Intel x86

Microsoft Macro Assembler (MASM32)

Sisukord

- Intel x86 arhitektuur
  - Inteli arhitektuuride areng
  - Arhitektuuri lühiülevaade
- Microsoft Macro Assembler
  - Lühiülevaade, kompileerimine, Hello World
  - Segmendid, Pinu, Põhikäsud
  - Hargnemised – loogika juhtimine
  - Kokkuvõte
  - Näited

Intel x86 arhitektuur

1978 – Intel 8086 16-bit
1982 – Intel 80286 „Protected Mode“

1985 – Intel 80386, IA-32
1989 – Intel 80486 Integreeritud FPU
1996 – Intel 80586 „Pentium MMX“
1993 – Intel 80686 Pentium III SSE

2001 – Intel Itanium, IA-64

Intel x86 arhitektuur

- Protsessori käivitusseaded
  - Real mode *(16-bit)*, 1MB reaalmälu, 20-bit aadressid
  - Protected mode *(16/32-bit)*, ~4GB virtuaalmälu
  - Long mode *(64-bit)*, 48-bit virtuaal- ja 40-bit füüsilised aadressid
  - Virtual 8086 mode *(virtual real mode)*
  - System Management mode *(debug mode)*

Intel x86 arhitektuur

- CISC – Complex instruction set computing
  - Disainitud kompilaatorite jaoks
  - Vähem opkoode
  - Lihtsam assembler
  - Arvutused toimuvad registrites
  - Virtualiseeritud *(kaitstud)* mälu

Intel x86 arhitektuur

Opkoodi formaat:

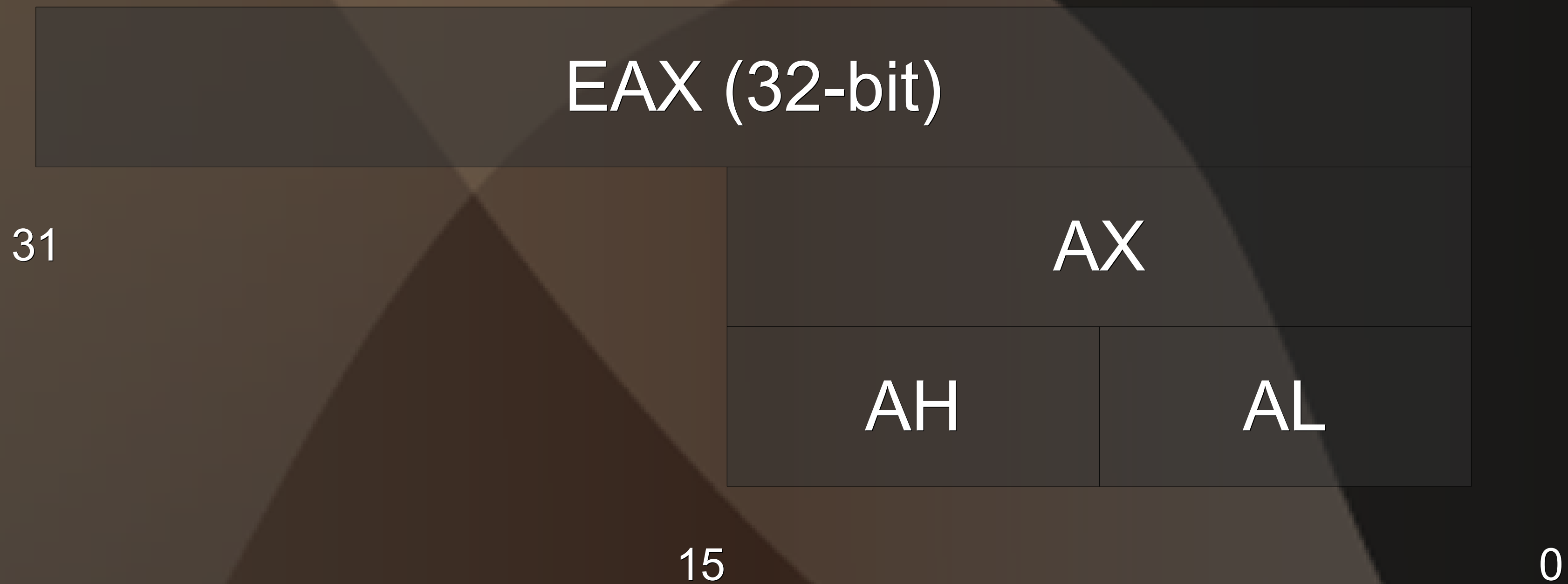
PREFIKS	0-4 baiti
OPKOOD	1-2 baiti
MOD REG/MEM	0-1 baiti
SRC INDEX BASE	0-1 baiti
DISPLACEMENT	0-4 baiti
IMMEDIATE	0-4 baiti

