a problem in n5.dll in windows 95

Source: http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2004−12/0973.html

---

From: Ro (*inp_out_at_sim.tim*)
Date: 12/16/04

```
Date: Thu, 16 Dec 2004 17:15:35 GMT
```

I have in the pc this system

;NASM version 0.98.34 compiled on Jun 13 2002
;ALINK v1.6
; Microsoft Windows 95
; 4.00.950a
; IE 4.04.72.3110.8

and the files n5.asm and t3.asm below
the action in the dos shell are
>nasmw −f obj n5.asm
>alink −oPE −dll n5.obj
>nasmw −f obj t3.asm
>alink −oPE t3
>t3.exe
Now is all ok the program run
but if I close t3.exe (press the ok button)
now open t3.asm in a window (.asm call wordpad)
then in the dos window I type
>t3.exe
result
>"Il formato del programma non è valido. Impossibile eseguire
> il programma. Il file potrebbe essere danneggiato."
and program not run.

How I have to solve this?
I note that in window−xp seems all ok and in windows95 there are case
were the program (for exaple close t3.asm) run and case when not run

Thank you very much
−−−

```
    MB_SYSTEMMODAL equ 01000h

    extern GetTickCount
    import GetTickCount kernel32.dll
    extern MessageBoxA
    import MessageBoxA user32.dll

    global atou , atoi , utoa , itoa , utoh , uPrint , hPrint ,
AsciiToSt0 , fpow10i , St0ToAscii
    export atou
    export atoi
    export utoa
    export itoa
    export utoh
    export uPrint
    export hPrint
    export AsciiToSt0
    export fpow10i
    export St0ToAscii

    global srandMl , ScriviArrayDword , copy_d , fill_rand ,
fill_rand_dword_x21 , RND_lineare
    export srandMl
    export ScriviArrayDword
    export copy_d
    export fill_rand
    export fill_rand_dword_x21
    export RND_lineare

    global len_str , copy_str , message , see_buffer
    export len_str
    export copy_str
    export message
    export see_buffer

    global RND_RangeLineare , RND_Multilinear , RND_RTimer_a
    global ChiSquare , TestChiSquare , RND_RangeMLineare
    export RND_RangeLineare
    export RND_Multilinear
    export RND_RTimer_a
    export ChiSquare
    export TestChiSquare
    export RND_RangeMLineare

    global _main , ..start , dllstart

section code public use32 class=CODE

; DLL entry point − do nothing, but flag success
; This is a STDCALL entrypoint, so remove 3 params from stack on
return
```

```
..start:
_main:
dllstart:
    mov eax , 1
    ret 12

; eax srandMl(uint lpm, uint* srfine )
; eax=1 e CF=1 finito altrimenti eax=0 e CF=0 continuare
; k = 0k, 4ra, 8lpm, 12srfine
srandMl:
    push ebp
    mov ebp, esp
    %define @lpm [ebp+8]
    %define @srfine [ebp+12]
    cmp dword @srfine, 1
    je .r1
    push dword @lpm
    push index
    push 16
    push base
    call fill_rand_dword_x21
    jnc .r0
    cmp dword[index], 32
    jne .c0
    shl dword[index], 1 ; uPrint(Inuu, D*index);
.r0:
mov eax, 0
clc
