathem. incorrect, so OF = 1
           ; also 1010 0000b = -96d, -96 - 96 = -192 ∉ [-128, 127]

mov AH, 1010 0000b
mov BH, 1010 1000b
add AH, BH ; 1 1 0 1 0 0 0 0 ; AF = 1

std ; sets the DF = 1
```

→ The instructions for direct accessing the flags are:

```
stc → CF = 1
clc → CF = 0
cmc → CF = ¬CF

std → DF = 1
cld → DF = 0

sti → IF = 1
cli → IF = 0
```

a) x dw -256, 256h | 00FF 56 02

-256 ~~[illegible]~~ 2's(256)

~~256 = 0000 0001 0000 0000~~

256 = 0000 0001 0000 0000

2's = 1111 1111 0000 0000 = FF00h ⇒ memory 00 FF

256h = 0000 0010 0101 0110 = 0256h ⇒ memory 56 02

b) y dw 256|-256, 256h & 256 | 00 FF 00 00

256 = 0000 0001 0000 0000

-256 = 1111 1111 0000 0000

256|-256 = 1111 1111 0000 0000 = FF 00 ⇒ memory 00 FF

256h & 256

~~256h = 0000 0010 0110~~

256h = 0000 0010 0101 0110

256 = 0000 0001 0000 0000

256h & 256 = 0000 0000 0000 0000 = 0000h ⇒ memory ~~56 02~~ 00 00

c) z db $-z, y-x | 00 04

$-z = 0 ($ - current offset, - offset of z) ⇒ memory 00

y-x = 4 (4 bytes difference) ⇒ memory = 04

d) db 'y'-'x', 'y-x' | 01 ?? asci('-') 46

'y'-'x' = 1 (the diff between the ascii codes) ⇒ memory 01

'y-x' = ~~[illegible]~~ asci(y) asci('-') asci('x') ⇒ memory ?? asci('-') 46

asci('y') = 121 = ??h

asci('x') = 120 = 46h

e) a db 512 >> 2, -512 << 2 / 80 00

$512 = 2^9$ = doesn't fit a byte ⇒ rep. on a word = 0000 0010 0000 0000
$(2^9 : 2^2) = 2^7$
⇒ 0000 0000 1000 0000 = 0080 ⇒ byte 80 ⇒ memory 80

$-512 = -2^9$ ⇒ rep. on a word = 1111 1110 0000 0000

-512 << 2 = 1111 1000 0000 0000 = F8 00 ⇒ memory 00

f) b dw z-a, !(z-a) / FA FF 00 00

z - a = 10 - 16 = -6 (-nr. of bytes between z and a)

-6 = 2's(6) = FFFA h ⇒ memory: FA FF

6 = 0000 0000 0000 0110

-6 = 1111 1111 1111 1010 ~~⇒ !(-6) = 0000 0000 0000 0101 = 0006 h~~

~~⇒ memory 06 00~~ !(-6) = 0 = 0000 h ⇒ memory 0000

g) c dd ($-b)+(d-$), $-2*y+3 syntax error
$ sits pe loc

$-b = 4
d-$ = 4 } ⇒ ~~memory~~ 00000008 ⇒ 0000 0800

~~$-2*y+3~~

~~2*y = an unknown address~~ } ⇒ ~~memory violation error (probably)~~

2*y is not allowed ⇒ syntax error

h) d db -128, 128^(~128) / 80 FF

-128 = 2's(128)

128 = 2^7 = 1000 0000 = 80 h

-128 = 1000 0000 = 80 h ⇒ memory: 80

