Seminar 911 (s1 11:10-13:00)

Thursday, October 14, 2021 11:07 AM

Representation / Interpretation

mov AX, 2

- unsigned representation of numbers
  - we represent positive natural nr.
  - base 2

  ex. 17 on a byte: 0001 0001

unsigned int a = 1;

- signed representation of numbers
  - we represent positive / negative nr.
  - for positive nr: base 2
  - for negative nr.: 2's complement

\+

\-

MSB = sign bit { 1 −, 0 + }

17 0001 0001

1's complement 1110 1110 −17

17 + (−17) 0001 0001 +
1110 1110
1111 1111

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

−17 |−17| = 17 0001 0001

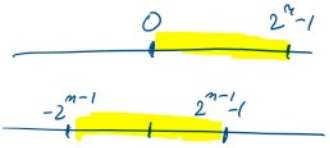
$(1110\ 1111)_2 = (-17)_{10}$

n = 3 [0, 7] unsigned rep.

[−4, 3] signed

n — unsigned $[0, 2^n - 1]$

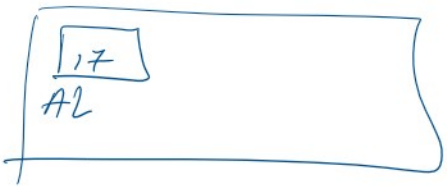
signed $[-2^{n-1}, 2^{n-1} - 1]$



0001 0001

1110 1110 +
1
1110 1111

XXXX XXX1
1111 1111
0000 0000 −
0001 0001
1110 1111



Interpretation

AL = 1110 1111 $= (239)_{10}$ $[0, 2^8 - 1]$ [0, 255]

AL = 1110 1111 $= (239)_{10}$ $[0, 2^8-1]$ $[0, 255]$

mul bl ; AX = AL · BL unsigned interpretation

imul bl ; AX = AL · BL signed interpretation

↳ 1110 1111 −17

$(0001\ 0001)_2 = (17)_{10}$

$AL = (0001\ 0011)_2 = (19)_{10}$

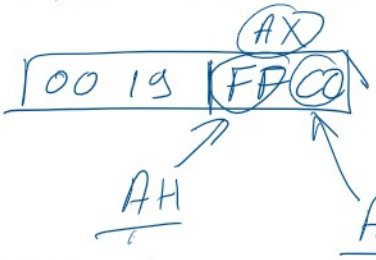
mov AL, −2
mov AL, 2

			CF	OF
mul AL	17	0001 0001 +		
imul AL	−17	1110 1111		
		0000 0000	1	

On IA 32 related to signed / unsigned representation of nr.

1) ↳ instr. which do not care about unsigned / signed repr. of nrs.
   mov, add, sub
2) ↳ instr. which interpret the operands as unsigned nr.: div, mul
3) ↳ — // — // — // — signed nr: idiv, imul, cbw, cwd, cwde

EAX



		big endian	little endian
−2	byte	FE	FE
	word	FFFE	FE FF
	dword	FFFF FFFE	FE FFFF FF