jmp .cf
.r1:
mov eax, 1
stc
jmp .cf
.c0:
push dword @lpm
push index
push 128
push base
call fill_rand
jnc .r0
    mov eax, @srfine
    cmp dword[eax], 1
    je .r1
    mov dword[eax], 1
    pushad
    push MB_OK | MB_SYSTEMMODAL
    push tit
    push Inuu
    push 0
    call [MessageBoxA] ; cambia c, r
    push 16
```

```
    push seed
    push seed0
    call copy_d
    push 32
    push base
    push 512
    push buffer
    call ScriviArrayDword
    push MB_OK | MB_SYSTEMMODAL
    push tit
    push buffer
    push 0
    call [MessageBoxA]
    popad
    jmp .r1

.cf:
    %undef @lpm
    %undef @srfine
    mov esp, ebp
    pop ebp
    ret 8

; eax:len
; ScriviArrayDword(char*buf,uint len_buf,uint*adw,uint len_adw_in_dw)
; 0i,4r,8c,12b,16ra, 20P_buf, 24P_len_buf, 28P_adw, 32P_len_adw_in_dw
ScriviArrayDword:
    push ebx
    push ecx
    push edx
    push esi
    %define @buf [esp+20]
    %define @len_buf [esp+24]
    %define @adw [esp+28]
    %define @len_adw_in_dw [esp+32]
    mov ebx, @buf
    mov esi, @adw
    mov ecx, ebx
    mov eax, @len_buf
    cmp ebx, 0
    je .ce
    cmp esi, 0
    je .ee
    cmp eax, 0
    je .ce
    cmp dword @len_adw_in_dw, 0
    je .ee
    mov edx, 0
    sub eax, 13
    jle .ee
    add ecx, eax ; in c vi e' il limite da non superare
```

```
    jmp short .c0
.ea:
dec ebx
mov byte[ebx], 0
mov eax, ebx
sub eax, @buf
clc
jmp .cf
.ee: ; .ea, .ee, .ce varie uscite per errore
mov byte[ebx], 0
.ce:
mov eax, 0
stc
jmp .cf
.c0: ; Scive la stringa dei seed in buffer
push dword[esi+4*edx]
push ebx utoa
    add ebx, eax ; notare se .ea non
    mov byte[ebx], '#'
    inc edx
    inc ebx
    cmp ebx, ecx
    jae .ea
    cmp edx, @len_adw_in_dw
    jb .c0 ; stampa # alla fine
    mov byte[ebx], 0
    mov eax, ebx
    sub eax, @buf
    inc eax
    clc
.cf:
    %undef @buf
    %undef @len_buf
    %undef @adw
    %undef @len_adw_in_dw
    pop esi
    pop edx
    pop ecx
    pop ebx
    ret 16

; void copy_d(uint* array, uint* origin, uint num_di_dword)
; 0j, 4i, 8c, 12a, 16ra, 20P_array, 24P_origin, 28P_num
copy_d:
    push eax
    push ecx
    push esi
    push edi
    %define @array [esp+20]
    %define @origin [esp+24]
    %define @num [esp+28]
```

```
   mov edi, @array
   mov esi, @origin
   mov ecx, 0
.c0:
mov eax, [esi+4*ecx]
mov [edi+4*ecx], eax
inc ecx
cmp ecx, @num
jb .c0
   %undef @array
   %undef @origin
   %undef @num
   pop edi
   pop esi
   pop ecx
   pop eax
   ret 12

; eax:stato fill_rand( char* arrayc, uint sizec, uint* lenc, uint
lpm )
; CF=1 e *len=size fine e tutto ok, altrimenti CF=0 deve continuare
; CF=1 e *len!=size: errore size o *len fuori range
; *len<size < 1000000 chars
; array da riempire con valori random, in *len dove siamo arrivati in
; word, lpm non e' altro che @lparm di MainWindowProc ma puo' essere