128^(~128)

~128 = ~(1000 0000) = 0111 1111
128 = 1000 0000 } ^ ⇒ 1111 1111 = FF h ⇒ memory FF

a) e times 2 resw 6

⇒ memory 00 00 00 00 00 00 00 00 00 00 00 00 ×2
(1) (2) (3) (4) (5)

f) times 2 dd 1234h, 5678h

⇒ memory 34 12 00 00 78 56 00 00 34 12 00 00 78 56 00 00

g) mov bh, 7Fh ; 0111 1111 = 127
cmp bh, al ; fictive subtraction that sets the flags { CF=1, ah starts with 1 - neg ; CF=0, ah starts with 0 - pos
ror ah, 1 ; first bit goes on the last position
sar ah, 7 ; fill with the sign bit
ah = 1000 0000 ⇒ ~~0100 0000~~ ~~0010 00~~ 0000 0001, CF = sign bit
so this instructions

b) mov bh, 7Fh ; bh = 0111 1111
cmp bh, al ; fictive subtraction : bh - al = 0111 1111 - al ⇒ ~~first bit~~ last bit of
ror ah, 1 ; last bit of ah = sign bit (bit 0101 ...) al = sign bit ⇒ will set { CF=1, al starts with 1 ; CF=0, al starts with 0
sar ah, 7 ; fill with the sign bit (bit bit bit ...)
so these instructions convert al into a word for signed interpretation
movsx ax, al OR CBW

↓
loads in AX the signed-extended AL

ax = 0000 0010

dx = 1111 1111

1 1111 11
0101 0110 = 86
1111 1111 = -1
1 0101 0101

ah

256
16
1536 3
256
4096

16
6
96 3

256
5
1280 +2

4096
1280
5376

16
5
80 3

96
5472

16
15
80 3

16
240

86
326

A
56
FF
55

21 : 16 = 1 R 5

21 = F6

0000 0000 1111 0110

0111 1011 0000 0000

ah al

10h = 0001 0000

bh = 0000 0011 / 0001 0011

a) mov eax, 200
mov ebx, 254h
idiv bl

200 = C8h = 1100 1000
signed(200) = 200
unsigned(200) = 200 › 200 fits a dword

254h = 4 + 5·16 + 2·256 = 4 + 80 + 512 = 596
signed(254h) = 596d
unsigned(254h) = 596d › 596 fits a dword

bl = 54h = 84d
idiv bl ⇒ 200 : 84 = 2 rest 32 / AX = 2 = 0002h
\ DX = 32 = 0020h

CF and OF are not modified

b) mov ax, 256h
mov dx, -1
add ah, dh

256h = 598d
⇒ unsigned(256h) = 598d
signed(256h) = 598d › 598 fits a word
(598 ∈ [0, 2^{16} − 1]
598 ∈ [−2^{16}, 2^{16} − 1]) ⇒ ~~ah = 56h~~ ~~= 86d~~ ah = 02h = 2d

−1 = 2's(1)
2's(0000 0000 0000 0001) = 1111 1111 1111 1111 = FFFFh
⇒ unsigned(−1) = (FFFFh) = 2^{16} − 1 = 65535d ⇒ dh = FFh = 255
signed(FFFFh) = (−1)d
unsigned addition : add ah, dh ; ah ← 2 + 255 = 254 ∉ [0, 255] ⇒
⇒ CF = 1