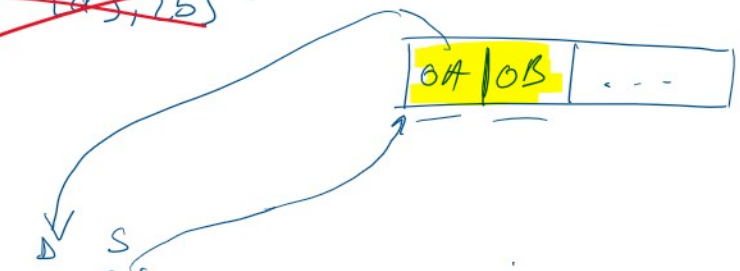
big endian 1 2 3 4

little endian 4 3 2 1

→ all operands must have the same size/type

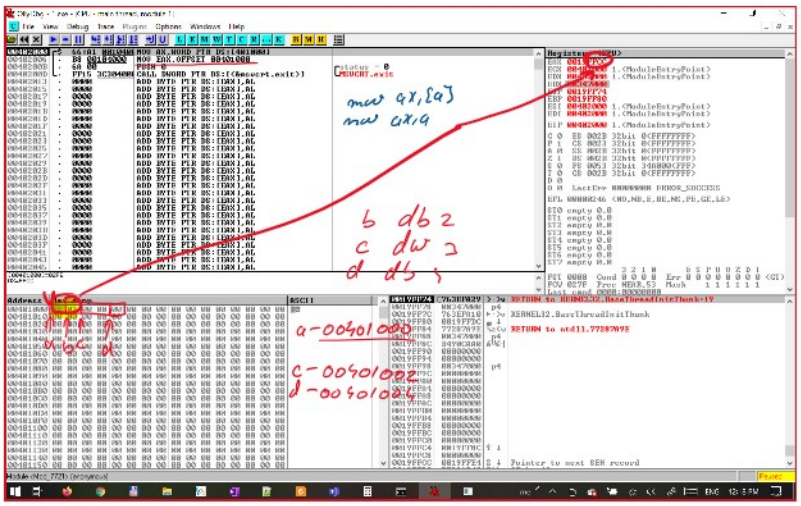
a db 10
b db 11
⋮
add eax, [a]
~~add [a], [b]~~

→ all operands must have the same size/type
↳ at least one of the operands must be a general purpose reg. or a constant and if it is a constant, the constant can not appear as a destination operand



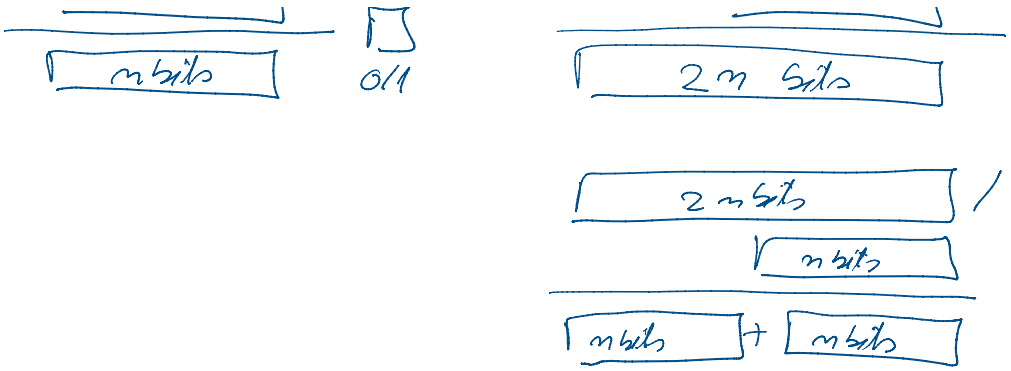
mov ax, [a] ; AX= 0B 0A
mov eax, a ; EAX= ?

[a] a



imith.