; un qualunque altro valore del sistema che varia indipendentemente
; dall'orologio
; tipico uso var dd 0
; array times 512 db 0
; fill_rand(array, 512, var, D [valore] )
; k= 0k, 4ra, 8@array, 12@size, 16@len, 20@lpm
fill_rand:
   push ebp
   mov ebp, esp
   push ebx
   push esi
   push edi
   %define @array [ebp+8]
   %define @size [ebp+12]
   %define @len [ebp+16]
   %define @lpm [ebp+20]
   mov esi, @len ; controllo errori
   mov edi, @array
   mov eax, @size
   cmp esi, 0
   je .ce
   cmp edi, 0
   je .ce
   mov ebx, [esi]
   cmp eax, 0
   je .ce
```

```
   cmp eax, 1000000
   ja .ce
   cmp ebx, eax
   jae .ce
   jmp short .c0
.ce:
stc
mov eax, 1
jmp .cf
.c0:
push dword @lpm
call RND_RTimer_a
   inc ebx
   cmp ebx, @size
   jne .c1
   dec ebx
   xor al, ah
   mov [edi+ebx], al
   inc ebx
   mov [esi], ebx
   jmp short .ce
.c1:
dec ebx
mov [edi+ebx], ax
add ebx, 2
mov [esi], ebx
   cmp ebx, @size
   jae .ce
   clc
   mov eax, 0
.cf:
   %undef @array
   %undef @size
   %undef @len
   %undef @lpm
   pop edi
   pop esi
   pop ebx
   mov esp, ebp
   pop ebp
   ret 16

; carry_flag
; fill_rand_dword_x21( uint* array_d, uint size_d, uint* len_w,
uint lpm )
; CF=1 e *len=size fine e tutto ok, altrimenti CF=0 deve continuare
; CF=1 e *len!=size: errore size o *len fuori range
; *len<size < 1000000 chars
; serve per riempore i seeds con dword di tipo ...x21 con x pari
; Array da riempire con valori random, in *len dove siamo arrivati in
; word, lpm non e' altro che @lparm di MainWindowProc ma puo' essere
```

```
; un qualunque altro valore del sistema che varia indipendentemente
; dall'orologio
; tipico uso var dd 0
; array times 512 dd 0
; fill_rand_dword_x21(array, 32, var, D [valore] )
; k= 0k, 4ra, 8@array, 12@size, 16@len, 20@lpm
fill_rand_dword_x21:
   push ebp
   mov ebp, esp
   push ebx
   push edx
   push esi
   push edi
   %define @array [ebp+8]
   %define @size [ebp+12]
   %define @len [ebp+16]
   %define @lpm [ebp+20]
   mov esi, @len ; controllo errori
   mov edi, @array
   mov eax, @size
   cmp esi, 0
   je .ce ; b e' misurato in *word*
   cmp edi, 0
   je .ce
   mov ebx, [esi]
   cmp eax, 0
   je .ce
   cmp eax, 100000
   ja .ce
   shl eax, 1
   cmp ebx, eax
   jae .ce
   jmp short .c0
.ce:
stc
mov eax, 1
jmp .cf
.c0:
push dword @lpm
call RND_RTimer_a
   test ebx, 1
   jnz .b0 ; word dispari in .b0
   test al, 1
   jz .w0 ; ax deve essere pari
   inc ax
.w0:
mov [edi+2*ebx], ax
jmp short .c1
.b0:
   xor al, ah
   mov ah, 0
```

```
    mov [edi+2*ebx], ax
    mov eax, [edi + 2*ebx − 2]
    mul dword[cento]
    add eax, 21
    mov [edi + 2*ebx − 2], eax
.c1:
inc ebx
mov [esi], ebx
    mov eax, @size
    shl eax, 1
    cmp ebx, eax
    jae .ce
    clc
    mov eax, 0
.cf:
    %undef @array
    %undef @size
    %undef @len
    %undef @lpm
    pop edi
    pop esi
    pop edx
