Seminar 913 (s1 13:00-14:50)

Thursday, October 14, 2021 11:07 AM

IA -> 2 related to signed/unsigned representations of nr. there are 2 classes of instructions

1) -> instr. which do not care about signed or unsigned representation of nr.: mov, inc, dec, add, sub

$(17)_{10}$ -> byte $(0001\ 0001)_2$

$(-17)_{10}$ -> $(1110\ 1111)$

17:
0001 0001
1110 1111 CF
0000 0000 1

2) -> instr. which interpret their operands as unsigned nr: MUL, DIV

3) -> — // — // — // — signed nr: IMUL, IDIV, CBW, CWD, CWDE

int a = 2;

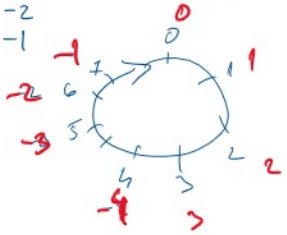
n bits
- unsigned repr. $[0, 2^n - 1]$
- signed repr $[-2^{n-1}, 2^{n-1} - 1]$

n = 3

	unsigned int.	signed int.
000	0	0
001	1	1
010	2	2
011	3	3
100	4	-4
101	5	-3
110	6	-2
111	7	-1

$[-2^{3-1}, 2^{3-1} - 1]$

$[-4, 3]$



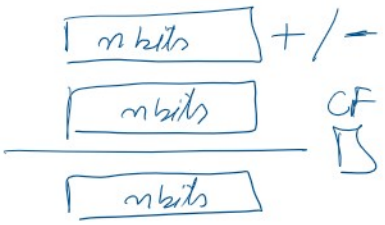
a, b - word

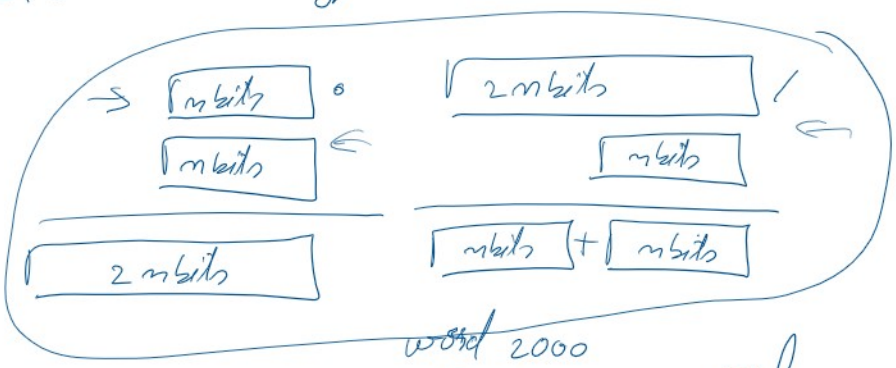
~~add [a], [b]~~

mov ax, [a]
add ax, [b]

important rules that must be obeyed by instr. with 2 operands

-> all operands must have the same size/type <-





word 2000

byte 1

mul

-> at least one of the operands must be a register or a constant, if it is a constant this constant can not appear as a destination operand

a db 2

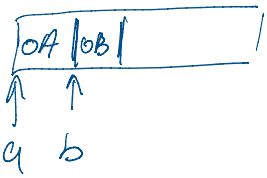
mov al, 2 ; AL = 2

~~mov 2, al~~ ;

