

## **Hoofdstuk 3**

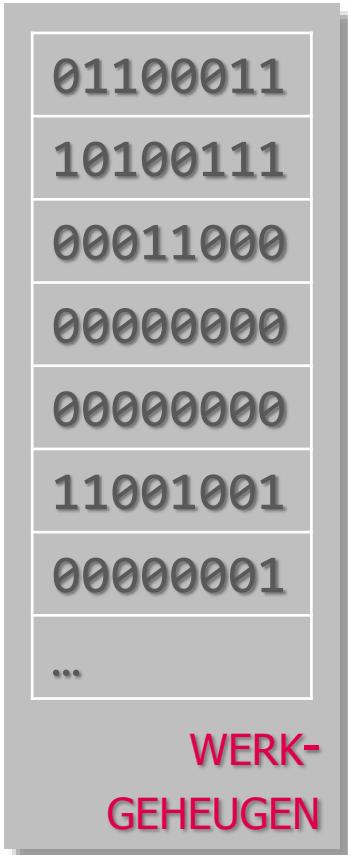
### Adressering



## H3: Adressering

- Hoeveel bits?
- Basisadres, verplaatsing
- Segment
- Adreswijziging
- Karakterstrings

# Adreslengte



- Het werkgeheugen is opgedeeld in geheugencellen
  - Bvb. Van 8 bits
  - Kan ook andere grootte hebben (19 bits, 32 bits, 60 bits, ...)
  - Grootte van de cel: **resolutie** van het geheugen
- Elke geheugencel heeft een uniek **adres** en een **inhoud**
  - Lengte van een adres bepaalt het aantal cellen

# Basis en Verplaatsing

- Programma's werken met **relatieve adressen**
  - absoluut adres = basis + verplaatsing
- Machinebevelen werken enkel met verplaatsingen
  - Bvb. `mov ebx, [2DA4h]`  
*is "kopieer de vier bytes uit het werkgeheugen met verplaatsing (2DA4)<sub>h</sub> t.o.v. de basis naar register ebx"*
- Tijdens uitvoering van het bevel wordt het (absolute) adres bepaald



bevelregister

kopieer het dubbelwoord met verplaatsing |00 00 2D A4| naar EBX

basisadres

CD2FA080

+

CPU

CD2FA080→

→ CD2FCE24→

WERKGEHEUGEN

**Merk op:** deze berekening gebeurt voor elk bevel waar een adres in gebruikt wordt!

# Segmenten

- Een aantal opeenvolgende bytes die (logisch) bij elkaar hoort heet een **segment**
- Segment-adres
  - Basis van een segment (adres van de eerste byte)
- **Codesegment** (bevelen), **datasegment** (data), **stapelsegment**, **extrasegment**, ...

# Segmenten

- Veel programma's in geheugen  
→ Veel segmenten in geheugen

**Merk op:** *de totale grootte van alle programma's samen kan groter dan het werkgeheugen zijn. Inactieve segmenten kunnen tijdelijk naar de schijf gekopieerd worden.*