    pop ebx
    mov esp, ebp
    pop ebp
    ret 16

; uint RND_lineare()
; dd, .base=31415821, .seed=123;
RND_lineare:
    push edx
    mov eax, [basea]
    mul dword[seeda]
    inc eax
    mov [seeda], eax
    pop edx
    ret

; 0 r, 4 c, 8 Ra, 12 @Min, 16 @Max
RND_RangeLineare:
    push ecx
    push edx
    %define @Min [esp+12]
    %define @Max [esp+16]
    call RND_lineare
    mov ecx, @Max
    sub ecx, @Min
    cmp ecx, 0
    jne .co
    inc ecx
.co:
```

```
mov edx, 0
div ecx
   mov eax, edx
   add eax, @Min
   %undef @Min
   %undef @Max
   pop edx
   pop ecx
   ret 8

; uint RND_MultiL(base, seed, seed0)
; 0 b, 4 r, 8 c, 12 Ra, 16@Base, 20@Seed, 24@Seed0
RND_MultiL:
   push ecx
   push edx
   push ebx
   %define @Base [esp+16]
   %define @Seed [esp+20]
   %define @Seed0 [esp+24]
   mov edx, @Base
   mov ecx, @Seed
   mov ebx, [edx]
   mov eax, [ecx]
   mul ebx
   inc eax
   mov edx, @Seed0
   mov edx, [edx]
   mov [ecx], eax
   cmp eax, edx
   jne .con
   call RND_lineare
   xor eax, ebx
   test eax, 1
   jne .l0
   inc eax
.l0: ; cambia base ma dispari
mov edx, @Base
mov [edx], eax
   call RND_lineare ; cambia seed0
   mov edx, @Seed0
   xor [edx], eax
   mov eax, @Seed ; cambia seed=seed0 e lo ritorna
   mov edx, [edx]
   mov [eax], edx
   mov eax, edx

.con:
   %undef @Base
   %undef @Seed
   %undef @Seed0
   pop ebx
```

```
   pop edx
   pop ecx
   ret 12

; eax:uint RND_Multiineare(void);
; risultato in eax numero pseudocasuale tra 0 e 0x7FFFFFFF
RND_Multilinear:
   push ebx
   push ecx
   push edx
   mov eax, [index1]
   lea ebx, [base+ 4*eax]
   lea ecx, [seed+ 4*eax]
   lea edx, [seed0+ 4*eax]
   push edx
   push ecx
   push ebx
   call RND_MultiL
   shr eax, 1
   mov bl, ah
   and ebx, 0Fh
   mov [index1], ebx
   bswap eax ; 0..7<->24..31, 8..15<->16..23
   test eax, 0x80000000
   jz .cil
   or ax, 0x80
   and eax, 0x7FFFFFFF

.cil:
   pop edx
   pop ecx
   pop ebx
   ret

; eax:uint RND_RangeMLineare(uint min, uint max)
; risultato in eax numero pseudocasuale tra min e max
; 0 r, 4 c, 8 Ra, 12 @Min, 16 @Max
RND_RangeMLineare:
   push ecx
   push edx
   %define @Min [esp+12]
   %define @Max [esp+16]
   call RND_Multilinear
   mov ecx, @Max
   sub ecx, @Min
   cmp ecx, 0
   jne .co
   inc ecx
.co:
mov edx, 0
div ecx
```

```
    mov eax, edx
    add eax, @Min
    %undef @Min
    %undef @Max
    pop edx
    pop ecx
    ret 8


;_____________________________________________________
; k= 0k, 4ra, 8 @MousePos
; 0FFFF_FFFFh F=1111
RND_RTimer_a:
    push ebp
    mov ebp, esp
    push ebx
    push ecx
    push edx
    push esi
    push edi
    %define @MousePos [ebp+8]
;−−−−−−−−−−−−−−−−−−−−−−
    mov ebx, @MousePos
    mov eax, @MousePos
    shr eax, 4
    and ebx, 0Fh
    and eax, 0F0h
    or ebx, eax
    push ebx
    call [GetTickCount] ; cambia r?
