```
; Comments are preceded by the ';' sign. This line is a comment (is ignored by the assembler)

bits 32 ; assembling for the 32 bits architecture

global start ; we ask the assembler to give global visibility to the symbol called start
             ;(the start label will be the entry point in the program)
extern exit ; we inform the assembler that the exit symbol is foreign
             ;it exists even if we won't be defining it
import exit msvcrt.dll  ;we specify the external library that defines the symbol
      ; msvcrt.dll contains exit, printf and all the other important C-runtime functions

; our variables are declared here (the segment is called data)
segment data use32 class=data
   a dw 3
   b dw 4

; the program code will be part of a segment called code
segment code use32 class=code
start:
   mov ax, [a]   ; ax = a
   add ax, [b]    ; ax = a + b = 7
   ; call exit(0) ), 0 represents status code: SUCCESS
   push dword 0 ; saves on stack the parameter of the function exit
   call [exit] ; function exit is called in order to end the execution of
the program
```

SEMINAR | 06.10.2022
ASC

iA-32

- Byte 0-7
- Word 0-15
- Doubleword 0-31 (LowWord/HighWord)
- Quadword 0-63 (LowDouble Word/-u-)

constants iA-32

- numbers
  - → base 2 ; 101b, 11100b
  - → base 16 : 34ABh, 0ABCDh
  - → base 10 : 20, -114
- character
- string

Declaring constant (assembly language)

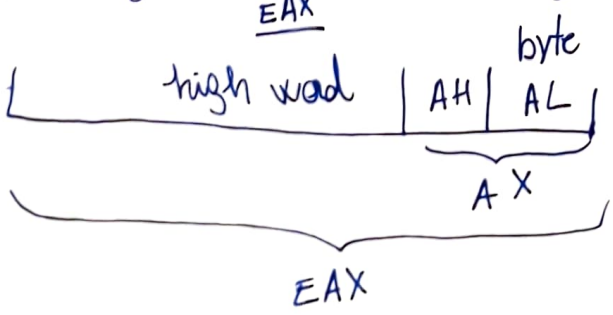
ten EQU 10

→ 2 kinds of variables

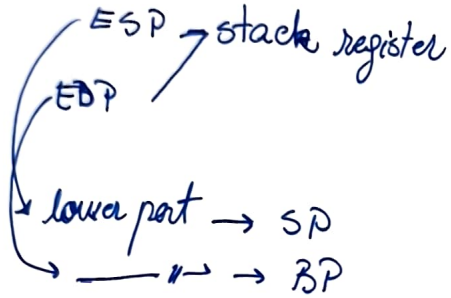
- predefined var (registers) — general (32 bits/register)
  - segment (16 bits/register)
  - other (32

FLAG → sets what the register does

General reg. (EAX, EBX, ECX, EDX)



L - lower byte
H - higher byte



EDI - index register → DI

2. Segment registers

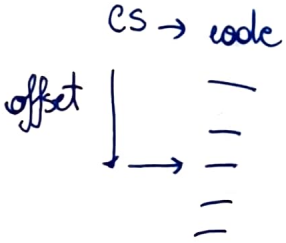
- CS - code
- DS - Data
- SS - stack
- ES - Extra

→ segment

↓ def storing addresses

3. Other reg

- EIP & Flags
  ↓
- IP (subregister)



User def var

name, data, type, initial value

name byte value
a DB 2 3

↓

declare byte

| a 1 | DW | 23ABh |

base16

↓

declare define word

a12 DB 101

↓

User def var

a RESB 1 → reserve 1 byte

a RESB 64 → reserve 64 bytes

a RESW

functions

parameters → dwords

add [a], 3 → wrong (data type!)
→ ever

add dw [a], 3 → right

MOV instruction

→ copy a value ~~from~~ a.

~~dest = s~~ dest ≠ source = registers, var, const

of type byte, word, dword

dest ≠ constant

Effect dest = source

Restrictions: both operands → same size

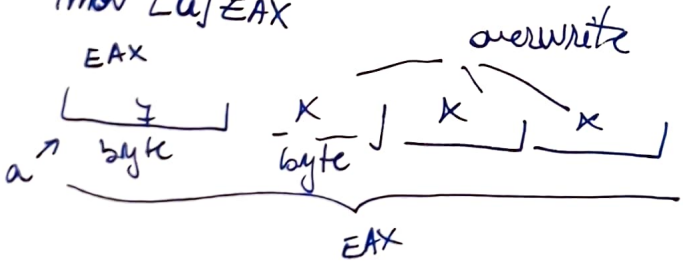
Ex: mov ax, 3; ax = 3

mov bx, ax; bx = ax

mov [a], eax

Ex: a db 7

mov [a] EAX



ADD instruction

syntax: ADD dest, source

Effect: dest = dest + source

at least 1 = register/constant

SUB instruction

Syntax: SUB dest, source

Effect: dest = dest − source

Repr. of integer numbers in the computer's memory

CPU → repr. an integer number on 1, 2, 4, 8 bytes on the IA-32

unsigned repr. ⊕ numbers = repr. of that number in base 2
↓ positive

signed representation of numbers

- bit sign + normal representation of the number

ex: 5 = 0000 0101
-5 = 1000 0101

- we don't use a complement of 1 → invert all bits

ex: 5 = 0000 0101
-5 = 1111 1010 } we don't get a 0

- the signed repr. of a positive number is equal to the unsigned repr. of that number in base 2