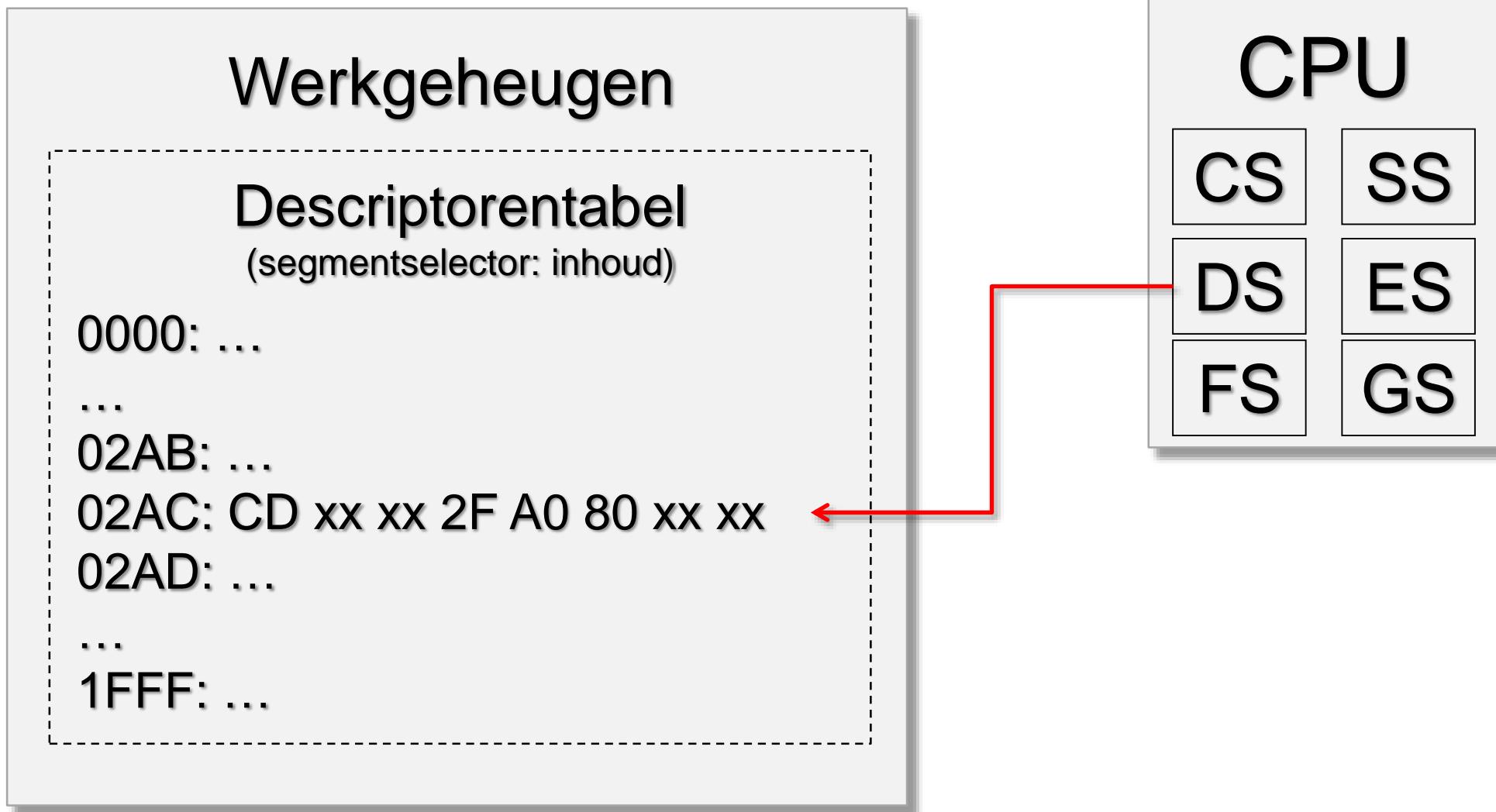
# Segmentdescriptor

- Elk segment heeft een **descriptor**
  - Bevat eigenschappen van het segment
    - Grootte
    - Staat in werkgeheugen?
    - Basisadres
    - ...
  - 8 bytes groot
  - Worden opgeslagen in de **descriptorentabel**
    - Plaats voor 8192 descriptoren

# Segmentselector

- Elke descriptor heeft een uniek nummer van  $(0000)_h$  tot  $(1FFF)_h$ , de **segmentselector**
  - Wordt bewaard in **segmentregisters** in de CPU
    - Intel x86-processor heeft 6 segmentregisters (CS, DS, SS, ES, FS, GS) van 2 bytes groot
  - In segmentregister staat een segmentselector.

# mov ebx, [hulpd]



# Segmenten

- Notatie: een segment met verplaatsing **00002DA4** en segmentselector **02AC** wordt genoteerd als **02AC:00002DA4**
- Waar staat het basisadres? In een **descriptor**.
- Waar staat die descriptor? In de **descriptorentabel** op de locatie aangegeven door de **segmentselector**.

**Merk op:** *de descriptorentabel wordt bijgehouden in de cache van de CPU, zodat dit proces geen grote vertraging veroorzaakt.*

# Industriële Archeologie

- De Intel 8086 CPU had een maximaal werkgeheugen van **1MB**
  - Adressen zijn **20-bits**
  - Verplaatsingen zijn **16-bits**
- De basis werd rechtstreeks opgeslagen in de segmentregisters
  - **CS** voor de bevelen, **DS** voor de data, ...
  - Geen segmentselectors/descriptoren/descriptorentabel

# Industriële Archeologie

- **Vraag:** Hoe kan een basis van 20-bits in een register van 16-bits?
  - Door slim te zijn. We eisen dat elk basisadres deelbaar is door 16!
- Deelbaar door 16 → laatste 4 bits 0
  - Enkel eerste 16 bits worden opgeslagen in de segmentregisters

# Industriële Archeologie

- Vb.: inhoud DS = 3A2C; byte met verplaatsing 12E8 heeft adres:

$$\begin{array}{r} 3A2C0 \\ + 12E8 \\ \hline 3B5A8 \end{array}$$

**Weetje:** *Deze manier van werken wordt nog altijd ondersteund. Wanneer een computer opgestart wordt, start hij in deze modus (= real mode). Later kan dan overgestapt worden naar de nieuwe modus (= protected mode).*

# Real vs. Protected

## Real Mode

- 20-bit adressen
- Basis in segmentregister

1A2C:12E8 betekent

$$\begin{array}{r} 1A2C0 \\ + \quad 12E8 \\ \hline 1B5A8 \end{array}$$

## Protected Mode

- 32-bit adressen
- Basis in segmentdescriptor

1A2C:12E8 betekent

Basis in descriptor **1A2C**

$$\begin{array}{r} CD2DA080 \\ + \quad 000012E8 \\ \hline CD2DB368 \end{array}$$

## Descriptorentabel

(segmentselector: inhoud)

0000: ...

...  
1A2C: **CD xx xx 2F A0 80 xx xx**

...  
1FFF: ...



## H3: Adressering

- Hoeveel bits?
- Basisadres, verplaatsing
- Segment
- **Adreswijziging**
- Karakterstrings

# Vb. Zonder Adreswijziging

- Schrijf een programma dat 10 getallen inleest, en het gemiddelde ervan toont.

```
som = 0;  
teller = 10;  
while (teller > 0) {  
    lees getal;  
    som = som + getal;  
    teller = teller - 1;  
}  
toon som / 10;
```

# Vb. Zonder Adreswijziging

```
%include "gt.asm"
covar
...
inleiding
    sub eax, eax      ; som in eax
    mov ecx, [tien]   ; teller in ecx
hoger: cmp ecx, 0
       jle verder
       inv [getal]
       add eax, [getal]
       sub ecx, 1
       jmp hoger
verder: imul dword [een]
        idiv dword [tien]
        mov [gem], eax
        uit [gem]
slot
```

INVOER	UITVOER
1	
12	8
6	
14	
2	
8	
18	
9	
9	
5	

# Vb. Met Adreswijziging

- Schrijf een programma dat 10 getallen inleest, en het gemiddelde ervan toont. **Toon daarna ook de verschillen tussen de ingelezen getallen en het gemiddelde.**
  - Lus 1: getallen inlezen, som bijhouden, en bewaren
  - Lus 2: de verschillen berekenen en tonen

# Meerdere Getallen

- In het programma plaats reserveren voor 10 getallen kan als volgt:

getal: resd 10

Resultaat:



# Getallen Inlezen

- Inlezen doen we met inv  
**inv [???**
- Dit inv-bevel staat in een lus
  - 1° keer: [???] = [getal]
  - 2° keer: [???] = [getal+4]
  - 3° keer: [???] = [getal+8]
  - Het adres moet gewijzigd worden.



# Indexregisters

- Adressen kunnen gewijzigd worden d.m.v. **indexregisters**
  - Bijv. registers **ESI** en **EDI**
- **inv [getal + edi]**
  - Berekend adres is: getal + (inhoud EDI)
  - De programmeur bepaalt de inhoud van de indexregisters

# Vb. Mét Adreswijziging

```
...
sub eax, eax
mov ecx, 10
sub edi, edi
hoger: cmp ecx, 0
       jle verder
       inv [getal + edi]
       add eax, [getal + edi]
       add edi, 4
       sub ecx, 1
       jmp hoger
verder: ...
```

...

```
sub eax, eax
mov ecx, 10
sub edi, edi
```

hoger: cmp ecx, 0

```
jle verder
inv [getal + edi]
add eax, [getal + edi]
add edi, 4
sub ecx, 1
jmp hoger
```

verder: ...