MUL , DIV , IMUL , DIV



MUL source

if source – byte AX = AL * source

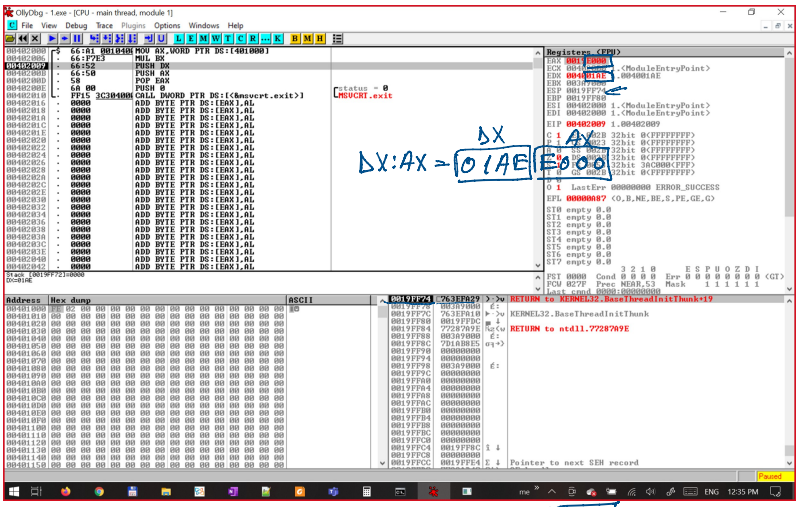
word DX:AX = AX * source

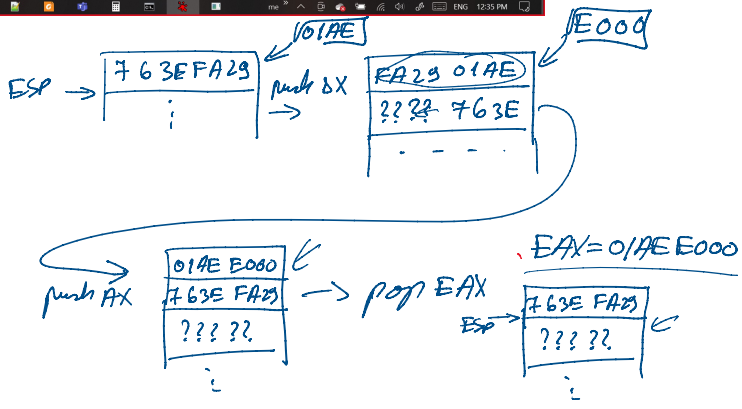
dword EDX:EAX = EAX * source

MUL 2

MUL word 2

MUL BX ; DX:AX = AX * BX



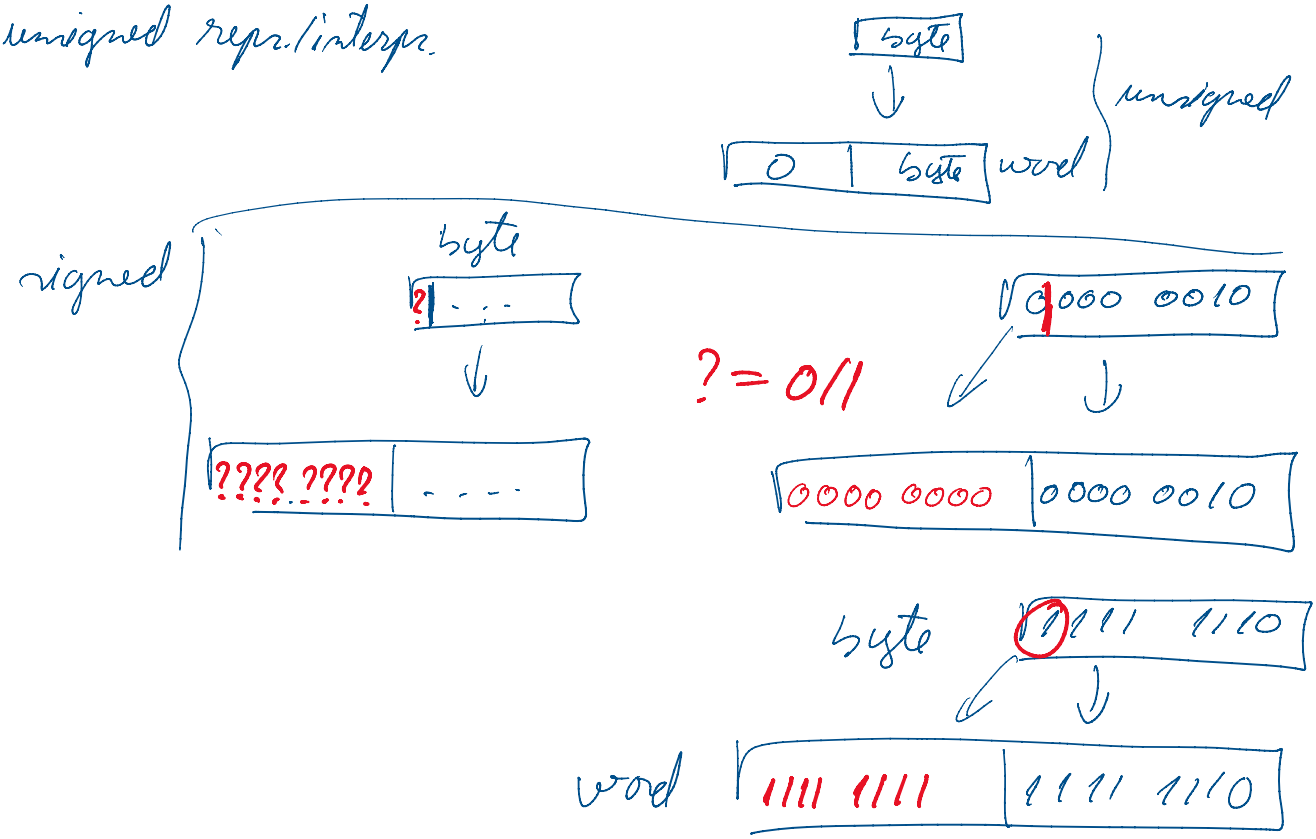


CBW , CWD , CWDE

mov al , 2

cbw

unsigned repr./interpr.




```
mov al, -2
cbw ;   AX = -2
```

CBW syntax
AX ← AL effect

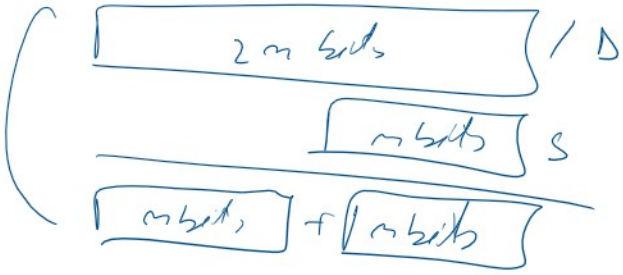
CWDE

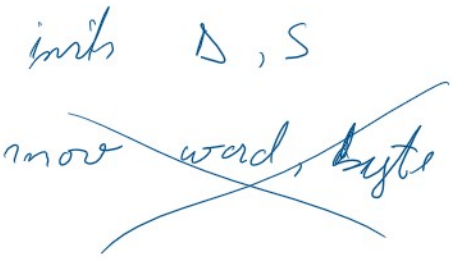
CWD syntax
DX:AX ← AX effect

```
DIV  BL   ; BL = 0
```

```
mov  ax = -2   ; AX = (FFFE)16 → (65534)10
mov  bl = 1      BL = 1
→ div  bl ;  AL =
             AH =
```

Seminar 3





D ∈ {S} {byte, word, dword}

<=> D ∈ {word, dword, qword}

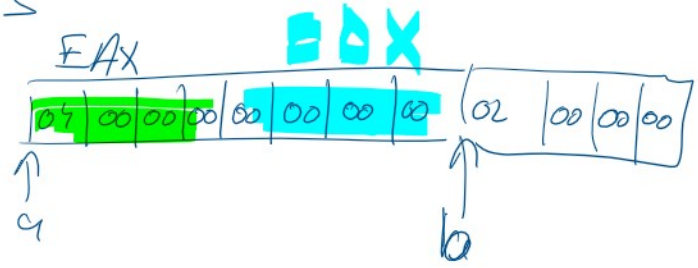
```
.data
    a  dq  4
    b  dd  2

.code