    pop ebx
    shl eax, 8
    or eax, ebx
    and eax, 0FFFFh
    %undef @MousePos
    pop edi
    pop esi
    pop edx
    pop ecx
    pop ebx
    mov esp, ebp
    pop ebp
    ret 4



; st0 ChiSquare(D@func, D@Points, D@Max)
; ove max < (10000 − 20)/4 = 2495
;
; Serve per fare il test "ChiSquare" alla funzione
; casuale di range funct(min, max) chiamata qui con
```

```
; funct(0, D @max) ove @Point> 10*@Max per la validita
; del test. Il libro dice che la sequenza e' ok
; se st0 E [@max−2*sqrt(@max), @max+2*sqrt(@max)]
; (tranne una volta su dieci)
; 0 k, 4Ra, 8 D@func, 12D@points, 16D@Max (k)
ChiSquare:
   push ebp
   mov ebp, esp
   push eax
   push ecx
   push ebx
   push edx
   push esi
   %define @func [ebp+8]
   %define @Points [ebp+12]
   %define @Max [ebp+16]
   mov eax, @func ; controllo dei dati di ingresso
   cmp eax, 0
   jne .l6
   jmp short .l9
.l6:
mov eax, @Points
cmp eax, 0
jne .l7
jmp short .l9
.l7:
mov eax, @Max
cmp eax, 0
jne .l8
.l9: ; st0=0 esci
fldz
jmp .fine
.l8:
shl eax, 2
add eax, 20
cmp eax, 10000
jbe .l0
jmp short .l9
.l0:
sub esp, eax
mov ecx, 0
.l1:
mov dword[esp+ecx], 0
add ecx, 4
cmp ecx, eax
jb .l1
   mov ecx, 0
   mov edx, @Max
.l2:
push edx
push 0
```

```
call dword @func
inc dword[esp+4*eax]
inc ecx
   cmp ecx, @Points
   jb .l2
   mov ecx, 0
   mov esi, 0
.l3:
mov eax, dword[esp+4*ecx]
mul dword[esp+4*ecx]
inc ecx
cmp edx, 0
jne .l4
add esi, eax
jc .l4
   cmp ecx, @Max
   jb .l3
   mov [esp], esi
; finit; il caller lo deve fare una volta
   fild dword @Points
   fild dword @Points
   fild dword [esp]
   fild dword @Max
; stack st = @Max , i, @points, @points
   fmulp st1 ; st = @max*i, @points, @points
   fdivrp st1 ; st = @max*i/@points, @points
   fsubrp st1 ; st = @max*i/@points − @points
   jmp short .l5
.l4: ; errore numero troppo grande per la somma
fldz
.l5:
mov eax, @Max
shl eax, 2
add eax, 20
   add esp, eax
.fine:
   %undef @func
   %undef @Points
   %undef @Max
   pop esi
   pop edx
   pop ebx
   pop ecx
   pop eax
   mov esp, ebp
   pop ebp
   ret 12



; double TestChiSquare(D@func, D@Points, D@Max)
```

```
; 0 k, 4Ra, 8 D@func, 12D@points, 16D@Max (k)
TestChiSquare:
   push ebp
   mov ebp, esp
   %define @func [ebp+8]
   %define @Points [ebp+12]
   %define @Max [ebp+16]
   push dword @Max
   push dword @Points
   push dword @func
   call ChiSquare
   push 4
   push 0
   push buffer
   call St0ToAscii
   fstp st0
   pushad
   push MB_OK | MB_SYSTEMMODAL
   push chi
   push buffer
   push 0
   call [MessageBoxA]
   popad
   %undef @func
   %undef @Points
   %undef @Max
   mov esp, ebp
   pop ebp
   ret 12


; eax atou(char* string, char* pos)
; 0 i, 4 r, 8 b, 12 Ra, 16 string, 20 pos
atou:
   push ebx
   push edx
   push esi
   %define @string [esp+16]
   %define @pos [esp+20]
   mov esi, @string
   jmp short .c1
.c0:
inc esi
.c1:
cmp byte[esi], ' '
je .c0
   cmp byte[esi], '+'
   jne .c2
   inc esi
.c2:
cmp byte[esi], '9'