Intel x86 arhitektuur

- Laiotstarbelised Registrid
  - EAX – akumulaator
  - EBX – akumulaator
  - ECX – akumulaator
  - EDX – akumulaator
  - ESI – lähteindeks „source index“
  - EDI – siirdeindeks „destination index“
  - EBP – raamiviit „frame pointer“
  - ESP – pinuviit „stack pointer“

Intel x86 arhitektuur

- Register



Intel x86 arhitektuur

- Inteli assembler
  - Lihtne süntaks
  - Lühem assembleri kood
  - Palju erinvaid variatsioonidega opkoode
  - Suur hulk vabu registreid *(MMX, SSE)*

```
mov    esi, [esp – 8]
mov    eax, dword ptr[esi + ecx*4]
mov    ebx, eax
mov    eax, [eax]
```

MASM - Ülevaade

- Microsoft © Macro Assembler
  - Populaarseim Makroassembler aastast 1981
  - Versioon 1.0 – 5.1 MS-DOS *(masm.exe)*
  - Versioon 6.0 – 11.0 VisualC++ SDK *(ml.exe)*
  - Masm32.com *(v8.0)*

MASM – Kompileerimine

- Kompileerimine

```
> ml  /c  /coff   test.asm
```

  - /c – ainult kompileeri *(ei lingi)*
  - /coff – kasuta Common Object File formaati
  - test.asm – lähtefail

MASM – Kompileerimine

- Linkimine

```
> link  /entry:main  /subsystem:console  test.obj
```

  - /entry – Sümbol kust programm algab
  - /subsystem
    - :windows - tavaline windowsi protsess
    - :console - käivitab programmi olemasolevas *(kus programm käivitati)* või uues konsoolis
  - test.obj – sisend objektfail*(id)*

MASM – Kompileerimine

- Kompileeri ja lingi

```
> ml  /c  /coff  *.asm
> link  /entry:main  /subsystem:console /out:test.exe  *.obj
```

MASM – Kompileerimine

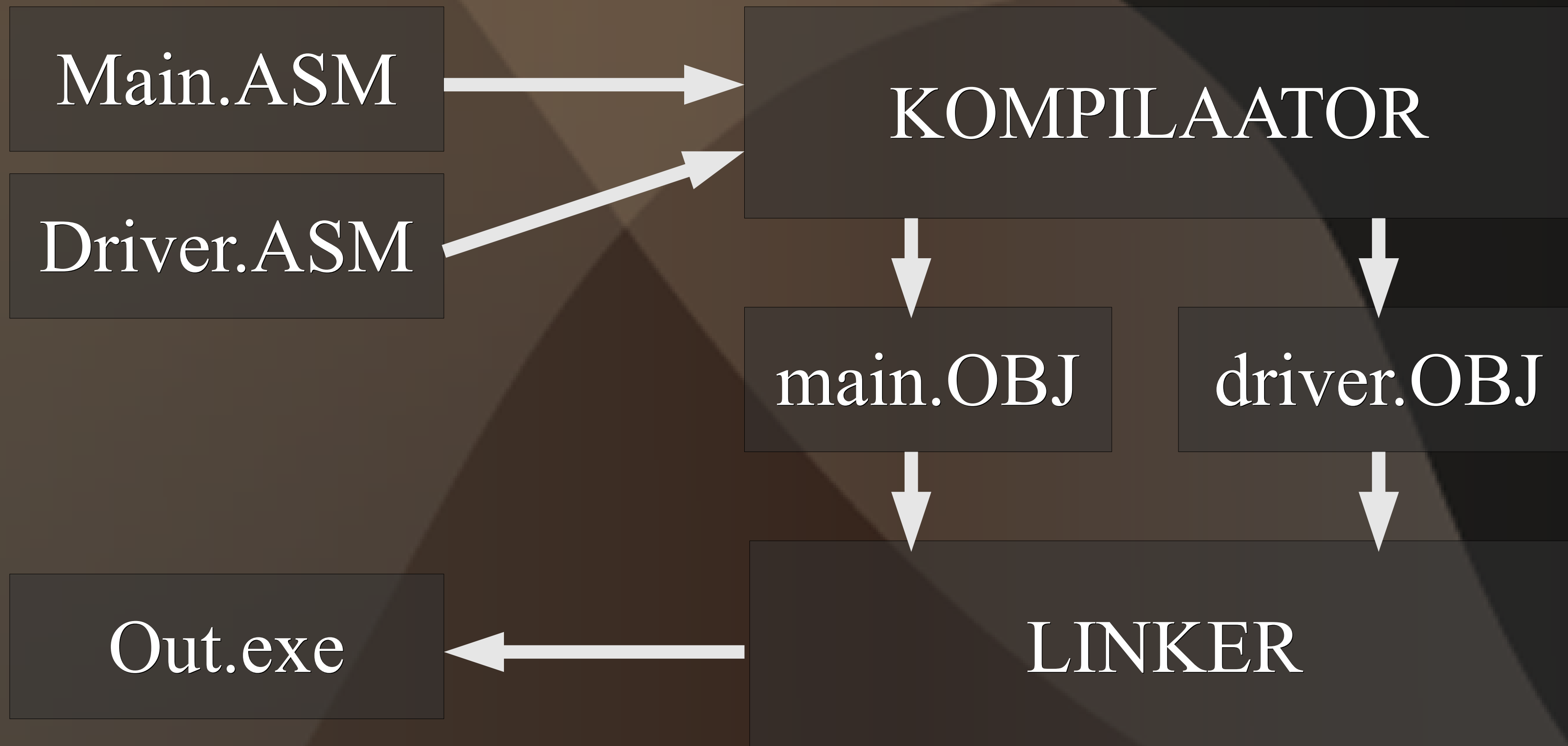
- Kompileeri ja lingi

```
> ml  /Bl \masm32\bin\link  /Fe test.exe  /coff
*.asm  /link  /entry:main  /subsystem:console
```