—— " —— neg number = repr. in 2's complement

ex: 17 on 8 bits

0001 0001
↓ positive

ex: -17

1 0000 0000 −
1 0001
1110 1111

17: 0001 0001
1110 1111

(copy)
We stop till we found the first 1 then invert 0 ↔ 1

10 (hexa) = 16 − 1 = 15 (F in hexa)
1
F

Exercises

1. Write a program that computes 4+2

```
bits 32
global start
extern exit
import exit msvcrt.dll
segment data use32 class = data
        ; no variables needed, but we can use them
segment code use 32 class = code
start :
    mov ax, 4      → al, but they are small values
    add ax, 2      → mov ax, 1
                     sub ax, 5      1-5
    push dword 0   2.
    call [exit]
```

4. Write a program that computes (a+b)-c, a, b, c - doublewords

```
bits 32
global start
extern exit
import exit msvcrt.dll
segment data use32 class data
    a dd 1
    b dd 2
    c dd 3
segment code use 32 class = code
start
    mov eax, [a]
    add eax, [b]
    sub eax, [c]
    push dword
    call [exit]
```

-39

```
1 0000 0000 0000 0000
                10 0111
-----------------------
1111 1111 1101 1001
```

~ filled with 1/0 (pos/neg) for bigger numbers

```
0001 0001        0100 1000
1110 1110 +      1011 1000
          1
-----------------------
1110 1111
```

Interpretation

mul instruction → multiplies the value from AL reg with the value from the BL reg and store the result in AX

- signed
- unsigned > interpretation

AL = 11101111

- unsigned: positive (classic) $1*2^7 + 1*2$
- signed • 1 - negative
  - complementary code

AL = 0000 0101 unsigned: 5
= 5 signed: still 5 (positive)

signed & unsigned instructions
mov, add, sub → don't care ab. signed/unsigned repr. of numbers.

do care: div mul (unsigned)
: idiv, imul, (signed)

-10 on a word

10:2=5 r0
5:2=2 r1
2:2=1 r0
1:2=0 r1

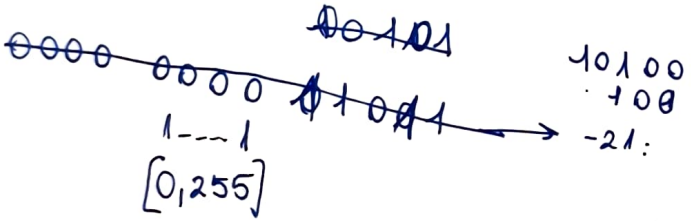
1010 - lyuay

10: 0000 1010 → -10: 1111 1111 0110

1111 1111 1111 0110

-21 on a word

21:2=10 r1
10:2=5 r0
5:2=2 r1
2:2=1 r0
1:2=0 r1



[0,255]

[-128,127]

a ∈ [-300,100] → data type: word

F = 8 + 4 + 2 + 1

2^1 3 2 1 0

A = 8 · + 2

1 0 1 0

1 → 0 0 0 1

7 → 0 1 1 1

Base 16 to base 2

$6CB1_{(16)} = 0110\ 1100\ 1011\ 0001_{(2)}$

10 → 0000 1010 ⇒ 0Ah

B_2 ↓ representation in memory

→ negative numbers representations (no losing memory) | complementary code

0000 1010 +

[?]

1 0000 0000

↓

borrow/digit (extra)

→

~~1~~

1 0000 0000 ←

ex: -10

0000 1010

1111 0110 (-10)

10 +

-10

0

100- (16)

0A

F6 ↓ (16-10)

1→0 / 0→1

0000 1010 → last 2 the same

1111 0110

1111 0110 (2) → 128 + 64 + 32 + 16 + 4 + 2 = 246 (10)

-10

H/W

6. 1010 - (2)

1

1001

5. 1000 - (2)

01

10

~~0000~~

110

6. ~~B + 2~~

B + (16)

2

D

(16)

10 + → 16

A 10

1A 26 → 1A

26 | 16

16 | 1

10

ASC - Seminar 2

20.10.2022

Signed and unsigned instructions (multiplications, divisions, conversions)