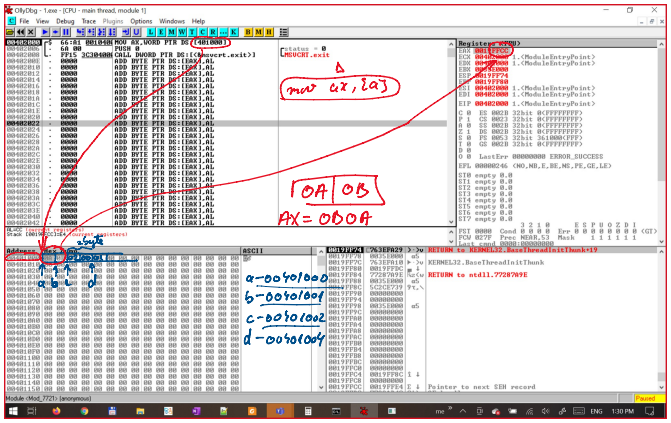
add [a], 2

~~add [a], [a]~~

```
a   db  10
b   db  11
c   dw  2
d   db  ?
```



mov ax, [a] ; AX = 0B0Ah



little endian

big endian

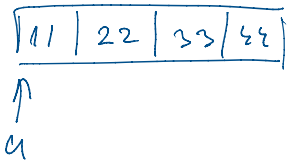
base 10 — 1234 — big endian in base 10

4321 — little endian in base 10

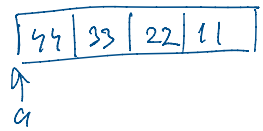
IA: little endian

a dd 11223344h

Big endian



IA (little endian)



MUL , DIV , IMUL , IDIV

MUL source ✓ MUL dest, source ✗

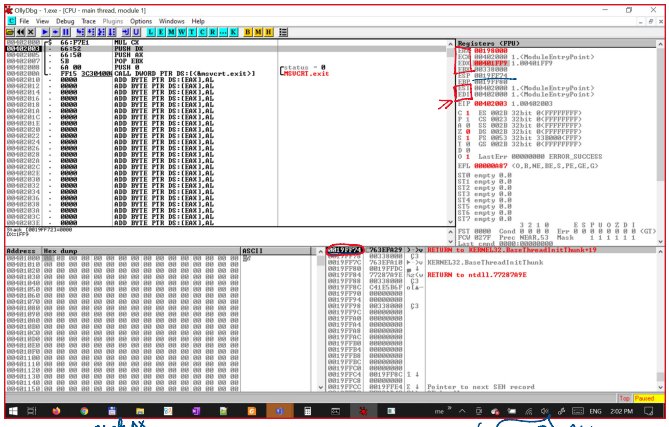
~~mul [a],~~

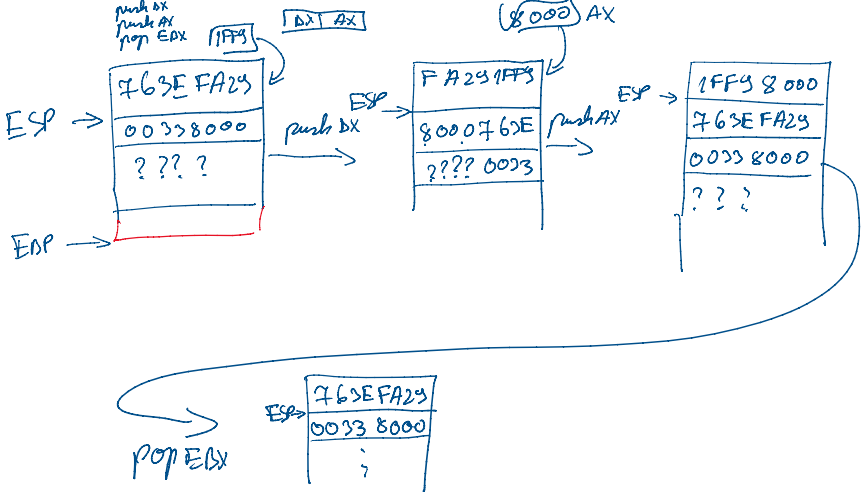
source
- byte — AX ← AL · source
- word — DX:AX ← AX · source
- dword — EDX:EAX ← EAX · source