```

```
ja .ce
   cmp byte[esi], '0'
   jb .ce
   jmp short .c4
.ce:
mov eax, @pos
mov edx, @string
mov [eax], edx
mov eax, 0
jmp short .cf
.c3:
inc esi
.c4:
cmp byte[esi], '0'
je .c3
   mov eax, 0
   mov ebx, 0
.c5:
mov bl, [esi]
   cmp bl, '9'
   ja .c7
   cmp bl, '0'
   jb .c7
   mul dword[dieci]
   cmp edx, 0
   jne .ci
   sub ebx, '0'
   add eax, ebx
   jc .ci
   inc esi
   jmp short .c5
.ci:
mov eax, 0xFFFFFFFF
.c6:
inc esi
mov bl, [esi]
cmp bl, '9'
ja .c7
   cmp bl, '0'
   jb .c7
   jmp short .c6
.c7:
mov edx, @pos
mov [edx], esi
.cf:
   %undef @string
   %undef @pos
   pop esi
   pop edx
   pop ebx
   ret 8
```

```
; eax atoi(char* string, char* pos)
; 0 i, 4 r, 8 c, 12 b, 16 Ra, 20 string, 24 pos
atoi:
   push ebx
   push ecx
   push edx
   push esi
   %define @string [esp+20]
   %define @pos [esp+24]
   mov esi, @string ; in c il segno
   mov ecx, 0
   jmp short .c1
.c0:
inc esi
.c1:
cmp byte[esi], ' '
je .c0
   cmp byte[esi], '+'
   jne .c2
   inc esi
   jmp short .c3
.c2:
cmp byte[esi], '−'
jne .c3
inc esi
inc ecx
.c3:
cmp byte[esi], '9'
ja .ce
   cmp byte[esi], '0'
   jb .ce
   jmp short .c5
.ce:
mov eax, @pos
mov edx, @string
mov [eax], edx
mov eax, 0
jmp short .cf
.c4:
inc esi
.c5:
cmp byte[esi], '0'
je .c4
   mov eax, 0
   mov ebx, 0
.c6:
mov bl, [esi]
   cmp bl, '9'
   ja .c8
   cmp bl, '0'
```

```
   jb .c8
   mul dword[dieci] ; in ci +/−oo
   cmp edx, 0
   jne .ci
   sub ebx, '0'
   add eax, ebx
   jc .ci
   test eax, 0x80000000
   jnz .ci
   inc esi
   jmp short .c6
.ci:
mov eax, 0x7FFFFFFF
.c7:
inc esi
mov bl, [esi]
cmp bl, '9'
ja .c8
   cmp bl, '0'
   jb .c8
   jmp short .c7
.c8:
cmp ecx, 0
jne .c9
neg eax
.c9:
mov edx, @pos
mov [edx], esi
.cf:
   %undef @string
   %undef @pos
   pop esi
   pop edx
   pop ecx
   pop ebx
   ret 8

; uso: str .titolo, "Numero " ; uPrint(titolo, D*num);
; uPrint( char* Title, unsigned n)
; k k= 0k, 4 Ra, 8 1P_Title, 12 1P_n
uPrint:
   push ebp
   mov ebp, esp
   pushad ; sono necessari perche messagebox sembra cambi anche
ecx!!!
   sub esp, 128
   %define @Title [ebp+8]
   %define @n [ebp+12]
   %define @string [esp]
;−−−−−−−−−−−−−−−−−−−−*/
   mov eax, @n
```

```
    lea ecx, @string
    push eax
    push ecx
    call utoa
    mov eax, @Title
    lea ecx, @string
    push MB_OK | MB_SYSTEMMODAL
    push ecx
    push eax
    push 0
    call [MessageBoxA]
    %undef @Title
    %undef @n
    %undef @string
    add esp, 128
    popad
    mov esp, ebp
    pop ebp
    ret 8


; eax utoa(char* string, unsigned n)
; ritorna in eax i chiars scritti
; 0 k, 4 j, 8 i, 12 r, 16 b, 20 Ra, 24 1P_string, 28 1P_n
utoa:
    push ebx
    push edx
    push esi
    push edi
    push ebp
    %define @string [esp+24]
    %define @n [esp+28]
;−−−−−−−−−−−−−−−−−−−−*/
    mov esi, @string
    mov ebx, 10
    mov eax, @n
    mov edi, esi
    mov ebp, 0
.utoaz_l:
    xor edx, edx
    div ebx ; a= (r:a)/b r= (r:a)%b
    add dl, '0'