11111111b * 11111111b → 255 · 255

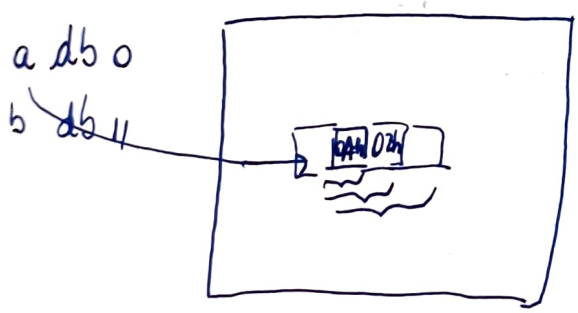
↘ (-1) · (-1)

unsigned: div, mul

signed: idiv, imul

ADD [a], [b] ; ! incorrect

ADD [a], 1 ; ! incorrect, nu are dimensiune

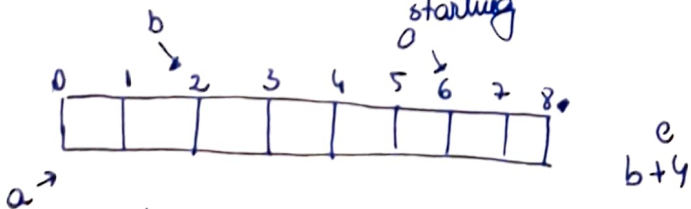


⇒ ADD byte [a], 1

ADD [a], AL

mov EAX, [a] ; moves the doubleword

mov EAX, a ; moves the adress



a dw 0

b dd 1

c dw 2

mul

source → byte ⇒ AX = AL * source

source → word ⇒ DX:AX = AX * source

source → dword ⇒ EDX:EAX = EAX * source

DX : AX (rotație)

ww ww

high low

EDX : EAX

dword high dword low

div

source = byte ⇒ AL = AX / source

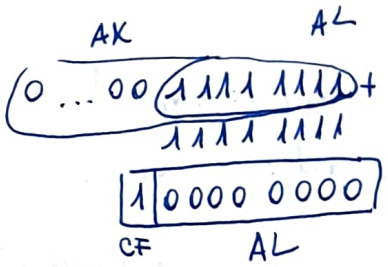
AH = AX % source

source = word ⇒

① $x \xrightarrow{byte} = ((a+b)*c)/d$

UNSIGNED + a, b, c, d - bytes

DATA SEGMENT	CODE SEGMENT
a db 10	mov AL, [a]
b db 11	add AL, [b]
c db 12	~~mov~~ mul byte[c] ; AX = (a+b)*c
d db 13	div byte[d] ; AL = ((a+b)*c)/d AH = ((a+b)*c) % d
x db 0	mov [x], AL
(x RESB 1)	



AL → AX : mov AH, 0

BX → eX : BX : mov eX, 0

AX → EAX : mov BX, AX

mov EAX, 0

mov AX, BX

EAX —unsigned→ EDX : EAX

mov EDX, 0

DX : AX —unsigned→ EDX : EAX

PUSH DX ; DX = 1234h

PUSH AX ; AX = 5678h

POP EAX (high DX, low AX)

mov EDX, 0

```
idiv BX ; AX = a*b / d

mov [x], AX
sub [x], [c]
mov AL, [c]
CBW ; AL → AX
sub [x], AX
```

(4) $x + \frac{a \cdot b}{c - d} + f \cdot g$

SIGNED + x - quad, a, d - byte, c - double, b, f, g - word

DATA SEGMENT

```
a  db  10
b  dw  11
c  dd  12
d  db  13
f  dw  -14
g  dw  15

x  dq  0
```

CODE SEGMENT

```
mov AL, [a]
cbw ; AX = a
imul word [b] ; DX:AX = a*b
push DX ; high part
push AX ; low part
pop EAX
cdq ; EDX:EAX = a*b
mov EBX, EAX
mov ECX, EDX ; ECX:EBX = a*b
mov AL, [d]
cbw
cwde ; EAX = d
mov EDX, [c]
sub EDX, EAX ; EDX = c-d
xchg EDX, ECX ; EDX:EBX = a*b
                ECX = c-d
mov EAX, EBX ; ~~EAX~~
idiv ECX ; EAX = (a*b)/(c-d)
cdq ; EDX:EAX = (a*b)/(c-d)
```

```
mov ECX, [x+4]
mov EBX, [x]
add EBX, EAX
adc ECX, EDX ; ECX:EDX = x + a*b / c-d

mov AX, [f]
imul word [g] ; DX:AX = f*g
push DX
push AX
pop EAX
cdq ; EDX:EAX = f*g
add EAX, EBX   adc EDX, ECX
```

Signed conversions

CBW ; AL ⟶ AX

CWD ; AX → DX:AX

CWDE ; AX → EAX

CDQ ; EAX → EDX:EAX

(2) $x = (a - b * c)/d$ $\quad b*c$ – word

UNSIGNED + a, b, c, d – byte $\quad$ a – byte ⟶ word

DATA SEGMENT	CODE SEGMENT
a db 10	mov AL, [b]
b db 11	mul byte [c] ; AX = b * c
c db 12	mov BL, [a]
d db 13	mov BH, 0 ; BX = a
x db 1	sub BX, AX
	mov AX, BX
	div byte [d]
	mov [x], AL