```
  1111 11
  0000 0000 +
  1111 1111
1|0000 0001
```

$\Rightarrow CF = 1$ (transp. dig. outside)

$OF = 0$ $(2 + (-1) = 1 \in [-127, 127]$

$\Rightarrow ah = 01h = 1d$

c) mov ax, ~(16h | 32)

$16h = 0001\ 0110b$

$32 = 0010\ 0000b$ (2^5) $\Rightarrow 16h | 32 = 0011\ 0110b$

$\Rightarrow \sim(16h | 32) = 1100\ 1001b$

$\Rightarrow ax = 1100\ 1001 = C9h$ ~~= 201d signed and unsigned~~

$FFC9h = 65481$ (unsigned)
$= -54$ signed

mov bx, 2000h >> 4

$2000h = 0010\ 0000\ 0000\ 0000 = \underbrace{0000\ 0010}_{bh}\ 0000\ 0000 = 0200h$

~~imul bh~~

imul bh ; DX:AX ← AX * BH (signed)

AX * BH = 201 * 2 = 402d (fits a word) $\Rightarrow CF = OF = 0$

402d =

mul bh

al = C9h = 201d = -55d sig.

bh = 02h = 2d

signed : -55 * 2 = ~~-108~~ -110d = 6535 - 110 = 65426 unsigned

unsigned : 201 * 2 = 402

$CF = OF = 0$ (b * b → b)

d) mov ax, 21 << 4

mov bh, 10h ^ 3

sub bh, al

ax = 21 = 0000 0000 0101 b = 0015 h

0000 0000 0001 0101

~~21 << 4 = 0000 1010 1000 0000 = 0~~

21<<7 = 0000 1010 1000 0000 = 0A80h = 8·16 + 10·256 = 2560 + 128 = 2688 d

al (signed and unsigned)

10h = 0001 0000 b

3 = 0000 0011 b } 10h ^ 3 = 0001 0011 = 13 h = 19 d

(signed and unsigned)

⇒ bh = 13h = 19 d

al = 1000 0000 = 2^7 = 128 d (unsigned)

−127 d (signed)

unsigned : sub bh, al ; bh = 19 − 128 (+256) = 147 d (unsigned)

signed : sub bh, al ; bh = 19 − (−127) = 146 d (signed) ⇒ CF = 1

146 ∉ [−128, 127] ⇒ OF = 1

e) shl bh, 8

add bx, ax

mov eax, ebx (=) lea eax, [ebx]

EXAM - 2022

I Call code, Entry code, Exit code

CDECL, STDCALL, ASM-ASM, ASM-C

II.

a1 dd '0abcdefh', 0abcdefh

'0abcdefh' = '0' - as a dword ⇒ 30h = 00 00 00 30 ⇒ mem: 30 00 00 00

'a' - as a dword ⇒

'a' = 97 = 61h ⇒ 00 00 00 61 ⇒ mem 61 00 00 00

'b' = 98 = 62h ⇒ 00 00 00 62 ⇒

'c' = 99

⋮

'f' = 102 = 66h ⇒ 00 00 00 66

'h' = 104 = 68h ⇒ 00 00 00 68

⇒ 30 00 00 00 | 61 00 00 00 | 62 00 00 00 | 63 00 00 00 | 64 00 00 00 | 65 00 00 00 | 66 00 00 00 | 68 00 00 00 | EF CD AB 00

'0' 'a' 'b' 'c' 'd' 'e' 'f' 'h'

0abcdefh ⇒ memory EF CD AB 00

00abcdef

a2 dw '0abcdefh', 316

'0abcdefh' = 0030 0061 0062 0063 0064 0065 0066 0068

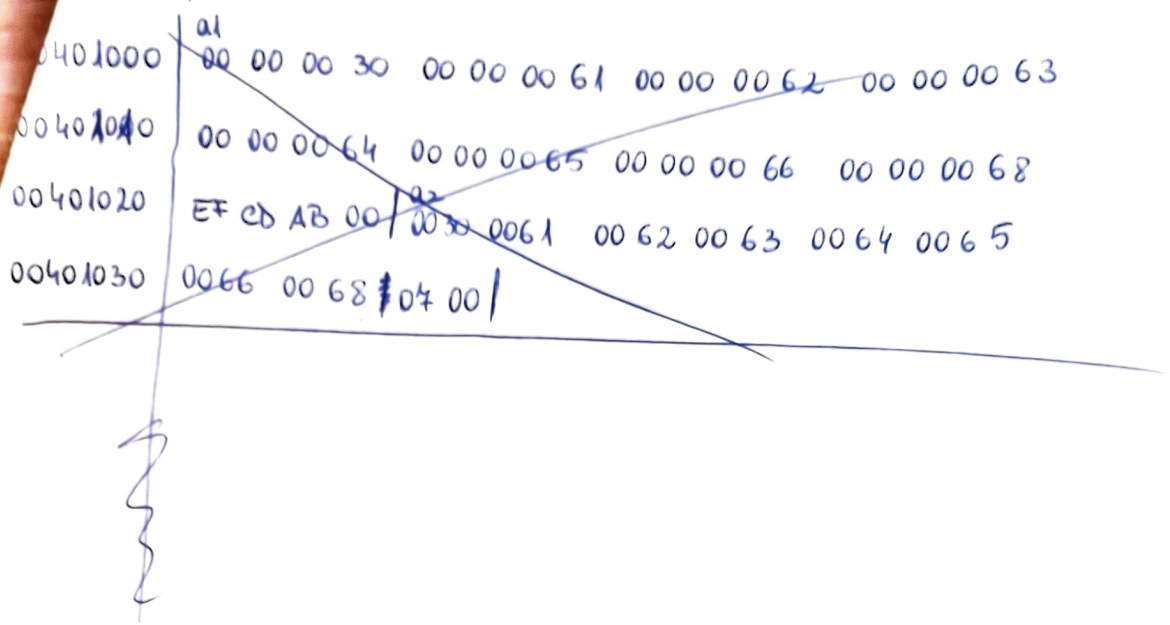
3 = 0000 0011

6 = 0000 0110 } ⇒ 316 = 0000 0111 = 013Ch 00 07h ⇒ mem: 04 00

a3 dw $-a2, a2-a1

$-a2 = 18 (18 bytes between $ and a2)

MEMORY LAYOUT