ECX

00	00	00	0A
----	----	----	----

EDI

00	00	00	00
----	----	----	----

ADRES = GETAL + 0



...

```
sub eax, eax
mov ecx, 10
sub edi, edi
```

hoger: cmp ecx, 0

```
jle verder
inv [getal + edi]
add eax, [getal + edi]
add edi, 4
sub ecx, 1
jmp hoger
```

verder: ...

ECX

00	00	00	0A
----	----	----	----

EDI

00	00	00	04
----	----	----	----

ADRES = GETAL + 0



...

```
sub eax, eax
mov ecx, 10
sub edi, edi
```

hoger: cmp ecx, 0

```
jle verder
inv [getal + edi]
add eax, [getal + edi]
add edi, 4
sub ecx, 1
jmp hoger
```

verder: ...

ECX

00	00	00	09
----	----	----	----

EDI

00	00	00	04
----	----	----	----

ADRES = GETAL + 0



...

```
sub eax, eax
mov ecx, 10
sub edi, edi
```

hoger: cmp ecx, 0

```
jle verder
inv [getal + edi]
add eax, [getal + edi]
add edi, 4
sub ecx, 1
jmp hoger
```

verder: ...

ADRES = GETAL + 4

ECX

00 00 00 09

EDI

00 00 00 04



...

```
sub eax, eax
mov ecx, 10
sub edi, edi
```

hoger: cmp ecx, 0

```
jle verder
inv [getal + edi]
add eax, [getal + edi]
add edi, 4
sub ecx, 1
jmp hoger
```

verder: ...

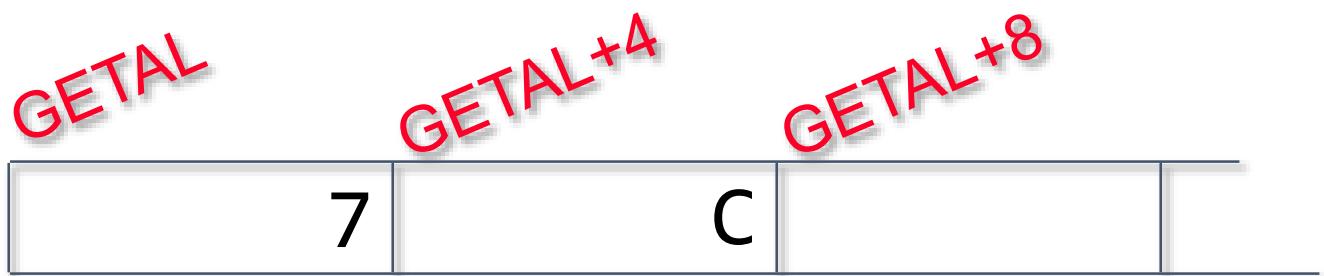
ECX

00 00 00 09

EDI

00 00 00 08

ADRES = GETAL + 4



...

```
sub eax, eax
mov ecx, 10
sub edi, edi
```

hoger: cmp ecx, 0

```
jle verder
inv [getal + edi]
add eax, [getal + edi]
add edi, 4
sub ecx, 1
jmp hoger
```

verder: ...

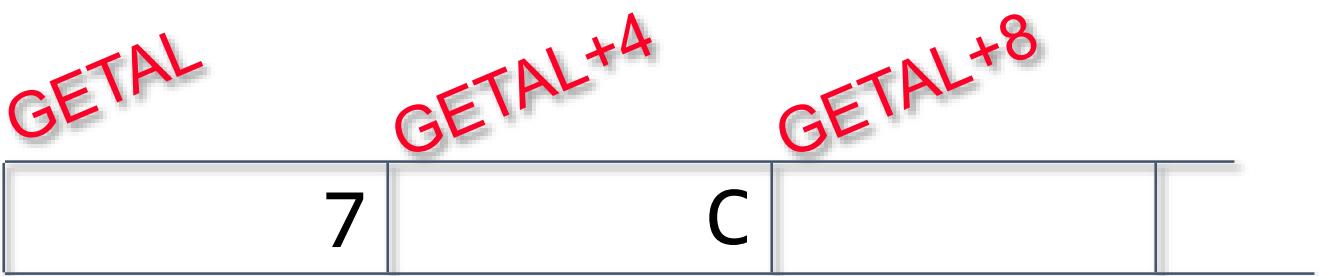
ECX

00	00	00	08
----	----	----	----

EDI

00	00	00	08
----	----	----	----

ADRES = GETAL + 4



...

```

    sub eax, eax
    mov ecx, 10
    sub edi, edi

hoger: cmp ecx, 0
        jle verder
        inv [getal + edi]
        add eax, [getal + edi]
        add edi, 4
        sub ecx, 1
        jmp hoger

verder: ...

```

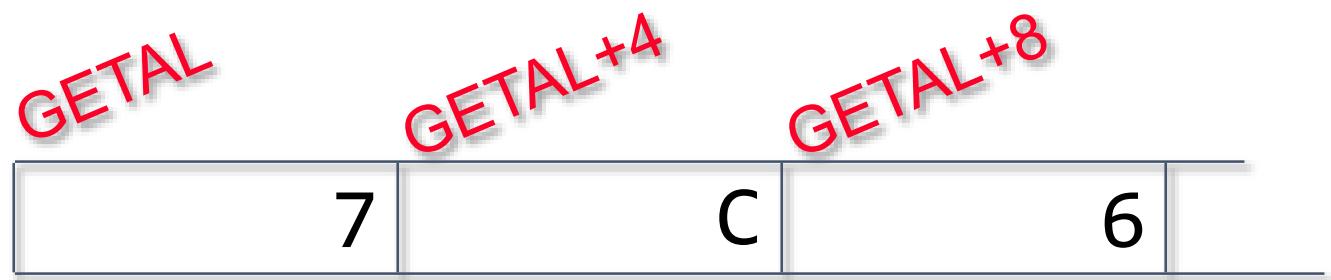
ADRES = GETAL + 8

ECX

00 00 00 08

EDI

00 00 00 08



...

```
sub eax, eax
mov ecx, 10
sub edi, edi
```

hoger: cmp ecx, 0

```
jle verder
inv [getal + edi]
add eax, [getal + edi]
add edi, 4
sub ecx, 1
jmp hoger
```

verder: ...