SIGNED

V

imul byte[c]

mov BX, AX

mov AL, [a]

CBW

sub AX, BX

✓

idiv

V

(3) $x = \frac{a*b}{d} - 1$

SIGNED + a, c, d – byte b – word

a – byte ⇒ word

a * b ⇒ dword

d – byte ⇒ word

$\frac{a*b}{d}$ – word

c – byte ⇒ word

x – word

DATA SEGMENT	CODE SEGMENT
a db	mov AL, [a]
b dw	CBW ; AX = a
c db	imul word [b] ; DX:AX = a * b
d db	mov BX, AX ; DX:BX = a
	mov AL, [d]
	CBW ; AX = d
	xchg AX, BX ; swap DX:AX = a * b BX = d

EDX : EAX +

ECX : EBX

EDX+ECX+CF EAX+EBX

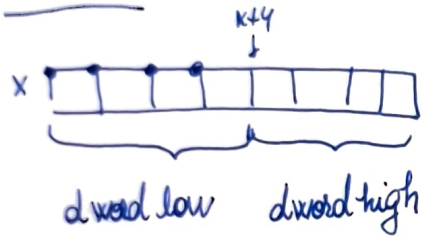
ADD EAX, EBX

ADC EDX, ECX

→ push/pop or the same

DX : AX +

CX : BX



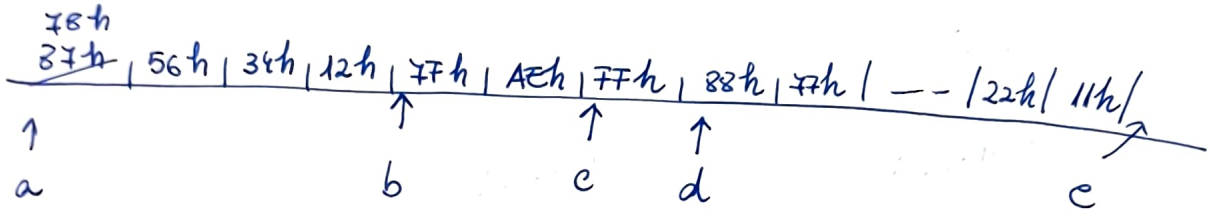
mov ECX, [X+4]

mov EBX, [X]

ASC - Seminar 3

03.11.2022

Little endian representation



a dd 12345678h

b dw 0AE7Fh

c db 0FFh

d dq 1122334455667788h

e dw 20

20 ⟶ 10000 -

-20 = FFECh

AX = 88FFh (AH AL)

~~78h 56h 34h 12h FFh~~

MOV AX, [a]

MOV DX, [a+2]

① • adress high byte of b =

• value b =

② . value of the high byte of a

= 34h

• adress of the low byte of the high word a

③ adr of the high byte of the low word of the high dw

low byte is the first one in the data segment

a*b + c*d , a-byte, b,c,d-word

```
a db 2
b dw 3
c dw 4
d dw 5
```

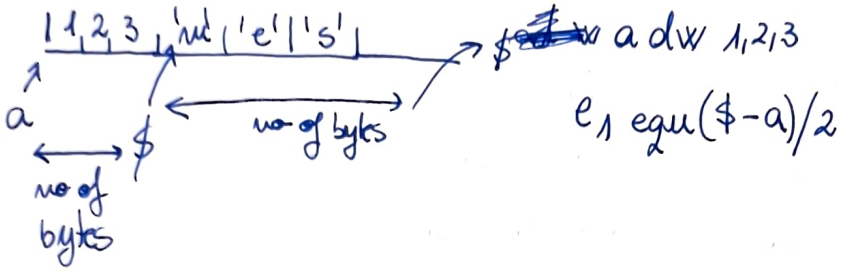
```
mov AL, [a]
mov AH, 0
mul word [b]
mov BX, [b]
mul word [d]
```

```
mov AL,[a]
mov AH,0
mov BX,[b]
mul BX ; res → DX:AX
mov BX, AX
mov BX, DX
mov AX, [c]
mul word [d] ; res → DX:AX = c*d
add AX,BX
adc DX,CX
```

```
mov AL,[a]
mov AH,0
mov BX,[b]
mul BX
push DX
push AX
mov AX,[c] ; res → DX:AX = c*d
mov BX,[d]
mul BX
pop BX   CX:BX → a*b
pop CX   <- add AX,BX
         <- adc DX,CX
```

Strings

$ - location counter



→ declare len as a constant

'm', 'e', 's', ... (=) 'message'

Adress = adress of the 1st element

Lenght = no of elements

Direction for iterating the string

a) ESI - index of the string

Ex:

```
a db 'abcdef'
l1 equ $-a
b resb l1
mov ESI, 0
cmp ESI, l1
JB instr
redo
cmp ESI, l1
JB instr
JMP fin
```

```
instr:
mov AL, [a+ESI]
sub AL, 'a'-'A'
mov [b+ESI], AL
INC ESI
JMP redo

fin:
push dword 0
call [exit]
```