CX = 2000h

AX = FFCC

mul cx ; DX:AX = 1FF9 8000 → EDX

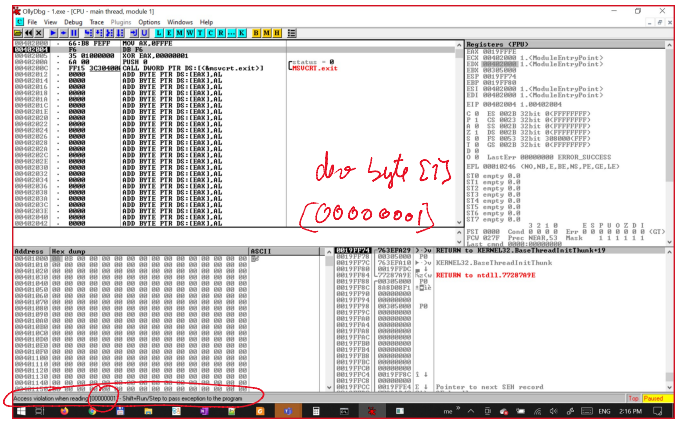




EDX = 1FF98000h

DIV source IDIV source

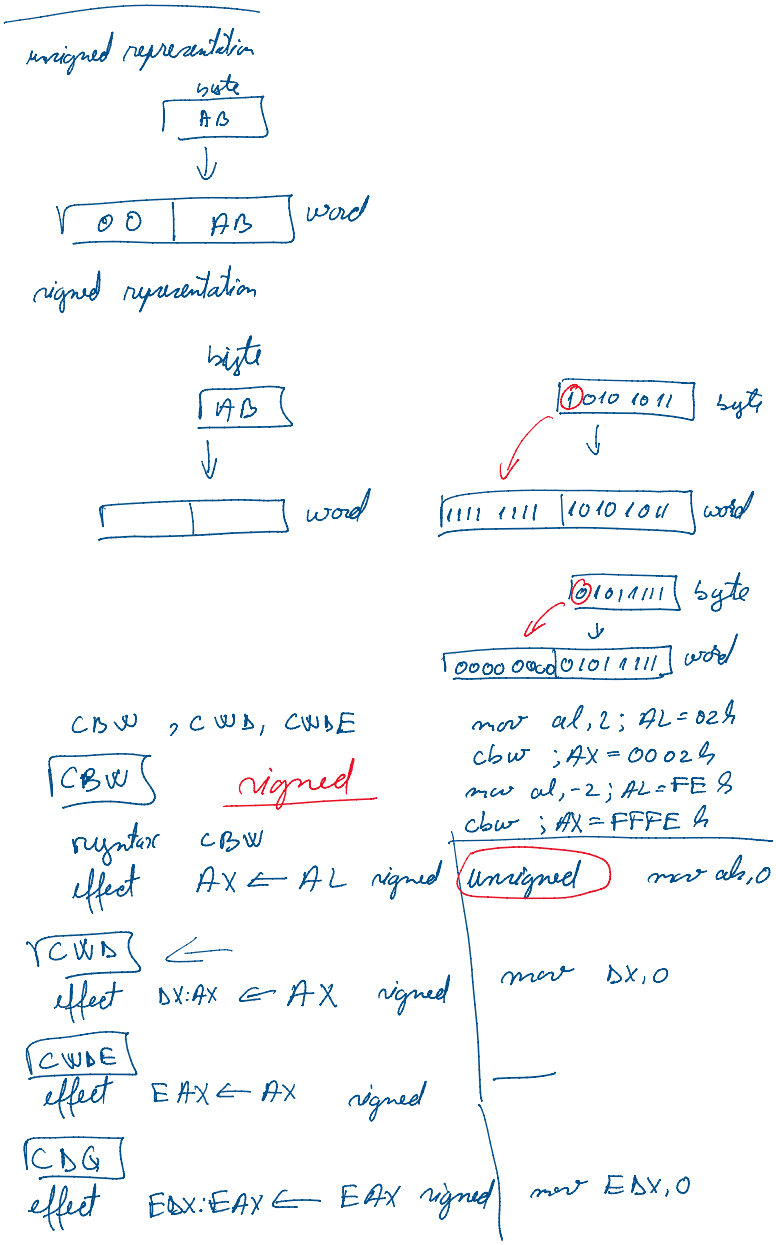
mov cx, -2
div 1
div byte (1)



mov ax, -2
mov bl, 1

```
mov ax, -2
mov bl, 1
div bl ; AH = ax % bl , AL = ax / bl    65534 / 1 = AL
                                  AL 8 biti [0, 255]
```