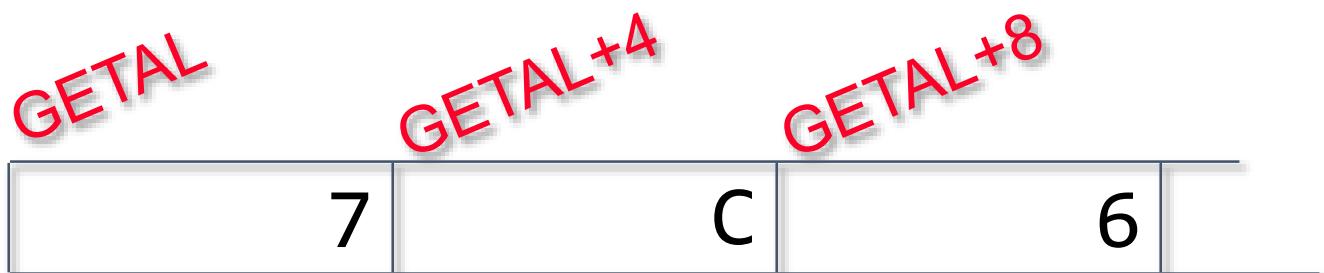
ECX

00 00 00 08

EDI

00 00 00 0C

ADRES = GETAL + 8



...

verder: **imul dword [een]**  
**idiv dword [tien]**  
**mov [gem],eax**

...

...

```
    mov ecx, 10
    sub edi, edi
lus:   cmp ecx, 0
        jle einde
        mov eax, [gem]
        sub eax, [getal + edi]
        mov [hulp], eax
        uit [hulp]
        add edi, 4
        sub ecx, 1
        jmp lus
```

- Belangrijk(st)e gegevensstructuur
  - gemiddelde temperatuur voor elke dag
  - uitslagen van één student
  - uitslagen verschillende studenten op één vak
  - ...
- In hogere programmeertaal: **array**
- Rij ≠ verzameling :
  - Orde: elk element heeft een opvolger (behalve het laatste); elk el. heeft een voorganger (uitg. 1<sup>e</sup>).

# Loop-bevel

**Opmerking:** *het loop-bevel is een zeer handig bevel om een lus te programmeren. Zie cursus blz. 74.*



## H3: Adressering

- ...
- **Karakterstrings**
  - **Stringdefinities en bevelen**
  - **Strings afdrukken**
  - **Van integer naar string**
  - **Invoerbestanden**
  - **Van string naar integer**

# Stringconstanten

- Vb. 1: antwrd: DB 'JA'
  - antwrd is het symbolisch adres van een byte
  - in het geheugen komt de ascii-waarde van de constante; d.i. **4A 41**
  - vertaalprogramma kent s.a. antwrd
  
- Vb. 2: xyz: DB 'Agent 007'
  - in het geheugen komt: **41 67 65 6E 74 20 30 30 37**

# Stringveranderlijken

- Vb. 3: gegeven: RESB 20
  - gegeven is het symbolisch adres van een byte
  - vertaalprogramma kent s.a. gegeven
  - in het geheugen komt: ?? ?? ?? ?? ... ?? ??  
(20 bytes)

# Stringdefinities

- Definities in willekeurige volgorde:

- Vb. 4:

covar

getal: RESD 1

tekst: DB 'jaja '

t: DB 'welnee '

const: DD -8

var: RESB 3

inleiding

...

slot

# Stringdefinitions

```
getal: RESD 1
tekst: DB 'jaja '
t: DB 'welnee '
const: DD -8
var: RESB 3
```

Adres:	00	04	09	10	14
	?? ?? ?? ??	6A 61 6A 61 20	77 65 6C 6E 65 65 20	FF FF FF F8	?? ?? ??
getal	tekst	t		const	var

# Stringdefinitions

- Vb. 5:

xyz: dd 46

uvw: db '46'



- $2E + 2E = 5C = (92)_d$
- $3436 + 3436 = 686C = ?$

# Stringbevelen

- x86 heeft instructies om met strings te werken
  - **movsb** – move string element byte
  - **rep movsb** – rep = repeat
  - **cld en std** – clear direction / set direction
  - **stosb** – store string element byte
  - **rep stosb**
  - **lodsb** – load string element byte

- Vb. 6:

a: resb 6

b: db ‘Jamaar’

...

cld

mov esi, b

mov edi, a

movsb

Adres: a

b



Adres: a

b



- 1 byte kopiëren
  - Van waar? **Verplaatsing in ESI.**
    - S = source
  - Naar waar? **Verplaatsing in EDI.**
    - D =destination
- Verschil met mov?
  - 4 bytes kopiëren
  - Van register naar dubbelwoord –of– van dubbelwoord naar register –of– van register naar register

- `mov esi, b`
  - De verplaatsing van symbolisch adres b wordt in register esi geplaatst.
- `mov edi, a`
  - De verplaatsing van symbolisch adres a wordt in register edi geplaatst.
- Waarom? **Nooit 2 adressen in 1 bevel.**
  - Bij mov is er altijd een register
  - Bij movsb wel twee geheugenadressen nodig

cld

mov esi, b

mov edi, a

movsb

movsb

movsb

Adres:	a	b
	4A ?? ?? ?? ?? ??	4A 61 6D 61 61 72

Adres:	a	b
	4A 61 ?? ?? ?? ??	4A 61 6D 61 61 72

Adres:	a	b
	4A 61 6D ?? ?? ??	4A 61 6D 61 61 72

```
cld
mov esi, b
mov edi, a
movsb
movsb
movsb
movsb
movsb
movsb
movsb
movsb
```

Adres: a

b

4A 61 6D 61 61 72	4A 61 6D 61 61 72
-------------------	-------------------

Hoe komt dat?