2) ECX : EBX - EAX

a) UNSIGNED

```
mov EDX, 0
sub EBX, EAX
sbb ECX, EDX
```

b) SIGNED

```
cdq ; EAX → EDX : EAX
sub EBX, EAX
sbb ECX, EDX
```

! EDX : EAX - EBX

```
push EDX
push EAX
mov EAX, EBX
cdq  mov EDX, 0
mov EBX, EAX
mov ECX, EDX
pop EAX
pop EDX
sub EAX, EBX
sbb EDX, ECX
```

JNA label 1

jump if not above

...

label 1

a - b

a < b (a below b) OF =

if (c1)

{ ... }

else

{ ... }

CMP ... c1

J ... l1 (true)

e_1 : ... if

else JMP l2

b) ESI - offset of the current byte

code sequence:

```
mov ESI, a
mov EDI, b
redo:
  cmp
mov ECX, l1
redo:
  jECX: end
mov AL, [ESI]
sub AL, 'a'-'A'
mov(EDI), AL
inc ESI
inc EDI
(dec ECX)
(jmp redo) LOOP redo
end:
push dword 0
call [exit]
```

ASC - Seminar

17.11.2022

24:16 = 1 r 8
16 = 0 x 1

Show the data segment for the following variables

a1 db '24', '68' - 18,

a2 dw 24, 68 - 18

a3 db 2, 4, 6, 8

a4 dw 2, 4, 6, 8

a5 db 2468h

a6 dw 24h, 68h

a7 dd 02040608h

a8 db 24h, 68h

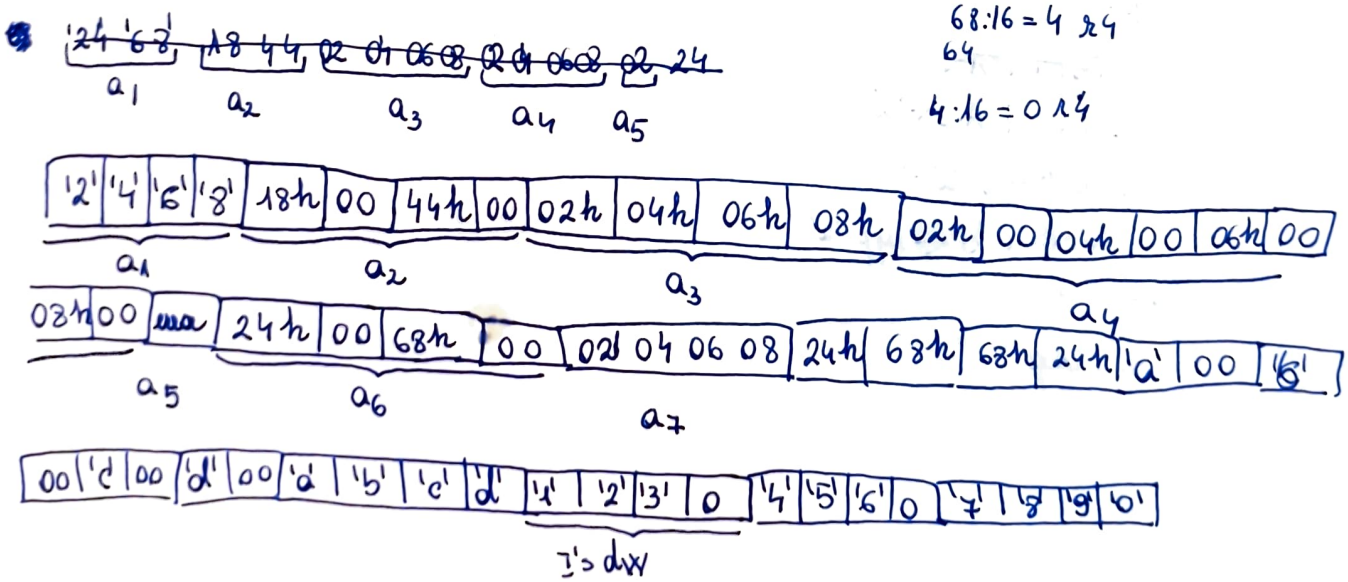
a9 dw 2468h

a10 dw 'a', 'b', 'c', 'd'

a11 dw 'abcd'

a12 dd '123', '456', '4890'

68:16 = 4 r 4
64
4:16 = 0 x 4



String instructions

1. Being given a string of bytes containing lowercase letters, build a new string of bytes containing the corresponding uppercase letters.