    mov ebx, [b]
    div ebx
```

<=> div dword [b] ✓

; div dword => EAX = EDX:EAX / S ; EDX = (EDX:EAX) % S

~~mov EDX:EAX, [a]~~

```
mov eax, [a]
mov edx, [a+4]
div dword [b]
```



65000 word / 1 byte = byte + byte

(0)

(-3)

$q^{2}word - b^{3}yte$

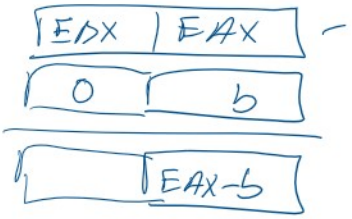
unsigned interpretation / representation

mov eax, [a]

EDX | EAX -

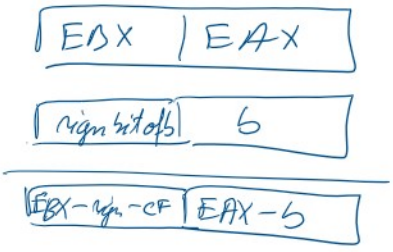
unsigned interpretation / representation

```
mov  eax, [a]
mov  ebx, [a+4]   ; ebx:eax = a
sub  eax, dword [b]
sbb  ebx, 0       ; ebx = ebx - 0 - CF
```



signed interpretation / representation

```
mov  ecx, [a]
mov  ebx, [a+4]
mov  eax, [b]
cdq  ; extend sign bit of eax -> EDX
sub  ecx, eax
sbb  ebx, edx
```



---

little endian

base 10 1234

34

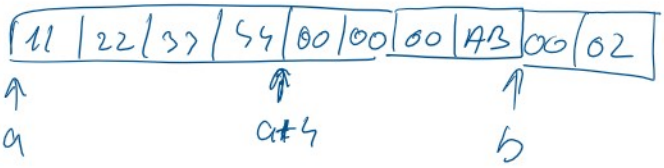
```
a  dd  1122334455h, 0ABh
b  dw  2
```