- Meestal wil men meer dan 1 byte kopiëren
  - movsb doet meer dan wat we dachten
- Van links naar rechts of rechts naar links? **Hangt af van de richtingsvlag (DF).**
  - DF=0 (na bevel CLD): van voor naar achter
  - DF=1 (na bevel STD): van achter naar voor

- Effect van movsb (volledig antwoord)
  - 1 byte kopiëren
    - Van waar? Verplaatsing in ESI.
    - Naar waar? Verplaatsing in EDI.
  - De inhoud van ESI en EDI aanpassen
    - Indien DF=0: met 1 vermeerderen
    - Indien DF=1: met 1 verminderen

# Veel bytes kopiëren

```
cld  
mov esi, b  
mov edi, a  
movsb  
movsb  
movsb  
movsb  
movsb  
movsb  
movsb
```

```
cld  
mov esi, b  
mov edi, a  
mov ecx, 6  
rep movsb
```

- Vb. 7:

```
string: resb 10
```

...

```
mov edi, string
mov al, '*'
stosb
```

Adres: **string**

?? ?? ?? ?? ?? ?? ?? ?? ?? ??

Adres: **string**

2A ?? ?? ?? ?? ?? ?? ?? ?? ??

- Effect van stosb
  - 1 byte kopiëren
    - Van waar? Inhoud van register AL.
    - Naar waar? Verplaatsing in EDI.
  - De inhoud van EDI aanpassen
    - Indien DF=0: met 1 vermeerderen
    - Indien DF=1: met 1 verminderen

# rep stosb

- Vb. 8:

```
string: resb 10
```

...

```
mov edi, string
mov al, '*'
```

```
mov ecx, 7
rep stosb
```

Adres: **string**

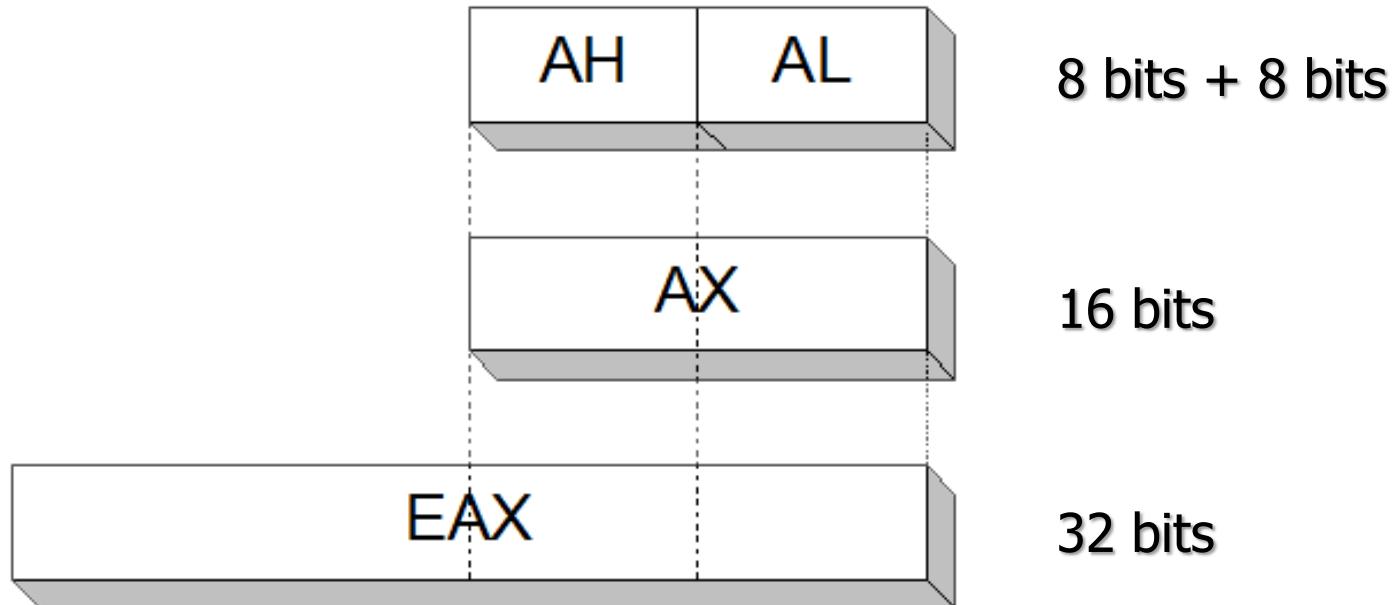
2A 2A 2A 2A 2A 2A 2A ?? ?? ??

Naar printer: : \*\*\*\*\*



# Registers

- Wat is register AL? **Een deel van EAX!**



# Registers

mov eax, 0

EAX: ~~?????????????????????????????????????????????~~

mov ax, -1

EAX: ~~0000000000000000000000000000000000000000~~

mov al, 0

EAX: ~~00000000000000001111111111111111~~

EAX: ~~00000000000000001111111100000000~~

# Deel van een string bewerken

- Vb. 9:

```
string: resb 10
```

...

```
mov edi, string + 3
mov al, '*'
mov ecx, 5
rep stosb
```

Adres: **string**

```
?? ?? ?? 2A 2A 2A 2A 2A ?? ??
```

**Opmerking:** een symbolisch adres is een getal dat ingevuld wordt door het vertaalprogramma

- Vb. 10:

```
string:DB 'jaja'
```

...

```
cld
```

```
mov esi, string
lodsb
cmp al, '
```

EAX: ?? ?? ?? ??