DS:

```
a db 'a', 'b', 'c'
la equ $-a
b times la db 0
```

CS:

```
mov ECX, la
mov ESI, a
mov EDI, b
cld → clear direction flag
jecxz end_loop
loop:
    LODSB ; AL ← 'a'
    SUB AL, 'a'-'A'
    STOSB ; b ← upp 'a'
loop loop
end_loop:
```

```
LODSB
cmp DL, AL
JLE not
    mov EDI, ESI
    sub EDI, 2
    STOSB
    mov AL, DL
    STOSB
    mov BL, 1
not:
  dec ESI
loop for
end for
cmp BL, 0
jne endwhile
```

DL = d[i]
AL = d[i+1]

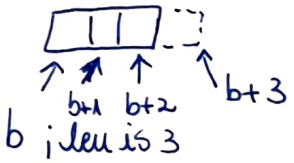
```
mov ECX, la
mov ESI, a
// std ; set direction flag DF=0
cld ; DF=1
mov EDI, d
JECXZ end_loop
loop_repeta: ; low bytes of the words from the first string (in AL)
    LODSW ;
    STOSB ;
end_loop:

mov ECX, lb
mov ESI, b
cld
JECXZ End
concatenate:
    LODSW
    mov AL, AH
    STOSB
loop concatenate
End
; setting part                    3 5
while:                            5, 5, 7, 4
mov BL, 0                         BL = 01
mov ESI, d
mov ECX, ld-1
cld
JECXZ endwhile endfor
for:
    LODSB ; AL = d[i]
    mov DL, AL
```

DS:

```
a db 'abcdef'
len equ $-'a'
lr resb len
```

CS:

```
mov ECX, len
mov ESI, a
mov EDI, b+len-1
JECXZ end.
loop:
    cld ; DF=0
    LODSB; AL='a'
    std
    STOSB ; CDS:ESD, a[l-1]='a'

    loop
```



~~a dw 1,2,3~~

l equ ($-a)/2 → because the number of words

a+l*2-1

a+6-1

```
edaud
    a dw 1,2,3
```

1	0	2	0	3	0

a, a+2, a+4

a+(l-1)*2 → correct offset for the last element

Ex3. ~~2 strings of words are given. Concatenate the strings of low bytes~~

```
a dw 1,2,3,4,5
b dw 6,7,8,9,10
la equ ($-a)/2 → contains the len(a+b)
lb equ ($-b)/2
c times ld db 0
ld equ la+lb
```

Quiz 1 : 10

Quiz2 : 9

Quiz : 10

1 0000 00

1 000000

1 0000000 CF=1

CD AB DC FE

3 = 011 ⇒ -3 = 101

4 = 010 ⇒ -4 = 110

100 (-3+-4)

1111111100

1234 dd 2 , 16 dup

34 12 34 12 16

34 12 00 00 34 12 00 00 16

2 00 00 00 2 00 00 00 2 00 00 00 20000000

00 00

15 14 13 12 11 10 9 8 | AH | 7 6 5 4 | AL | 3 2 1 0

7 6 5 AH 3 2 1 0 | AL

15 14 13 12 11 10 9 8 | 7 6 5 4 3 2 1 0

3c 2b 1a 00 9f 4d 00 00 27 5d 6e 00

DCAB DCFE

010 254 $256 \cdot 7.16^3$

110 128

0001 0000 16

0000 0001 6

963

0110 0010

0010 0000

34 12 00 00 34 12 00 00 16 0100 0010

1 2 3 10 20 30 20

0110 0010

1100 1000

10 3 h = 3 10000 1010

2^1 0

16^2 +

103 h = 2563

$200 = 2^7 + 2^6 + 2^3$

128 64

192

c) AB DC FE

16 | 200
6 | 98
96 3 | 102

182
128 64 16
2^7 2^6 2^4 2^1

0110 0010
1101 0010
1000 0000

ax, 400h 1024

bl, 0feh

idiv bl

14.1
15
16 | 3
90
15
240
14
254

0100 0000 0000
1111 1111
0000 1111 1110

240
224
192
250 = 128 + 64 + 32 + 16 + 8 + 2
2^7 2^6 2^5 2^4 2^3 2^1

1

11111010
01111

a344h 30h 11223344h 46ab89ch

~~44a3~~

44 a3 00 00 30 00 00 00 44 33 22 11

9c b8 6a 40

1 1 1

1 1 0

CF=1 1 0 0

CF=1 0 0 0

6553

9

16
9
144 5

256
9
2304 5

256
16
1506 3

256
4066
2304
6370
144
6514
9

192 224 240 248 252

255 = 128 + 64 + 32 + 16 + 8 + 4 +

1111 1111
1111 1111
1 1.000 0000

! La operațiile OR, AND, XOR, TEST, SHL, SHR, ROL, ROR, RCL, RCR NICIODATĂ const nu e pe prima poziție

RCL : CF = 1 (ca la rol + știa CF înainte și-l poți reține)

```
mov al, binary-num
mov cl, 2
rcl al, cl . binary_num CF
```

RCR

ROL/ROR : ! the last rotated bit is kept in CF

! mov → uses the little endian representation

256
3
768 1

1 2 3 10 20 30

310 = 16 + 768 =

10 3 = 2563

~~200~~
100h

mov al, -2
mov bl, -128

(-2) * (-128) = 256
ax

ABCDh ← FEDCh ← ←

ax = ffffh de ce pune f ?
cwd → dx:ax : ffff ffff
dx ax

dx + 1 → dx = 0000

ax = ffffh
cwd → dx:ax : 0000 ffff ?