a6 dd $+a2-1, a2 ; syntax error addition of addresses not allowed

a7 dd 256h ^256, 256256h | 56 03 00 00 56 62 25 00

256h = 0000 0256 h = 0000 0000 0000 0000 0000 0010 0101 0110 ⇒

256 = 0000 0100 h = 0000 0000 0000 0000 0000 0001 0000 0000

⇒ 256h ^256 = 5*(0000) 0011 0101 0110 = 00000 356 h ⇒ mem: 56 03 00 00

256256h = 00256256 h ⇒ mem: 56 62 25 00

a8 dd ($-a7)+(a9-$), -256 | 10 00 00 00 00 FF FF FF

$-a7 = 8 (bytes)

8 = 0000 0008 h

a9-$ = 8 (bytes)

($-a7)+(a9-$) = 16 = 0000 0010 h ⇒ mem: 10 00 0000

-256 = 2's(256)

256 = 0000 0001 0000 0000 0000 0000 0000 0001 0000 0000 ⇒
(2^3)

⇒ -256 = FFFFFF00 h ⇒ mem: 00 FF FF FF

⇒ mem: 00 FF FF FF

II. a1 dd 'aabcdefh', 0abcdefh | 30 61 62 63 64 65 66 68 EF CD AB 00

'a' = 97 = 61h

0abcdefh = 00AB CD EF

a2 dw 'aabcdefh', 3|6 | 30 61 62 63 64 65 66 68 07 00

3 = 0000 0000 0000 0011
6 = 0000 0000 0000 0110
⇒ 3|6 = 0000 0000 0000 0111 = 0007h

⇒ mem: 07 00

a3 dw $-a2, a2-a1 | 0A 00 0C 00

$-a2 = 10 bytes = 000Ah ⇒ mem: 0A 00

a2-a1 = 12 bytes = 000Ch ⇒ mem: 0C 00

a4 db 129 >> 1, -129 << 1 | 40 FE

129 >> 1, 129 = 1000 0001 ⇒ 129 >> 1 = 0100 0000 = 40h

-129 = 2's(129) (unsigned ext.) = 2's(0000 0000 1000 0001) = 1111 1111 0111 1111 = FF7Fh

= 7Fh

0111 1111 << 1 = 1111 1110 = FEh

a5 dw a2-a4, ~(a2-a4) | F2 FF 0D 00

a2-a4 = -14 (14 bytes between

-14 = 2's(14)

14 = 0000 0000 0000 1110 = 000Eh ⇒ mem: 0E 00

~(a2-a4) = ~(

⇒ -14 = 1111 1111 1111 0010 = FFF2h ⇒ mem: F2 FF

~(a2-a4) = ~(1111 1111 1111 0010) = 0000 0000 0000 1101 = 000Dh

⇒ mem: 0D 00

2b)
```
push ebp
mov ebp, esp
pop eax ; mov eax, ebp
shl eax, 1 ; eax = eax * 2
sub eax, 5 ← mov ebx, esp
add ebx, eax ; ebx = eax - 5 = ebp <<< 1 - 5   ebx = esp + eax = esp + 2
mov esp, ebp
```

234 : 16 = 14
16
= 74
64
10

1002 : 256 = 3
768
234

16
14
64 2
16
224

III. a)
```
mov ax, 0100h
mov bx, 1000+1Ob
idiv bl
```

2^9 = 512
1002
512
490
256
134
128
6

256 : 58 = 4
232
= 24

256 : 198 = 1
198
58

256
234
22

0100h = 256 d (signed and unsigned)

1000+1Ob = 1000 + 2 = 1002 = 0000 0001 1100 0110 = 6 + 64 + 128 = 198 d (unsigned)
bl

03EAh ⇒ bl = EA = 234 d (unsigned)
= -22 d (signed)

-2's(198) = -58

2's(198) = 0011 1010 = 10 + 16 + 32 = 58

signed : idiv bl ⇒ quotient in al, remainder in ah

256 : (-58) = -4 rest 24, al = -4 = 1111 1100 = FCh

4 = 0000 0100

ah = 24 = 18h

unsigned idiv bl