dword

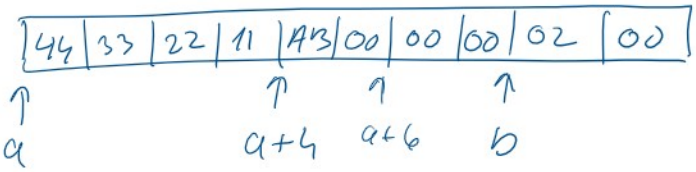
1234

A = 4 bits

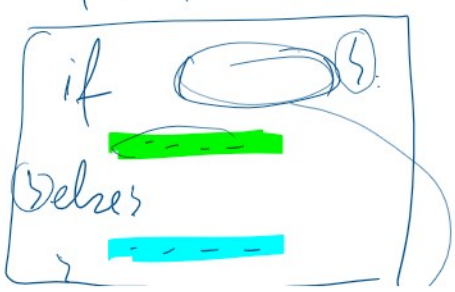
big endian:

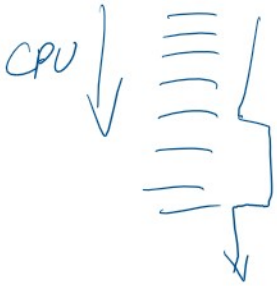


little endian:



if, for, while





a ? b
> <
≥ ≤
==

test
cmp

Cmp D, S ; D-S and set flags

a db 2
b db 3

cmp a,b ; a-b = 2-> => set flags

- a < b CF jb ZF, CF, OF, SF
- a > b CF
- a == b ZF
- a ≥ b CF, ZF
- a ≤ b CF, ZF

if ⟹ asm

cmp D, S
jcc label
label:

unsigned nr:

cmp a,b

jb label ; jb = jump if below (a < b)
jbe label ; a ≤ b
jnb label ; a ≥ b
ja label ; a > b
jae label ; a ≥ b

jae label ; a >= b

jna label ; a < b

jnae label ; a <= b

signed interpretation

jl label ; (jump if less) jump to label if a < b (interpret operands as signed nrs)

jg label ; (jump if greater) jump to label if a > b (signed int.)

jle , jlne . . .

jge , jgne . . .

---

a = b => je label

---

unconditional jump instruction

jmp label

- working with strings of bytes

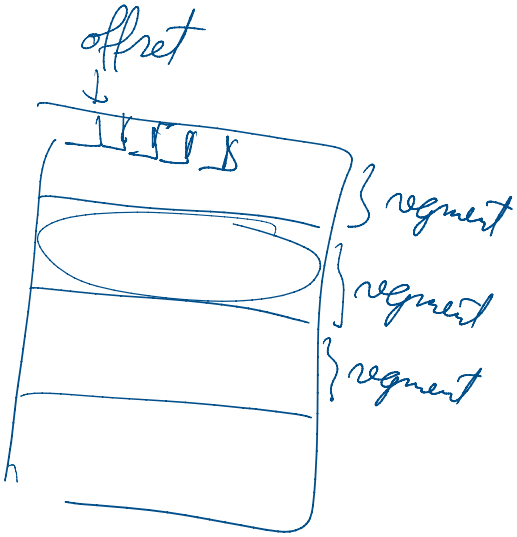
memory addresses and offsets

mov ax, [a]

segment_selector : offset

16bit 32bit

CS, SS, DS, ES.



offset = base + (index * scale) + displacement

offset = [EAX … ESI] + [(EAX, EDX, ECX, EBX, ESP, EBP, ESI, ESI)] * [1, 2, 4, 8] + [None, 8bit, 16bit, 32bit]

base | index | scale | displacement

00400010

→ mov ax, [a] ; only displacement

mov ax, [eax] ; base, index

mov ax, [a + eax + ebx] ; base, index and displacement

:

---

string of bytes containing lowercase letters, build a new string of bytes containing the corresponding uppercase letters

a, b, c → A, B, C

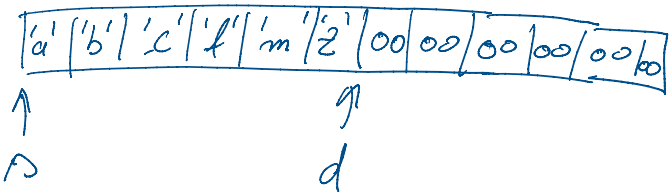
.data

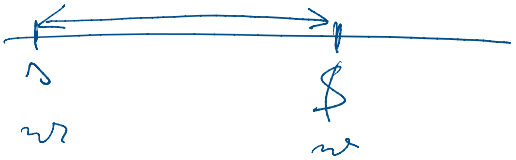
s db 'a|b|c|f ... z'