AL = ?
a = 0ABCDh, 0FEDCh | CD AB DC FE

lodsw ; AX = ABCD
lodsb ; AX = AB DC
AL

→ când lodsw ia word-ul ca în DS
→ când lodsb ia byte-ul din Mem. seg.

lodsw ; AX = ABCD
lodsw ; AX = FEDC ⇒ AL = DC !!!

mov al, -2
mov bl, -128
mul bl ; o să vadă al = 254 (+256) și bl = 128 (+256)
ax = FF00h

nu dă error

A B C D E F
10 11 12

a db 1, 2, 3

b dw 1, 2, 3

c dd 1, 2, 3

MEMORY SEGMENT

01, 02, 03,

01 00, 02 00, 03 00,

01 00 00 00, 02 00 00 00, 03 00 00 00

16 | 15
9 | 16
144 | 90
47 | 240
188

---

d dw 300, 400, 500
012Ch 0190h 01F4h

MEMORY SEGMENT

2C 01, 90 01, F4 01,

location counter = the number of generated bytes ($)
in hexa

$$ → the start of the current section

$-$$ → how far into the section (how many bytes)

-128, 127

253 | 128
25

$253 = \underbrace{2^7}_{128} + \underbrace{2^4}_{16} + \underbrace{2^3}_{8} + 2^1 + 2^0$

$252 = 128 + 16 +$

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

mov dh, 62 | 62(s)

mov eh, 200 | -56(s)

sub dh, eh

$(256 + 62) - 200 =$

-3 & -4

256
256

15.16 + 14

```
  15
  16
 ---
  90   3
  15
 ---
 240
 264
```

Seminar 6

15.12.2022

ASC

Multimodule programming

① Write a ~~function~~ program that concatenates 2 strings by calling a function from another module and then prints the result on the screen.

Main:

```
global

segment data use32 class=data:
    str1 db "Hello", 0
    len1 equ $-str1
    str2 db "world", 0
    len2 equ $-str2
    strres resb len1+len2, 0
segment code use32 class=code:
start:
    push dword str1
    push dword str2
    push dword strres ; the address of the resulted string
    call strcat
    add ESP, 4*3
    push dword strres
    call [printf]
    add ESP, 4
    push dword 0, call [exit]
```

concatenate:

```
global strcat
segment code use32 class=code:
strcat:
    mov EBX, 0 ; string result position
    mov EDI, [ESP+4]
    mov ESI, [ESP+12]
    cld
```

```
        while_loop1:
            lodsb
            cmp AL,0
            jz end_loop1
            stosb
            jmp while_loop1
    end_loop1:
        mov ESI,[ESP+8] ; str 2

    while_loop2:
        lodsb
        cmp AL,0
        jz end_loop2
        stosb
        jmp while_loop2
end_loop2:
mov AL,0
stosb
ret
```

REP MOVSB

↓

repeat an instruction
ecx times

(3) (Seminar 5)

Read from the keyboard a number n in base 16 that can be repr. ~~[illegible]~~ on a word (no validation needed). Open the

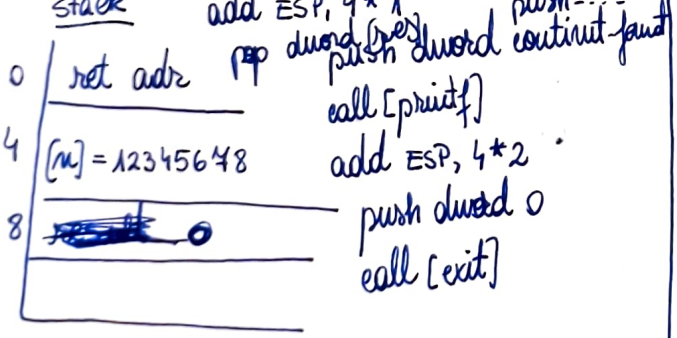
```
data:
n dd 0
format db "%x", 0
fd dd -1
file_name dd "in.txt", 0
acc_mode dd "r", 0
x dd 0
format_1 db "%c", 0
code:
    start
        push dword n
        push dword format
        call [scanf]
        add ESP, 4*2
        push dword acc_mode
        push dword file_name
        call [fopen]
        add ESP, 4*2
        mov [fd], EAX
        cmp EAX, 0
        je end_loop
        mov ECX, 16
    solve: push ECX
        push dword x
        push dword format_1
        push dword [fd]
        call [fscanf]
        add ESP, 4*3
        shr [n], 1
        jnc do_not_print
        push dword [x]
        push dword
        call [printf]
        add ESP, 4*2
    do_not_print:
        loop solve ← pop ECX
```