256 : 198 = 1 r 58 ⇒ al = 1 = 01h
ah = 58 = 3Ah

256 : 22 = 11 r 2
22
36
22
14

signed idiv bl : 256 : (-22) = -11 r E (14)
al = -11
ah = 14 = E

unsigned idiv bl 256 : 234 = 1 r 22
al ah

MEMORY LAYOUT

~~00401000~~

00401000	a1 30	61	62	63	64	65	66	68	EF	CD	AB	00	a2 30	61	62	63
00401010	64	65	66	68	04	00	a3 0A	00	0e	00	a4 40	FE	a5 ~~FB~~	~~FC~~	0D	00
00401020	a* 56	03	00	00	56	62	25	00	a8 0e	00	00	00	00	FF	FF	FF
00401030	a9 01	FF	80	FF	28	10	80	FF	28	a6	80	FF	28	a6	80	FF
	28	a6	80	FF	~~a11~~	~~16~~	10									

a9 dw -255, -128 | 01 FF 80 FF

-255 = 2's(255)

~~255 = 0000 0004 0000~~

255 = 0000 0000 1111 1111 ⇒ -255 = 1111 1111 0000 0001 = FF01 h

⇒ mem: 01 FF

-128 = 2's(128)

128 = 0000 0000 1000 0000 ⇒ -128 = 1111 1111 1000 0000 = FF80 h

⇒ mem: 80 FF

a10 times 4 dw 128 h, -128 | 28 a6 80 FF 28 a6 80 FF 28 a6 80 FF

128 h = 0128 h ⇒ mem: 28 a6

-128 = 2's(128) = FF80 h ⇒ mem: 80 FF

a11 db a3 ~~+1~~ syntax error, offsets not step on 2bits.

~~a3 = 0040101A ⇒ last byte = 1A h~~

a12 dw a3 | ~~16~~ 10

a3 = 0040101~~6~~ ⇒ last word 1017 h ⇒ mem: ~~16~~ 10

b) mov ah, 0edh
mov al, 0ebh
add ah, al

ah = ~~0ed~~ ED = 192 d (unsigned) = 1100 1101 (+13 = 205)
~~1100 1101~~ = ~~-64~~ d (signed)
256 - 192 = 64 (-51)
al = EB = 224 d (unsigned) = 1110 1011 (256 + 11 = 235)
= -32 d (signed)

add ah, al
1100 1101 +
1110 1011
1 | 1011 1000 ⇒ ah = 1011 1000 = B8h = 176 + 8 = 184 (unsigned)
= -42 (signed)

CF = 1 (a transp. digit outside...)
OF = 0 (no overflow on signed)

c) mov ax, 1010h
mov bx, 1111b
mul bl

1010h = ⇒ al = 10h = 16
1111b = 15 d
mul bl ⇒ al * bl = 16 * 15 = 240 (result in ax) = ~~F0h~~ 00F0h

CF = OF = 0 (b * b = b) no overflow on multiplication
{ al = F0h = ~~1111 0000~~ (-16 signed)
{ ah = 00h

d) mov dh, 200
mov ch, 62h
sub dh, ch

200 = C8h = 1100 1000b
62h = 96 + 2 = 98d = 0110 0010b
sub dh, ah ⇒ 1100 1000 −
0110 0010
0010 0010 ⇒
= 34 (sub + unsigned)
{ CF = 0 (no transp. dig.)
{ OF = 1 (neg - pos = pos)

push ebp

mov ebp, esp ; mov ebp, ebp

pop eax ; mov eax, ebp

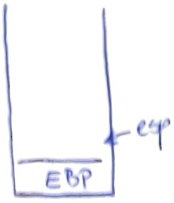
shl eax, 1 ; eax ← 2* eax

sub eax, 5 ; eax ← 2*eax - 5

mov ebx, esp ; ebx ← esp point to ebp ebx ← ebp

add ebx, eax ; ebx ← ebp + eax = ebp + 2*eax - 5

mov esp, ebp



(=) lea ebx, [esp + 2*eax - 5]

I. EXAM 2022 - feb

00401000	99 FF FF FF 34 10 40 00
00401010	
00401020	