s_l equ $ - s (32bit value)

d times s_l db 0

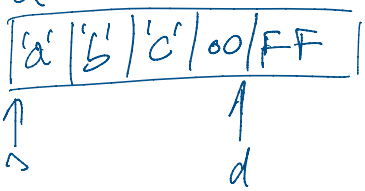
ASCII → 8bits





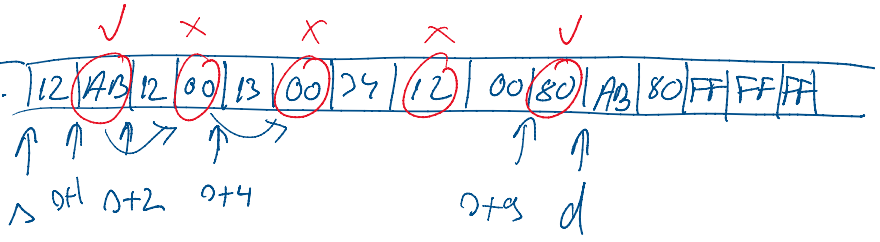
o dbw '@bc'

d db -1



string of word → string of bytes

```
.data
s   dw   0AB12h, 12, 13, 1234h, 8000h
s_w equ (s-s)/2
d   times s_w db -1
```



AX = AB12h

ESI = s+1 00401001h

s = 00401000h

d = 0040100Ah

ESI = 00401001
00401003
00401005

---

string instructions

strings
- (pointer) offset to the first elem. of the string (ESI, EDI)
- type of each elem. (_)
- nr. of elem. (ECX)
- parsing direction (→, ←) (DF) 0 → / 1 ←

- instructions for data transfer

LODS_ (_ = B, W, D)

LODSB (load string of bytes)

AL ← <DS:ESI>

if DF=0 inc (ESI) else dec (ESI)

LODSW AX ← <DS:ESI>

if DF = 0 ESI = ESI+2

LODSW AX ← <DS:ESI>
if DF < 0 ESI = ESI + 2
       1 ESI = ESI − 2

LODSD EAX ← <DS:ESI>
DF < 0 ESI = ESI + 4
     1 ESI = ESI − 4

STOS_ (B, W, D)

STOSB (store string of bytes)
<ES:EDI> ← AL
DF < 0 EDI ++
     1 EDI − −

STOSW
<ES:EDI> ← AX
DF = < 0 EDI = EDI + 2
     1 EDI = EDI − 2

MOVS_ (B, W, D)

MOVSB (move string of bytes)

MOVSW

MOVSD

<ES:EDI> ← <DS:ESI>
if DF < 0 ESI ++, EDI ++
     1 ESI − −, EDI − −

- string instr. for data comparisons

SCAS_ (B, W, D)

SCASB (scan string of bytes)

CMP AL, <ES:EDI>
if DF < 0 EDI ++
     1 EDI − −

CMPS_ (B, W, D)

CMPSW (compare strings of words)

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

DF — 0: ESI = ESI + 2, EDI = EDI + 2

1: ESI = ESI - 2, EDI = EDI - 2

---

string of bytes → mirror the string of bytes

s db 17, 20, 42h, 1, 10, 2Ah

↓

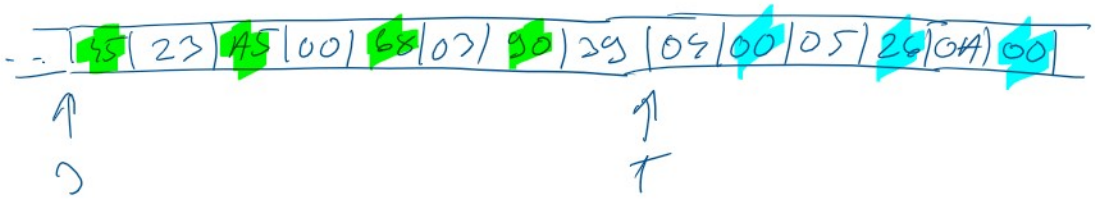
d db 2Ah, 10, 1, 42h, 20, 17

---

two string of words, concatenate the string of low bytes of the words from the first string to the string of high bytes of the words from the second string. Sort the resulting string in ascending order (signed interpretation).

s dw 2345h, 0A5h, 3685h, 3990h

t dw 4, 2655h, 10



45, 45, 68, 90, 00, 26, 00

↓

| 90, 45, 00, 00, 26 45, 68 |