EAX: ?? ?? ?? 6A

# lodsb en stosb

- **stosb:** 1 byte van AL naar geheugen
  - rep stosb: dezelfde byte in AL naar geheugen
- **lodsb:** 1 byte van geheugen naar AL
  - rep lodsb: heeft geen zin
  - Wat is het effect van:

```
string: DB 'jaja'  
...  
cld  
mov ecx,4  
mov esi, string  
rep lodsb
```

**Beter:**  
mov esi, string + 3  
lodsb



## H3: Adressering

- ...
- **Karakterstrings**
  - Stringdefinities en bevelen
  - **Strings afdrukken**
  - Van integer naar string
  - Invoerbestanden
  - Van string naar integer

# Strings afdrukken

- Afdrukken van
    - tekst;
    - meerdere getallen op 1 lijn
  - Niet via uit; wel via **schrijf**.
- 
- Er werden een aantal keuzes gemaakt:
    - Uitvoer naar bestand
    - Eenheid : 1 lijn
    - Aantal tekens ligt vast : 70 (+2)
    - Adres ligt ook vast: outarea

# openuit en schrijf

- Eerst bestand creëren: **openuit**
- Dan lijn toevoegen aan bestand: **schrijf**
- Welke string?
  - Vanaf adres outarea
- Hoeveel?
  - $70 + 0D\ 0A$ ; 72 bytes dus
  - **outarea** moet gedefinieerd zijn (= de **uitvoerzone**)

# openuit en schrijf

covar

outarea: resb 70

db 0Dh, 0Ah

vb1: db 'Dit is een voorbeeld'

vb2: db 'Dit ook'

inleiding

openuit

...

- In de 70 bytes vanaf outarea komen de strings
  - Die strings worden naar een bestand gekopieerd
- De bytes vanaf outarea noemen we: uitvoerzone

# Voorbeeld: openuit en schrijf

covar

outarea: resb 70

db 0Dh, 0Ah

vb1: db 'Dit is een voorbeeld'

vb2: db 'Dit ook'

inleiding

openuit

...

Toestand na vertaling:

?? ?? ?? ?? ?? ?? ?? ?? ?? ... ?? ?? ?? 0D 0A

# Voorbeeld: openuit en schrijf

covar

outarea: resb 70  
db 0Dh, 0Ah

...

openuit

mov ecx, 70  
mov al, ''  
mov edi, outarea  
rep stosb

Uitvoerzone wordt:

20 20 20 20 20 20 20 20 20 20 ... 20 20 20 20 0D 0A

# Voorbeeld: openuit en schrijf

covar

outarea: resb 70

db 0Dh, 0Ah

vb1: db 'Dit is een voorbeeld'

vb2: db 'Dit ook'

...

mov ecx, 20

mov esi, vb1

mov edi, outarea

rep movsb

Uitvoerzone wordt:

44 69 74 20 69 73 20 65 65 6E ... 20 20 20 0D 0A

# Voorbeeld: openuit en schrijf

covar

outarea: resb 70

db 0Dh, 0Ah

vb1: db 'Dit is een voorbeeld'

vb2: db 'Dit ook'

...

**schrijf**

**Uitvoerzone:**

44 69 74 20 69 73 20 65 65 6E ... 20 20 20 0D 0A

**wordt naar een bestand gekopieerd**



# Voorbeeld: openuit en schrijf

covar

outarea: resb 70

db 0Dh, 0Ah

vb1: db 'Dit is een voorbeeld'

vb2: db 'Dit ook'

...

mov ecx,7

mov esi, vb2

mov edi, outarea

rep movsb

**Vraag:** wat is de  
inhoud van EDI?

Uitvoerzone wordt:

44 69 74 20 6F 6F 6B 65 65 6E ... 20 20 20 0D 0A

# Voorbeeld: openuit en schrijf

```
covar
outarea: resb 70
          db 0Dh, 0Ah
vb1: db 'Dit is een voorbeeld'
vb2: db 'Dit ook'
...
mov al, ' '
mov ecx, 13
rep stosb
```

*Inhoud EDI  
was nog ok!*

Uitvoerzone wordt:

44 69 74 20 6F 6F 6B 20 20 20 ... 20 20 20 0D 0A

# Voorbeeld: openuit en schrijf

covar

outarea: resb 70

db 0Dh, 0Ah

vb1: db 'Dit is een voorbeeld'

vb2: db 'Dit ook'

...

**schrijf**



**Uitvoerzone:**

44 69 74 20 6F 6F 6B 20 20 20 ... 20 20 20 0D 0A

**wordt naar een bestand gekopieerd**

# Uitvoerbestand besluit

- outarea: resb 70  
db 0Dh, 0Ah  
...  
openuit
- Wat je wil tonen eerst in **uitvoerzone** zetten
  - met **movsb**, **stosb**, enz...
- Dan **schrijf** oproepen

**Tip:** spaties in de uitvoerzone komen er niet van zelf. Je maakt ze dus best eerst blanco!



## H3: Adressering

- ...
- **Karakterstrings**
  - Stringdefinities en bevelen
  - Strings afdrukken
  - **Van integer naar string**
  - Invoerbestanden
  - **Van string naar integer**

# Van integer naar string

- Een getal heeft meer dan één voorstelling
  - Binair (of hexadecimaal) om mee te rekenen
  - ASCII om af te drukken
- $(24000)_d = (00005DC0)_h = (32\ 34\ 30\ 30\ 30)_{ASCII}$
- We moeten onze berekende resultaten dus **omzetten!**

# Omrekenen naar ASCII

Vb.: in het geheugen staat  $(00\ 00\ 05\ 16)_h$

- De decimale voorstelling is 1302
  - Zo willen wij het zien staan op het papier
  - We moeten 31 33 30 32 naar de printer sturen.
  - We moeten 31 33 30 32 in de uitvoerzone plaatsen (OUTAREA ...).
- Hoe verkrijgen we 31 33 30 32?