    mov [edi], dl
    inc edi
    inc ebp
    cmp eax, 0
    jne .utoaz_l

    mov byte[edi], 0
    dec edi
    jmp short .vai
```

```
.utoaz_f:
   mov al, [esi]
   mov ah, [edi]
   mov [edi], al
   mov [esi], ah
   inc esi
   dec edi
.vai:
   cmp esi, edi
   jb .utoaz_f

   mov eax, ebp
.utoaz_z:
   %undef @string
   %undef @n
   pop ebp
   pop edi
   pop esi
   pop edx
   pop ebx
   ret 8


; eax itoa(char* string, int n)
; ritorna in eax i chiars scritti
; 0 k, 4 j, 8 i, 12 r, 16 b, 20 Ra, 24 1P_string, 28 1P_n
itoa:
   push ebx
   push edx
   push esi
   push edi
   push ebp
   %define @string [esp+24]
   %define @n [esp+28]
;−−−−−−−−−−−−−−−−−−−−*/
   mov esi, @string
   mov ebx, 10
   mov eax, @n
   mov ebp, 0
   test eax, 0x80000000
   jz .c0
   neg eax ; a&0x70000000;
   mov byte[esi], '−'
   jmp short .c1
.c0:
mov byte[esi], '+'
.c1:
inc esi
inc ebp
mov edi, esi
.itoaz_l:
```

```
    xor edx, edx
    div ebx ; a= (r:a)/b r= (r:a)%b
    add dl, '0'
    mov [edi], dl
    inc edi
    inc ebp
    cmp eax, 0
    jne .itoaz_l

    mov byte[edi], 0
    dec edi
    jmp short .vai
.itoaz_f:
    mov al, [esi]
    mov ah, [edi]
    mov [edi], al
    mov [esi], ah
    inc esi
    dec edi
.vai:
    cmp esi, edi
    jb .itoaz_f

    mov eax, ebp
.itoaz_z:
    %undef @string
    %undef @n
    pop ebp
    pop edi
    pop esi
    pop edx
    pop ebx
    ret 8



; void utoh(char* Title, uint n)
; Betov
; scrive 8 chars in Title => 8+1'\0'=9 chars richiesti
; 0−7 digits 8'\0' 9 caratteri in tutto
; 0 a, 4 c, 8 b, 12 i, 16 Ra, 20 1P_Title, 24 1P_n
utoh:
    push eax
    push ecx
    push ebx
    push esi
    %define @n [esp+24]
    %define @Title [esp+20]
    mov ebx, @n
    mov ecx, @Title
    mov esi, ecx
```

```
   add esi, 7
   mov byte[esi+1], 0
.l0:
   mov al, bl
   and al, 0Fh
   add al, '0'
   cmp al, '9'
   jbe .l1
   add al, 7
.l1:
   mov byte[esi], al
   dec esi
   shr ebx, 4
   cmp ecx, esi
   jbe .l0

   %undef @n
   %undef @Title
   pop esi
   pop ebx
   pop ecx
   pop eax
   ret 8




Tprinth:
   pushad
   sub esp, 64
   mov dword[esp+8], 0
   fstp tword[esp] ; il valore in st0 e levato dallo
fpu−stack
   lea ebx, [esp+12]
   mov esi, esp
   push dword[esi]
   push ebx
   call utoh ; 32
   add ebx, 8
   mov byte[ebx], '#'
   inc ebx
   push dword[esi+4]
   push ebx
   call utoh ; +32
   add ebx, 8
   mov byte[ebx], '#'
   inc ebx
   push dword[esi+8]
   push ebx
   call utoh ; +16
   add ebx, 8
```

```
    mov byte[ebx], '#'
    inc ebx
    mov byte[ebx], '#'
    inc ebx
    mov byte[ebx], 0
    lea ebx, [esp+12]
    push MB_OK | MB_SYSTEMMODAL
    push titolo
    push ebx
    push 0
    call [MessageBoxA]
    add esp, 64
    popad
    ret


; uso: str titolo, "Numero " ; hPrint(titolo, D*num);
; hPrint(char* Title, unsigned n)
; k= 0k, 4 Ra, 8 1P_Title, 12 1P_n
hPrint:
    push ebp
    mov ebp, esp
    pushad ; sono necessari perche messagebox sembra cambi anche
ecx!!!