II. a1 dd -103, a6

$-103 = 2's(103)$

00401000 + 36

00401010 → 00401020 → 00401030 → 00401034

~~103 = 67h = 0000 0000 0000 0000 0110 0111 ⇒ -103 = 1111 * 4 1001 1001 = FFFF FF 99h~~

103 = 00 00 00 67 h = FF FF FF 99h ⇒ mem: 99 FF FF FF

offset(a6) = 00401034 ⇒ memory: 34 10 40 00

a2 dw 1&2, 3|4, 5^6, 1|2, 3^4, 5&6

1 = 0000 0000 0000 0001
2 = 0000 0000 0000 0010
⇒ 1&2 = 00...0 = 0000h

3 = 0000 0000 0000 0011
4 = 0000 0000 0000 0100
⇒ 3|4 = 0000 0000 0000 0111 = 0007h ⇒

⇒ mem: 07 00

5 = 0000 0000 0000 0101
6 = 0000 0000 0000 0110
⇒ 5^6 = 000...0011 = 0003h ⇒ mem: 03 00

1 = 0000 0000 0000 0001
2 = 0000 0000 0000 0010
⇒ 1|2 = 0003h ⇒ mem: 03 00

3 = 0000 0000 0000 0011
4 = 0000 0000 0000 0100
⇒ 3^4 = 0003h ⇒ mem: 03 00

5 =

6 =

I a) segment data use32 class=data

little endian ⇒ 00 FF 56 02

x dw -256, 256h ; FF00 0256 ~~offset(x) = 0040 1000~~

y dw 256|-256, 256h ~~& 256~~ ; FF00 0000, ~~offset(y) = 0040 1008~~

00 FF 00 00

x, y

(5602) = 4

z db $-z, y-x ; 00 02, ~~offset(z) = 0040 1010 (?)~~

↳ ~~2 bytes~~ dif distanta adr(x) - adr(y) = 4 bits = 2 bytes

db 'y'-'x', 'y-x' ; 01 ~~FF~~ (code of '-') 70, offset = 0040 1012

a db 512 >> 2, -512 << 2 ; 08 80, offset(a) = 0040 1015

b dw z-a, !(z-a) ; (Syntax error, because ! operator can)

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

d db -128, 128 ^ (-128) ; 00 00

e ~~twice~~ 2 resw 6 ; 06 06

twice 2 dd 1234h, 5678h ; 3412 7856

(b) mov BH, 4Fh

cmp BH, AL

ror AH, 1

sar AH, 4

(Vreau să pun val din carry în AH)

256 − 60 = 196

200 = C8

400 − 256 = 144

32

200 16 = 12 = C

0011 0010
0011 0010

200 16 = 12
16 / 40 C8
32 / 8

16 / 5 / 80 3

1100 1000
1100 1000
1] 1001 0000

1010 0000
1010 0000
1] 0100 0000

$2^5 = 32$, $2^7 = 128$ } 256 − 160 = 96

III. a) mov eax, 200
mov edx, 254h
idiv bl ; the result will be stored in AX ~~= 100 = 64h~~ = 50d = 32h
(quotient)

and remainder in DX = 0
CF = OF = 0 (the res. fit on a word)

1111 1 1
0101 0110
1111 1111
1] 0101 0100
2^6 2^4 2^2
64 16 4

b) mov ax, 256h ; ax = ~~0101 0110~~ 0000 0010 0101 0110
mov dx, -1 ; dx = 1111 1111 1111 1111
add ah, dh ; ah = ~~0101~~ 0101 0100 CF = 1, OF = 0

---

b) mov ax, 256h ; ax = 0000 0010 0101 0110b =
mov dx, -1 ; dx = 1111 1111 1111 1111b = FFFFh
add ah, dh ; ax = 0101 0101b = 99h =
unsigned : 256h = 5442 ~~= 86h~~ ah = 56h = 86d