# Omrekenen naar ASCII

Vb.: in het geheugen staat  $(00\ 00\ 05\ 16)_h = (1302)_d$

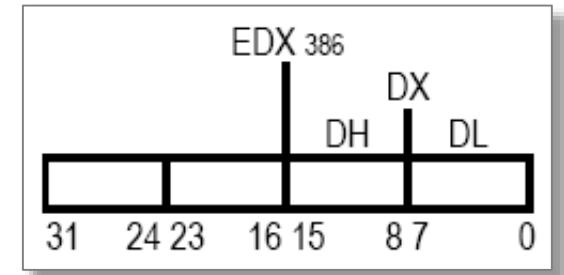
- Hoe verkrijgen we 31 33 30 32?

$$\begin{array}{r} 1302 \quad | \quad 10 \\ -2 \quad | \quad 130 \quad | \quad 10 \\ \hline 0 \quad | \quad 13 \quad | \quad 10 \\ -3 \quad | \quad 1 \quad | \quad 10 \\ \hline 1 \end{array}$$

- Dus: delen door 10 en letten op de resten

# Omrekenen naar ASCII

- We hebben 2, 0, 3, 1
  - Waar staan die getallen? **In EDX.**
    - 00 00 00 02
    - 00 00 00 00
    - 00 00 00 03
    - 00 00 00 01
  - Hoe kunnen we aan de laatste byte van EDX? **Via DL.**
- We willen 32, 30, 33, 31 (ASCII)



# Omrekenen naar ASCII

- Inh. van DL is 02, moet 32 worden;  
00, moet 30 worden;  
03, moet 33 worden;  
01, moet 31 worden.
- Via een ADD-bewerking, bvb.:  
`add dl, 30h`

$$\begin{array}{r} 0000 \ 0010 \\ + 0011 \ 0000 \\ \hline 0011 \ 0010 \end{array} \quad \begin{array}{l} (02)_h \\ (30)_h \\ (32)_h \end{array}$$

# Omrekenen naar ASCII

- Hoe inhoud van DL naar uitvoerzone kopiëren? **Met stosb!**
  - stosb kopieert enkel vanuit AL
  - Eerst inhoud van DL naar AL kopiëren, maar dan verandert EAX
- **xchg al, dl** – inhoud van AL en DL **omwisselen**
- Van achter naar voor werken met **STD**

# Omrekenen naar ASCII

```
...
std
    mov ebx, 10
lus: mov edx, 0
     idiv ebx
     add dl, 30h
     xchg al, dl
     stosb
     xchg al, dl
     cmp eax, 0
     jne lus
```



# Omrekening voorbeeld

Vb.: EAX = (00 00 05 16)<sub>h</sub> = (1302)<sub>d</sub>

...

```
    mov edi, outarea + 13  
    std  
    mov ebx, 10  
lus: mov edx, 0  
     idiv ebx  
     add dl, 30h  
     xchg al, dl  
     stosb  
     xchg al, dl  
     cmp eax, 0  
     jne lus
```

EDX	EAX
00 00 00 00	00 00 05 16
00 00 00 02	00 00 00 82
00 00 00 32	00 00 00 82
00 00 00 82	00 00 00 32

OUTAREA

EDI  
↓

Antwoord: 20 20 20 20 20 20 20 20 20 20 ... 0D 0A

# Omrekening voorbeeld

Vb.: EAX = (00 00 05 16)<sub>h</sub> = (1302)<sub>d</sub>

...

```
    mov edi, outarea + 13  
    std  
    mov ebx, 10  
lus: mov edx, 0  
     idiv ebx  
     add dl, 30h  
     xchg al, dl  
     stosb  
     xchg al, dl  
     cmp eax, 0  
     jne lus
```

EDX	EAX
00 00 00 00	00 00 05 16
00 00 00 02	00 00 00 82
00 00 00 32	00 00 00 82
00 00 00 82	00 00 00 32
00 00 00 32	00 00 00 82

= 0?

OUTAREA

EDI

Antwoord: 20 20 20 32 20 20 20 20 20 20 ... 0D 0A



# Omrekening voorbeeld

Vb.: EAX = (00 00 05 16)<sub>h</sub> = (1302)<sub>d</sub>

...

```
    mov edi, outarea + 13  
    std  
    mov ebx, 10  
lus: mov edx, 0  
     idiv ebx  
     add dl, 30h  
     xchg al, dl  
     stosb  
     xchg al, dl  
     cmp eax, 0  
     jne lus
```

EDX	EAX
00 00 00 00	00 00 00 82
00 00 00 00	00 00 00 0D
00 00 00 30	00 00 00 0D
00 00 00 0D	00 00 00 30

OUTAREA

EDI  
↓

Antwoord: 20 20 20 32 20 20 20 20 20 20 ... 0D 0A

# Omrekening voorbeeld

Vb.: EAX = (00 00 05 16)<sub>h</sub> = (1302)<sub>d</sub>

...

```
    mov edi, outarea + 13  
    std  
    mov ebx, 10  
lus:  mov edx, 0  
      idiv ebx  
      add dl, 30h  
      xchg al, dl  
      stosb  
      xchg al, dl  
      cmp eax, 0  
      jne lus
```

EDX	EAX
00 00 00 00	00 00 00 82
00 00 00 00	00 00 00 0D
00 00 00 30	00 00 00 0D
00 00 00 0D	00 00 00 30
00 00 00 30	00 00 00 0D

= 0?

OUTAREA

EDI



Antwoord: 20 20 30 32 20 20 20 20 20 20 20 20 ... 0D 0A

# Omrekening voorbeeld

Vb.: EAX = (00 00 05 16)<sub>h</sub> = (1302)<sub>d</sub>

...

mov edi, outarea + 13

std

mov ebx, 10

lus: mov edx, 0

idiv ebx

add dl, 30h

xchg al, dl

stosb

xchg al, dl

cmp eax, 0

jne lus

	EDX	EAX
	00 00 00 00	00 00 00 0D
	00 00 00 03	00 00 00 01
	00 00 00 33	00 00 00 01
	00 00 00 01	00 00 00 33

OUTAREA

EDI  
↓

Antwoord: 20 20 30 32 20 20 20 20 20 20 20 ... 0D 0A

# Omrekening voorbeeld

Vb.: EAX = (00 00 05 16)<sub>h</sub> = (1302)<sub>d</sub>

...

```
    mov edi, outarea + 13  
    std  
    mov ebx, 10  
lus: mov edx, 0  
     idiv ebx  
     add dl, 30h  
     xchg al, dl  
     stosb  
     xchg al, dl  
     cmp eax, 0  
     jne lus
```

EDX	EAX
00 00 00 00	00 00 00 0D
00 00 00 03	00 00 00 01
00 00 00 33	00 00 00 01
00 00 00 01	00 00 00 33
00 00 00 33	00 00 00 01

= 0?