  - /Bl – määra õige linker
  - /Fe – määra väljundfail
  - /link – suuna järgnevad käsud linkerile

MASM – Kompileerimine

- Kompileerimise protsess



MASM – Hello World

- Lihtne programm „hello.asm“

```
1  .386                          ; arhitektuur
2  .model flat, stdcall          ; mudel 32-bit, stdcall fn-id
3  option casemap:none           ; case sensitive
4  include     \masm32\include\kernel32.inc
5  includelib  \masm32\lib\kernel32.lib
6  .data
7      hello byte 'Hello world!'
8      input byte 64 DUP(?)
9  .code
10 main proc
11     LOCAL n:dword
12     invoke WriteFile,7,addr hello,12,addr n,0
13     invoke ReadFile,3,addr input,64,addr n,0
14     ret
15 main endp
16 END;
```

MASM – Hello World

- Programmi selgroog

```
.386
.model flat, stdcall

.data  ; data segment
    A dword 10
    B byte 'B'

.code  ; text segment

main proc ; funktsioon main()

    ret
main endp


END; programmiteksti lõpp
```

MASM – Segmendid

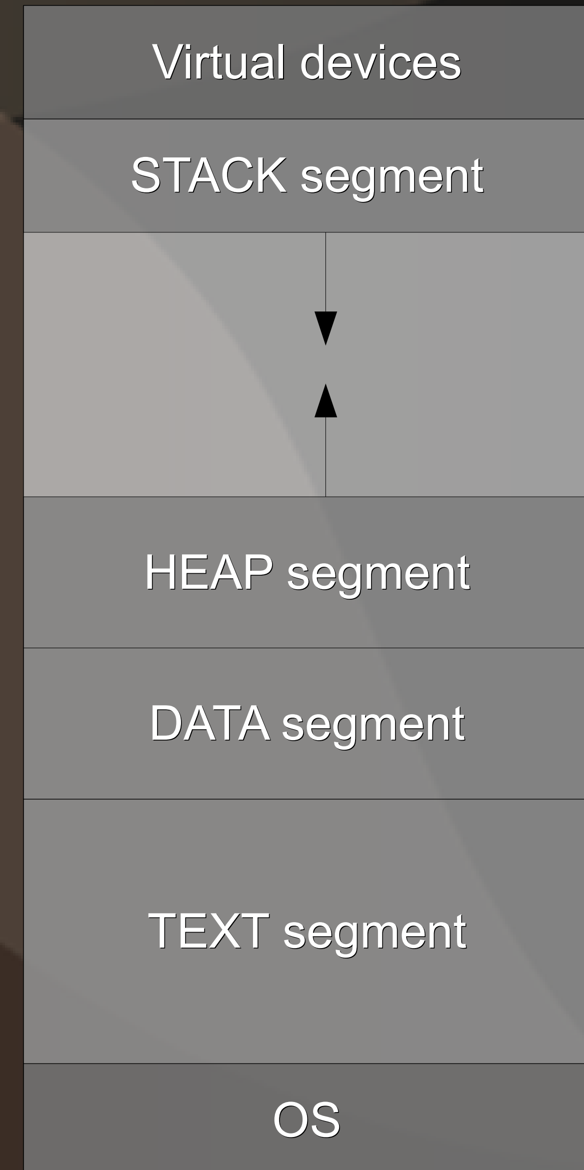
- Andmesegmendid mälus

  TEXT – programmi kood

  DATA – staatilised andmed

  STACK – funktsiooni pinu

  HEAP – dünaamiline mälu



MASM – Segmendid

- DATA segment
  - Staatilised andmed
  - Andmetüübid:
    - byte, word, dword – tavalised muutujad
    - qword, xmmword – 64 & 128 bit tehted
    - real4, real8 – floating point
  - Deklareerimine:
    - <muutuja> <tüüp> <väärtus/?>

MASM – Segmendid

- DATA segment

```
.386
.model flat, stdcall
option casemap:none


.data
    a   dword  10                   ; täisarv
    b   dword  10,20,30,40,50       ; massiiv[5] täisarve
    c   byte   10  DUP('e')         ; massiiv[10] 'eeeeeeeeee'

.data?
    d   dword  ?                 ; väärtustamata täisarv
    e   byte   64  DUP(?)        ; reserveeritud massiv[64]

```

MASM – Segmendid

- DATA segment
  - Tüübid määravad AINULT andmete suuruse mälus!
  - Programmeerija saab luua oma „struktuure“:

```
Vector2 struct
    x dword ?
    y dword ?
Vector2 ends

.data
    v1 Vector2 {0,0} ; [0,0]
.data?
    v2 Vector2 {}    ; [?,?]
```

MASM – Segmendid

- DATA segment
  - Keerulisem struktuur:

```
Line struct
    a Vector2 {}
    b Vector2 {}
Line ends

.data
    l1 Line {{0,0}, {10,10}} ; [0,0], [10,10]
.data?
    l2 Line {}               ; [?,?], [?,?]
```

MASM – Segmendid

- TEXT segment

  Kõik kompileeritud masinkood

```
.code

include     \masm32\include\kernel32.inc
includelib  \masm32\lib\kernel32.lib

; write src[len] to console
write proc src:ptr byte, len:dword
    invoke WriteFile,7,src,len,addr len,0
    ret
write endp

main proc
    .data
        hello byte "Hello World!"
    .code

    ; write "Hello World!"
    invoke write, addr hello, 12

    ret
main endp
```

MASM – Segmendid

- STACK segment

  Funktsioonide lokaalsed muutujad

```
add2 proc a:dword, b:dword
    mov eax, a   ; eax = *a
    add eax, b   ; eax += *b
    ret          ; return eax
add2 endp

main proc
    LOCAL x:dword       ; local int
    invoke add2, 6, 4 ; add2(6,4)
    mov x, eax          ; x = add2
    ret
main endp
```

High Address

ESP

STACK
LOCAL x:dword
arg1: 4
arg0: 6
return address
save frame pointer

Low Address

MASM – Segmendid

- Automaatselt genereeritud fn-i proloog/epiloog

```
func1 proc stdcall a:dword, b:dword
   mov  eax, a
   add  eax, b
   ret
func1 endp
```

MASM – Segmendid

- Käsitsi kirjutatud proloog/epiloog

```
func2:
   ; ------- prologue ------
   push  ebp       ; save frame ptr
   mov   ebp, esp ; new frame ptr
   ;sub    esp, 0    ; make room for locals (no locals)
   ; ----------------------


   mov   eax, dword ptr[ebp + 8]   ; arg0
   add   eax, dword ptr[ebp + 12] ; arg1


   ; ------- epilogue -------
   mov   esp, ebp ; restore stack ptr
   pop   ebp       ; load frame ptr
   ret 8
   ; ----------------------
```