unsigned : FFFFh = −1 ⇒ dh = FFh = ~~255d~~ 255

56h + FFh = 86d + 255d = ~~341~~ = 155h doesn't fit byte AH ⇒
⇒ AH = 55h and CF = 1 (because there's a transport digit outside the byte rep)

signed : 256h, ah = 56h = 86d

signed : FFFFh, dh = FFh = -1d

~~56h + FFh = 86d + (-1)d = 85d =~~

56h + FFh = 155h = 34d

AH = 55 (0101 0101) = 85d $\in [-128, 124] \Rightarrow$ OF = 0 (no overflow on signed interpretation)

d) mov ax, 21 << 4

mov bh, 10h ^ 3

sub bh, al

21 << 4 ; F6 << 4 ; 0000 0000 1111 0110 << 4 $\Rightarrow$ 0111 1011 0000 0000
unsigned

$\Rightarrow$ al = 0000 0000 = 00h

10h ^ 3 ($\Rightarrow$ 0001 0000 ^
0000 0011) $\Rightarrow$ bh = 0001 0011 = 13h
0001 0011

sub bh, al $\Rightarrow$ 0001 0011 -
0000 0000
0001 0011 = 13h $\Rightarrow$ CF = 0 (no transp digit, res fit)
OF = 0 (res fit)

signed - the same

c) mov ax, ~(16h | 32)

mov bx, 2000h >> 4

imul bh

16h = 0000 0000 0001 0110
32 = 0000 0000 0010 0000
$\Rightarrow$ (1) $\Rightarrow$ 0000 0000 0011 0110 $\Rightarrow$ ~
0 0 36h

1111 1111 1100 1001b = FFC9h $\Rightarrow$ ax = FFC9h = (signed)

2000h >> 4 $\Rightarrow$ 0200h $\Rightarrow$ bx = 0200h $\Rightarrow$ bh = 02h = 2d

2's(1111 1111 1100 1001) = 0000 0000 0011 0111 = 55d $\Rightarrow$ FFC9h = -55d $\Rightarrow$
$\Rightarrow$ ax = -55

dx ax = ax * bh = -55 * 2 = -110d

CF = OF = 0

e) shl bh, 8

add bx, ax

mov eax, ebx / movzx eax, bx (loads in eax the unsigned version of bx)

```
bits 32

global start

extern exit, printf, scanf

import exit msvcrt.dll

import printf msvcrt.dll

import scanf msvcrt.dll

segment data use32 class = data

    string times 100 db 0

    number dd 0

    formats db "%s", 0

    formatnr db "%d", 0

segment code use32 class = code

    start:

        push dword number
        push dword [formatnr]      } read nr
        call [scanf]
        add esp, 4*2

        push dword [number]
        push dword [formatnr]      } print nr
        call [printf]
        add esp 4*2

    loop_read:
            push dword number
            push dword [formatnr]
            call [scanf]
            add esp, 4*2

            cmp dword [number], 0
            je end_progr
            jmp loop_read
```

200 = C8h

eax = 0000 0000 1100 1000

signed(eax) = 0000 0000 1100 1000

ebx = 0000 0010 0101 0100

signed(bl) = 0100 = 4

0000 0001

1111 1111