idiv bl



CBW , CWD, CWDE

CBW — signed

syntax CBW

effect AX ← AL signed

CWD

effect DX:AX ← AX signed

CWDE

effect EAX ← AX signed

CDQ

effect EDX:EAX ← EAX signed

```
mov al, 2 ; AL = 02h
cbw ; AX = 0002h
mov al, -2 ; AL = FEh
cbw ; AX = FFFEh
```

unsigned

```
mov ah, 0
mov DX, 0
mov EDX, 0
```

$x = \frac{a \cdot b}{d} - c$ a, c, d - bytes; b - word

signed representation / interpretation

- data

```
a   db  -2
b   dw  4
c   db  2
d   db  1
```

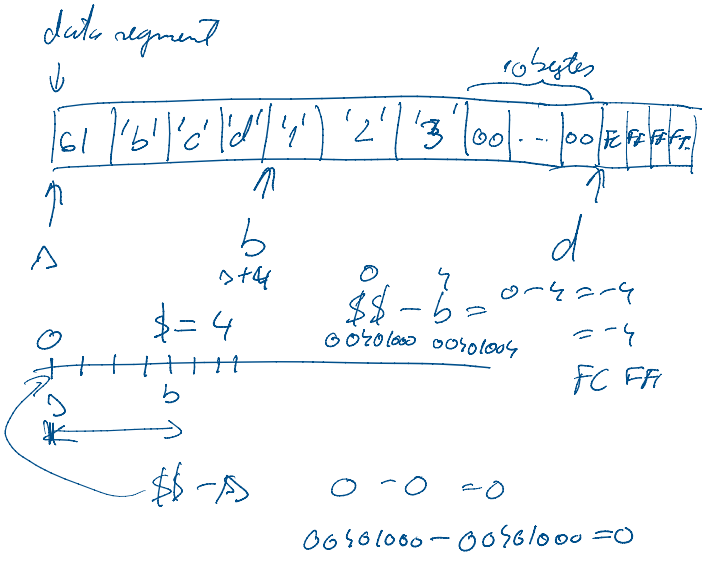
```
x  dw 0
.code
start:
    mov al, [a] ; AL = a = -2
    cbw         ; AX = -2
    imul word [b] ; DX:AX = AX * B = -2 * 4 = -8
                  ; d-byte -> word
    mov bx, ax  ; DX:BX = AX - b
    mov al, [d]
    cbw         ; AX = d = 1
    mov cx, ax
    mov ax, bx
    idiv cx     ; AX = DX:AX / CX ; DX = (DX:AX) % CX
    mov bx, ax
    mov al, [c]
    cbw
    sub bx, ax
    mov [x], bx
```

Seminar 3

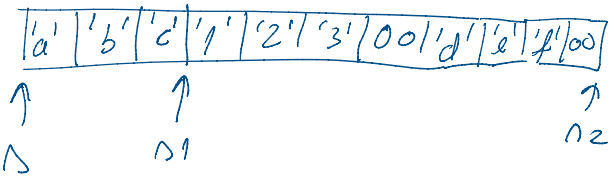
```
.data
    a   db   'abcd'
    l_a equ  $-a
    l_aa equ $$-a
    b   db   '123'
    resb 10
    d   dd   $$-b
```

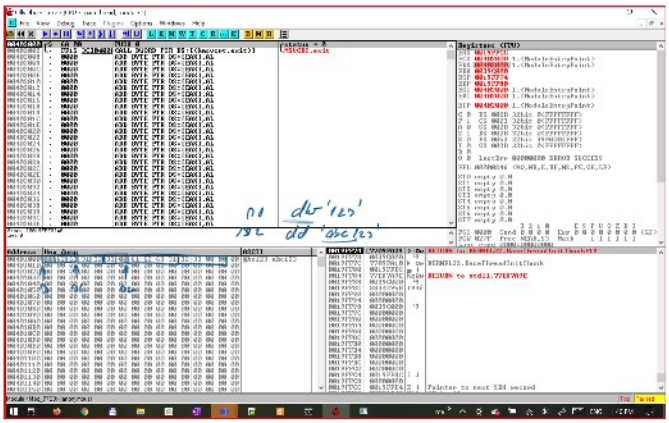
→ `times 10 db 0`

`times 10 db '0'`



```
a   db   'abc'
a1  dw   '12', 'def'
a2  dd   'abc123'
```