MASM – Käsud

- MOV – Liiguta andmeid mälu ja registrite vahel

```
mov   eax, 10h         ; immediate value
mov   eax, ebx         ; register to register
mov   eax, [6df80h]    ; immediate addressing
mov   eax, [eax]       ; indirect addressing
mov   eax, [eax-4]     ; indirect with offset
mov   eax, [esi+ecx*4]; SoureIndexBase addressing

; assume ebx:ptr Vector2
mov   [ebx].Vector2.x, 0  ; v.x = 0
mov   [ebx].Vector2.y, 0  ; v.y = 0
mov   (Vector2 ptr [ebx]).x, 0
```

MASM – Käsud

- ADD, SUB, INC, DEC, DIV, MUL, AND, OR, XOR – lihtsad operatsioonid täisarvudega

```
add   eax, 1     ; eax += 1
sub   eax, ebx   ; eax -= ebx
inc   eax        ; ++eax
and   eax, ebx   ; eax &= ebx
or    ebx, eax   ; ebx |= eax

imul  ebx           ; eax *= ebx
imul  eax, ebx      ; eax *= ebx
imul  eax, ebx, 4 ; eax = ebx * 4

xor   edx, edx ; edx = 0, avoids int 0!
div   ebx        ; eax /= ebx
```

MASM – Käsud

- PUSH, POP, CALL, INVOKE

```
local x:dword
mov   x, 4


push  5
push  x
call  add2
add   esp, 8 ; 2x pop
mov   x, eax ; x = add2(x, 5)



invoke add2, x, 5
mov   x, eax ; x = add2(x, 5)
```

MASM – Käsud

- MACRO – makroassembleri põhivõlu
  - Lihtlabane tekstitöötlus
  - Võimaldab keerulisi ja tüütuid tegevusi lihtsustada

```
add2m MACRO a, b
    mov eax, a
    add eax, b
    EXITM<eax>
ENDM

    ; x = add2(x, 5)
    mov   x, add2m(x, 5)
```

MASM – Käsud

- Floating point käsud
- FLD – float load
- FST(P) – float store *(pop)*
- FADD, FSUB, FMUL, FDIV, etc.
- FILD – float load int
- FIST(P) – float store as int (pop)
- FSQRT – hardware sqrt

```
LOCAL f0:real4 ; float
LOCAL f1:real4 ; float

fld   f0        ; st[0] := f0
fstp  f1        ; [f1]  := pop(st[0])

fld   f0        ; st[0] := f0
fadd  f1        ; st[0] += f1
fstp  f1        ; f1 := pop(st[0])

fld   f1        ;
fsqrt           ;
fstp  f1        ; f1 = sqrt(f1)

fild  i         ; st[0] := (float)i
fadd  f0        ; st[0] += f0
fistp i         ; i := (int)st[0]
```

MASM – Käsud

- MMX ja SSE
- MMX:
  - .586 & .MMX
- SSE:
  - .686 & .XMM

```
.686
.XMM
.model flat, C
option casemap:none
.code
main proc
    LOCAL x:qword
    align 16
    LOCAL y:xmmword
    LOCAL z:xmmword

    movq mm0, x   ; auto-enter mmx mode
    emms          ; exit mmx mode

    movdqa xmm0, y
    movdqa xmm1, z
    addps  xmm0, xmm1 ; 4x packed float add
    movdqa z, xmm0    ; flush to mem

    ret
main endp
```

MASM – Loogika

- Branches – haruloogika opkoodid

```
label:
    dec eax    ; --eax

    je  label ; jump if equal
    jne label ; jump if not equal
    jz  label ; jump if zero
    jg  label ; jump if greater          (signed)
    jge label ; jump if greater or equal (signed)
    jl  label ; jump if less             (signed)
    jle label ; jump if less or equal    (signed)
    ja  label ; jump if above          (unsigned)
    jae label ; jump if above or equal (unsigned)
    jb  label ; jump if below          (unsigned)
    jbe label ; jump if below or equal (unsigned)
    jmp label ; unconditional jump
```

MASM – Loogika

- Branches – haruloogika makrokäsud

```
.if eax == 0
    ; 0
.elseif eax == 1
    ; 1
.else
    ; ?
.endif

.while eax != 0
    ; loop
.endw

.repeat
    ; loop
.until eax == 0
```