OUTAREA      EDI  
↓

Antwoord: 20 33 30 32 20 20 20 20 20 20 ... 0D 0A

# Omrekening voorbeeld

Vb.: EAX = (00 00 05 16)<sub>h</sub> = (1302)<sub>d</sub>

...

```
    mov edi, outarea + 13  
    std  
    mov ebx, 10  
lus: mov edx, 0  
     idiv ebx  
     add dl, 30h  
     xchg al, dl  
     stosb  
     xchg al, dl  
     cmp eax, 0  
     jne lus
```

	EDX	EAX
	00 00 00 00	00 00 00 01
	00 00 00 01	00 00 00 00
	00 00 00 31	00 00 00 00
	00 00 00 00	00 00 00 31

OUTAREA    EDI  
↓

Antwoord: 20 33 30 32 20 20 20 20 20 20 ... 0D 0A

# Omrekening voorbeeld

Vb.: EAX = (00 00 05 16)<sub>h</sub> = (1302)<sub>d</sub>

...

```
    mov edi, outarea + 13  
    std  
    mov ebx, 10  
lus: mov edx, 0  
     idiv ebx  
     add dl, 30h  
     xchg al, dl  
     stosb  
     xchg al, dl  
     cmp eax, 0  
     jne lus
```

EDX	EAX
00 00 00 00	00 00 00 01
00 00 00 01	00 00 00 00
00 00 00 31	00 00 00 00
00 00 00 00	00 00 00 31
00 00 00 31	00 00 00 00

= 0?

OUTAREA<sup>EDI</sup>  
↓

Antwoord: 31 33 30 32 20 20 20 20 20 20 ... 0D 0A

# Omrekenen naar ASCII

- Besluit: afdrukken van inhoud EAX altijd zo:

```
    mov edi, outarea + ??  
    std  
    mov ebx, 10  
lus: mov edx, 0  
    idiv ebx  
    add dl, 30h  
    xchg al, dl  
    stosb  
    xchg al, dl  
    cmp eax, 0  
    jne lus
```

*??: waar moet het laatste cijfer komen?*



## H3: Adressering

- ...
- **Karakterstrings**
  - Stringdefinities en bevelen
  - Strings afdrukken
  - Van integer naar string
  - **Invoerbestanden**
  - Van string naar integer

# Structuur van een bestand

- Vb.:

kol 1	kol 31	kol 41	kol 70
↓	↓	↓	↓
PIET JANSEN ...	180 ... 0 ... ↲		
JAN AHA ...	20000 ... 400 ... ↲		
...			

- Bestand bestaat uit **records**
  - Iets dat logisch samenhoort
  - Alle records even lang, max. 70 tekens

# Een bestand lezen

- In het programma een **invoerzone** definiëren  
**inarea: resb 70**
- Bestand openen met **openin**
- Een lijn lezen met **lees**
- Het besturingssysteem houdt een **bestandswijzer** bij
  - = het aantal bytes dat al gekopieerd is
  - Begint bij 0

# Een bestand lezen

- Vb.: **inarea: resb 70**  
...  
**lees**
- Kopieer bytes van invoerbestand naar werkgeheugen
  - Naar waar? **Inarea**.
  - Welke bytes? **Vanaf bestandswijzer tot vóór 0D0A.**
    - Eventueel aangevuld met spaties
- Bestandswijzer **verhogen** met
  - Aantal gekopieerde bytes
  - +2 (omwille van 0D0A)

# Voorbeeld

- Vóór de uitvoering van lees

INAREA

?? ?? ?? ?? ...

... ??

70 bytes

Bestand:

51 49 45 55 ... 20 31 38 ... 30 20... 20 0D 0A

48 bytes

- Na de uitvoering van lees

INAREA

51 49 45 55 ... 20 31 38 ... 30 20 ...

... 20

70 bytes

# Een bestand verwerken

Vb.:

inarea: resb 70

...

hoger:

lees

... ;verwerking van record

jmp hoger

...

slot

**Probleem:** *wat gebeurt er als het bestand op is? Wanneer stopt dit programma?*

# Een bestand verwerken

- Wanneer moeten we stoppen? **Als EAX na lees 0 bevat.**
  - Lees steekt in EAX het aantal uitgelezen bytes
  - Bestand op  $\Rightarrow$  EAX = 0

# Een bestand verwerken

Vb.:

inarea:	resb 70
hoger:	...
	lees
	cmp eax,0
	je eof
	... ; verwerking record
	jmp hoger
eof:	... ; eindverwerking
	slot



## H3: Adressering

- ...
- **Karakterstrings**
  - Stringdefinities en bevelen
  - Strings afdrukken
  - Van integer naar string
  - Invoerbestanden
  - **Van string naar integer**

# Invoerbestand: alles is ASCII

- Vb.: in test.in heeft men ingetypt:

1234  67

- Dan staat er op de schijf (in ASCII):

31 32 33 34 20 20 36 37

- Zo optellen heeft geen zin

- $(31323334)_h + (20203637)_h = (5152696B)_h = (QRik)_{ASCII}$

# ASCII omrekenen

- Vb.:      **hulp:**      **resd 1**  
                  ...  
**hoger:**      ...  
                  **lees**  
**mov esi, inarea**  
**mov ecx, 4**  
**tekstbin**  
**mov [hulp], eax**  
**mov esi, inarea + 4**  
**mov ecx, 4**  
**tekstbin**  
**add eax, [hulp]**

# Tekstbin: van tekst naar binair

- Vooraf:
  - Verplaatsing 1<sup>e</sup> byte in ESI
  - Aantal bytes in ECX
- Dan **tekstbin**
- Welk effect? De binaire voorstelling komt in EAX.

# Tekstbin: van tekst naar binair

- Vb.:  
`mov esi, inarea`  
`mov ecx, 4`  
`tekstbin`
- De inhoud van de 4 bytes vanaf inarea (d.i. : 31 32 33 34) wordt omgerekend
  - Inhoud EAX: 00 00 04 D2
- Zelf doen? **Veel werk!**

# Tekstbin: van tekst naar binair

- Enkel de ASCII-waardes van getallen (30 t.e.m. 39) mogen in het getal voorkomen
  - Uitzondering: spaties vooraan mogen
- Het resultaat moet in EAX kunnen