if ( ) { }

else { }

Cmp b,s ; b - s => EFLAGS

a , b

Cmp a,b

unsigned interpretation of a,b		signed interpretation of a,b	
a > b	ja label ; (jump if above/jump to label if a>b) CF,ZF	jg label	
a < b	jb label (jump if below/ jump to label if a<b)	jl label	
a ≥ b	jae label	jge label	
a ≤ b	jbe label	jle label	
a == b	je label	je label	ZF=1
		jnl jng . .	

jnae

jnb . . .

if (x < a) { }

x += a;

else {

x -= a;

}

.data

x, a words

.code

start:

~~cmp x, a~~

~~cmp [x], [a]~~

mov ax, [x]

cmp ax, [a]

, unsigned int.

jb label

Cmp a,b

j<u>n</u> condition label

jmp end_if

label:

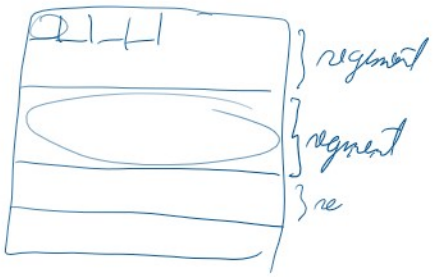
end_if:

, unsigned int.
jb label
; x -= a
jmp end_if
label:
; x += a
end_if:

) else

end_if:

unconditional jump instr
jmp label



working with strings
memory addresses and offsets

segment_selector : offset
16bit 32bit
32bit

offset = [base] + ([index] * [scale]) + [displacement]

$$offset = \begin{bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ ESP \\ EBP \\ EDI \\ ESI \end{bmatrix} + \left(\begin{bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ EDI \\ ESI \end{bmatrix} * \begin{pmatrix} 1 \\ 2 \\ 4 \\ 8 \end{pmatrix} \right) + \begin{bmatrix} None \\ 8\text{-bit} \\ 16\text{-bit} \\ 32\text{-bit} \end{bmatrix}$$

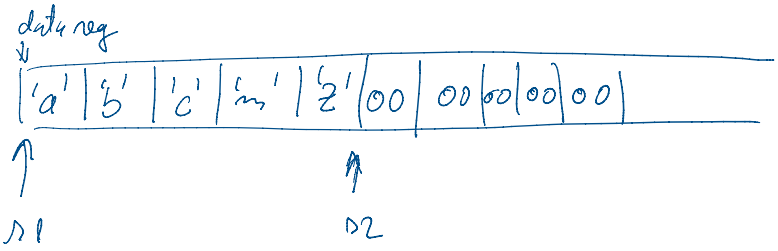
ex 'abc' ⟶ 'ABC'
D1 D2

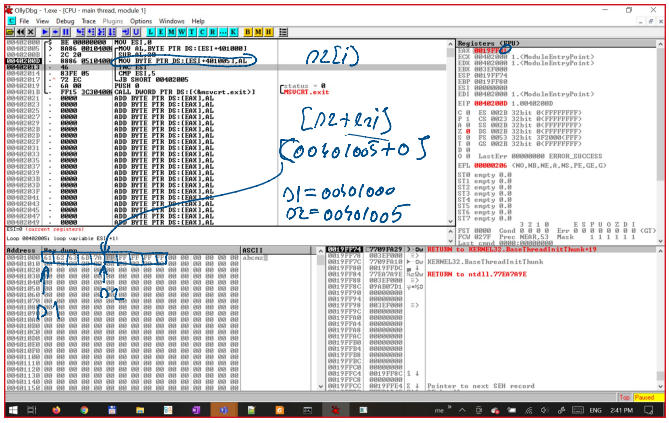
- data
D1 db 'abcmz'
l_D1 equ $-D1 ; l_D1 db $-D1

l_D1 = 5

l_n1 equ $-n1 ; (l_n1) db $-n1

n2 times l_n1 db 0





Seminar 4 string instructions

string → length of the string
→ type of each elem.
→ pointer to the first / last elem. of the string
→ parsing direction (→ , ←)