MASM – Kokkuvõte

- MASM32 – Kuidas kasutada?
  - Kiired optimeeritud alamprogrammid. Nt: strlen, strcmp, memcpy, jne.
  - Aitab paremini programmeerida
    - Mõistad mis assembleri koodi iga C reaga tekitatakse
  - Mõnikord kompilaator lihtsalt ei saa hakkama

MASM – Näiteid

- „Naiivne“ strlen

```
strlen proc C src:ptr byte
   xor   eax, eax  ; n := 0
   mov   edx, src  ; src
loop0:
   mov   cl, [edx]
   test  cl, 0       ; *src == 0 ?
   jz    end0        ;   go to end
   inc   eax         ; ++n
   inc   edx         ; ++src
   jmp   loop0       ; goto sloop
end0:
   ret               ; return EAX (n)
strlen endp
```

MASM – Näiteid

- Optimeeritud strlen

```
strlen  proc buf:ptr byte
        OPTION PROLOGUE:NONE, EPILOGUE:NONE
        .FPO    ( 0, 1, 0, 0, 0, 0 )
string  equ     [esp + 4]
        mov     ecx,string ; ecx -> string
        test    ecx,3         ; test if string is aligned
        je      short main_loop
str_misaligned:
        ; simple byte loop until string is aligned
        mov     al,byte ptr [ecx]
        add     ecx,1
        test    al,al
        je      short byte_3
        test    ecx,3
        jne     short str_misaligned
        add     eax,dword ptr 0    ; 5 byte nop align
        align   16                 ; redundant
main_loop:
        mov     eax,dword ptr [ecx] ; read 4 bytes
        mov     edx,7efefeffh
        add     edx,eax
        xor     eax,-1
        xor     eax,edx
        add     ecx,4
        test    eax,81010100h
        je      short main_loop
        ; found zero byte in the loop
        mov     eax,[ecx - 4]
        test    al,al              ; is it byte 0
        je      short byte_0
        test    ah,ah              ; is it byte 1
        je      short byte_1
        test    eax,00ff0000h; is it byte 2
        je      short byte_2
        test    eax,0ff000000h; is it byte 3
        je      short byte_3
        jmp     short main_loop
                                   ; 31 is set
byte_3:
        lea     eax,[ecx - 1]
        mov     ecx,string
        sub     eax,ecx
        ret
byte_2:
        lea     eax,[ecx - 2]
        mov     ecx,string
        sub     eax,ecx
        ret
byte_1:
        lea     eax,[ecx - 3]
        mov     ecx,string
        sub     eax,ecx
        ret
byte_0:
        lea     eax,[ecx - 4]
        mov     ecx,string
        sub     eax,ecx
        ret
strlen  endp
```

MASM – Näiteid

- Optimeeritud strlen - SSE2

```
strlen: ; size_t strlen(const char* str);
   mov   edx, [esp + 4]; load arg str
   pxor  xmm0, xmm0    ; xmm0 := 0
   movd  xmm2, edx      ; store arg str in xmm2
   mov   ecx, edx       ; copy arg str
   and   edx, -16       ; 16-byte align ptr
   or    eax, -1
   pcmpeqb xmm0, [edx]    ; check whole qw for 0s
   and   ecx, 15      ; get #bytes of aligned dq before operand
   shl   eax, cl      ; create mask for the bytes of aligned dq in operand
   pmovmskb ecx, xmm0    ; collect mask of 0-bytes
   and   ecx, eax       ; mask out any 0s that occur before 1st byte
   jnz   strlen_end  ; found 0-byte, jump to end
   pxor  xmm0, xmm0    ; xmm0 := 0
   add   edx, 16        ; advance ptr
   align  4  ; auto-align the loop
@@:
   movdqa xmm1, [edx] ; get next chunk
   add   edx, 16        ; advance pointer
   pcmpeqb xmm1, xmm0  ; check for 0s
   pmovmskb ecx, xmm1    ; collect mask of 0-bytes
   test  ecx, ecx       ; any 0-bytes?
   jz    @B           ; no 0-bytes, so get next dq
   sub   edx, 16        ; back up ptr
strlen_end:
   bsf   eax, ecx      ; find first 1-bit (ie, first 0-byte)
   movd  ecx, xmm2; recover arg str from xmm2
   add   eax, edx       ; get address of the 0-byte
   sub   eax, ecx       ; subtract address of 1st byte to get string length
   ret
```