② Write a program that prints the sum of the digits of an unsigned number represented on a doubleword (by calling a function for the sum)

Main:

```
global start
extern sum_digits

segment data use32 class=data

n dd 12345678

res dd 0
continut_format db "%u", 0
segment code use32 class = code

start:
    push dword res
    push dword [n]
    call sum_digits
    add ESP, 4*1
    push dword [res]
    push dword continut_format
    call [printf]
    add ESP, 4*2
    push dword 0
    call [exit]
```

stack



sum_digits:

```
global sum_digits
segment data use32 class=data
ten dd 10
segment code use32 class = code
sum_digits:
    mov EAX, [ESP+4]
    while_loop:
        mov EDX, 0 ; EAX → EDX:EAX
        div dword [ten]
        add [result], EDX
        mov EDX, 0
        cmp EAX, 0
        jne while_loop
    mov [ESP+8], [result]
    mov [EDX], [result]
    mov [ESP+8], EDX
    ret
```

ASC - Seminar 8

12.01.2022

Exercise 1

Main:

```
data
input_format db '%s', 0
input resb 100
max dd 0
max_filename resb 100
code
start:
loop_reading
    push dword input
    push dword input_format
    call [scanf]
    add ESP, 4*2
    cmp byte [input], 'e'
    jne not_end
    cmp byte [input+1], 'n'
    jne not_end
    cmp byte [input+2], 'd'
    jne not_end
    cmp byte [input+3], 0
    je end
not_end:
    push dword input
    call most_upper
    add ESP, 4
    cmp dword [max], eax ; eax -> set by function
    jae keep_reading
    mov dword [max], eax
```

1 function

```
most_upper
global most_upper
data
    extern
    filemode db 'r', 0
    fd dd 0
    char_format db '%c', 0
    char resb 1
code
most_upper: mov edx, [esp+4]
    push dword filemode
    push dword [ESP+4]
    call [fopen]
    add ESP
    mov dword [fd], EAX
    add ESP, 4*2
keep_reading:
    push dword char
    push dword char_format
    call [fscanf]
    add ESP, 4*2
    cmp byte [char], -1
    je done_reading
    cmp byte [char], 'A'
    jb keep_reading
```

```
cmp byte [char], 'Z'
ja keep_reading
inc EBX
jmp keep_reading

done_reading:
mov eax, ebx
ret
```

```
main:
mov ESI, input
mov EDI, max_filename

copy_name:
    movsb

cmp [esi], 0
jne copy_name   ← mov [edi], 0
jmp keep_reading

end:
push dword max_filename
call print_file_contents
add esp, 4
call [exit]

push dword [file]
call [fclose]
add esp, 4
```

```
; function
print_file_contents
mov edx,
push dword filemode
push dword [esp+4]
call [fopen]
mov dword [file], eax
add esp, 4*2

push dword [esp+4]
call [printf]
add esp, 4
print_loop:
    push dword char
    push dword format
    push dword [file]
    add esp, 4*3
    cmp [char], -1
    je done
    push dword char
    push dword format
    call [printf]
    add esp, 4*2
jmp print_loop
done:
ret
```

```
data
filemode db "r", 0
format db "%c", 0
fd dd 0
name_format db "%s", 0

char db 0
```

ret adds	esp
filename	esp + 4

```
keep_reading:
    push dword nr
    push dword format_nr
    call [fscanf]
    add esp, 4*2
    push dword [fd]
    call [fscanf]
    add esp, 4*3
    mov edx, [nr]
    shr eax, 16
    mov ah, 0
    push eax
    call prime_nr
    cmp eax, 0
    je keep_reading
    push dword [nr]
    push dword format
    call [printf]
    add esp, 4*2
jmp keep_reading

end:
push dword
```

```
function I
prime_nr
DS

ES
prime_nr:
mov ebx, [esp+4]
mov bl, 2
mov ecx, 0
division:
  div bl
  cmp ah, 0
  je end
  mov eax, [esp+4]
  inc bl
  cmp bl, al
  je end_yes
jmp division
end_yes:
  mov el, 1
end_not:
  mov eax, ecx
  ret
```

```
cmp eax, 0
jle end
push dword [fd]
call [fclose]
add esp, 4
```

~~Exercise 2~~

```
char db, 0
format db '%c', 0

(*)
push dword [char]   → char is a byte ⇒ conversion ⇒ (*) mov eax, 0
                                                     mov al, [char]
push dword format
call [printf]
add esp, 4*2
```

Exercise 2

```
DS
format
~~file_name~~ db "%s", 0
file_mode db "r", 0
fd dd -1
file_name resb 100
format_nr db "%d", 0
nr dd 0

CS
start:
  push dword file_mode
  push dword file_name
  call [scanf]
  add esp, 4*2

  push dword file_mode
  push dword file_name
  call [fopen]
  add esp, 4*2
```