ECX
—
ESI, EDI
DF

string instructions for data transfer

LODS_ , STOS_ , MOVS_ — {B, W, D}

LODSW (load string of words)

AX ← <DS:ESI>

DF: 0 → ESI ← ESI+2
1 → ESI ← ESI-2

STOSD (store string of doublewords)

<ES:EDI> ← EAX

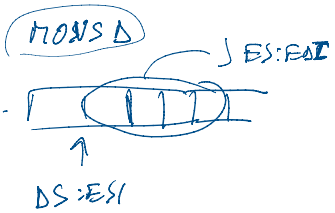
if DF: 0 → EDI ← EDI+4
1 → EDI ← EDI-4

MOVSD — [ES:EDI]

4 B: -

1 ESI ← ESI - 4

MOVSB (move string of bytes)



<ES:EDI> ← <DS:ESI>

DF { 0 => ESI ← ESI+1 , EDI ← EDI+1
   { 1 => ESI ← ESI-1 , EDI ← EDI-1

DS ESI
ES EDI

---

string instructions for comparisons

SCAS_ , CMPS_

SCASW (scan string of words)

CMP AX, <ES:EDI>

DF { 0 => EDI ← EDI+2
   { 1 => EDI ← EDI-2

SCASB

CMP AL, ...

CMPSB (compare string of bytes)

CMP <DS:ESI>, <ES:EDI>

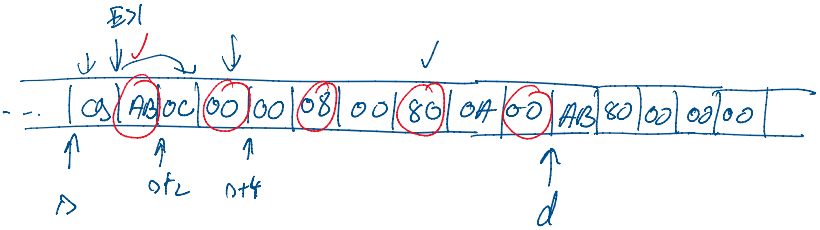
if DF { 0 => EDI ← EDI+1, ESI ← ESI+1
      { 1 => EDI ← EDI-1, ESI ← ESI-1

---

string of words →

s dw 0ABCDh, 12, 800h, 8000h, 10

d dw



s = 00401000h ; ESI = 00401001h

---

string of bytes → mirror the string of bytes

s db 17, 20, 42h, 1, 10, -1

string of bytes – modified the string of bytes

s db 17, 20, 42h, 1, 10, -1

↓

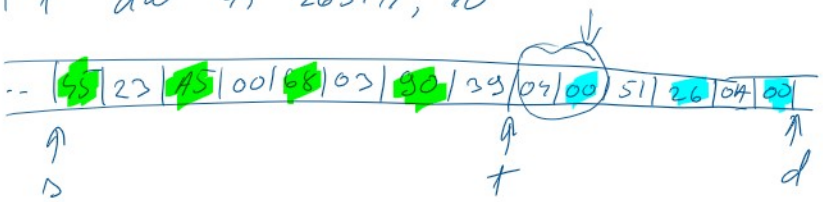
d db -1, 10, 1, 42h, 20, 17

---

two string of words. concatenate the string of low bytes of the words from the first string to string of high bytes of the words from the second string. The resulting string of bytes should be sorted in ascending order (signed interpretation).

s dw 2355h, 0A5h, 3684h, 3990h

t dw 4, 2651h, 10



d: 55h, A5h, 68h, 90h, 00h, 26h, 00h

↓

d: 90h, A5h, 00h, 00h, 26h, 55h, 68h