    sub esp, 128
    %define @Title [ebp+8]
    %define @n [ebp+12]
    %define @HexprintString [esp]
;--------------------*/
    mov eax, @n
    lea ecx, @HexprintString
    push eax
    push ecx
    call utoh
    mov eax, @Title
    lea ecx, @HexprintString
    push MB_OK | MB_SYSTEMMODAL
    push eax
    push ecx
    push 0
    call [MessageBoxA]
    %undef @Title
    %undef @n
    %undef @HexprintString
    add esp, 128
    popad
    mov esp, ebp
    pop ebp
    ret 8
```

```
; st0 AsciiToSt0(P_string, P_address_then);
; se [P_address_then]==[P_string] allora st0=0 e non e' preso
altrimenti
; il numero e' preso. Utilizza due registri dello stack−fpu
; per k: 0 k, 4Ra, 8P_string, 12P_address
AsciiToSt0:
   push ebp
   mov ebp, esp
   push ebx
   push ecx
   push edx
   push esi
   push edi
   sub esp, 16 ; s=0num0, 4num1, 8sign, 12cw_save
   fnstcw [esp+12]
   fldcw [cw_data] ; resetta tutto + troncamento
; converti scrive nello 2 dd nello stack qui allocato
   mov esi, [ebp+8]
   call converti
   jnc .a0
   mov eax, [ebp+8]
   mov ebx, [ebp+12]
   mov [ebx], eax
   fldz
   jmp .cx
.a0: ; caso non errore
mov [esp+8], ebx
mov eax, 0
   jmp short .c1 ; leva gli spazi
.c0:
inc edi
.c1:
cmp byte[edi], ' '
je .c0
   mov bl, byte[edi]
   cmp bl, 'e'
   je .c2 ; 44.44 e −12455
   cmp bl, 'E'
   jne .c3
.c2: ; in a esponente
call converti_exp
   cmp eax, 4931
   jg .ce_piu_oo
   cmp ecx, 4931
   jg .ce_piu_oo
   cmp eax, −4931
   jl .ce_zero
.c3:
add eax, ecx
; uPrint(titolo, a);
   cmp eax, 4931
```

```
    jg .ce_piu_oo
    cmp eax, −4931
    jl .ce_zero
    jmp short .c4
.ce_piu_oo:
cmp dword[esp+8], 0
jne .seielmeno
fld tword[piu_oo]
jmp .cq
.seielmeno:
fld tword[meno_oo]
jmp .cq
.ce_zero:
fldz
jmp .cf
.c4: ; st = 10^a
call fpow10i
.c5: ; st = num, 10^a
fild qword[esp]
    fmulp st1 ; st = num*10^a
.cf:
cmp dword[esp+8], 0
je .cq
FCHS
.cq:
mov eax, [ebp+12]
mov [eax], edi
.cx:
fldcw [esp+12]
    add esp, 16
    pop edi
    pop esi
    pop edx
    pop ecx
    pop ebx
    mov esp, ebp
    pop ebp
    ret 8

; <j>pos <a>num+sign converti_exp<j>
converti_exp:
    push ebx
    push ecx
    push edx
    push esi
    push ebp
    mov esi, edi
    mov ebx, 0
    mov edx, 0
.c0: ; leva gli spazi
inc edi
```

```
cmp byte[edi], ' '
je .c0
   cmp byte[edi], '+'
   jne .c1
   inc edi
   jmp short .c2
.c1:
cmp byte[edi], '−'
jne .c2
inc edi
inc ebx
.c2:
cmp byte[edi], '9'
ja .ce
   cmp byte[edi], '0'
   jb .ce ; nel caso dopo eventuale + o −
   jmp short .c3
.ce:
mov edi, esi
mov eax, 0
jmp .cf
.c3: ; leva gli zeri
jmp short .c5
.c4:
inc edi
.c5:
cmp byte[edi], '0'
je .c4
   mov eax, 0
   mov ecx, 0
   mov ebp, 0
.c6:
mov cl, [edi ]
   cmp cl, '9'
   ja .c8
   cmp cl, '0'
   jb .c8
   mul dword[dieci]
   sub cl, '0'
   add eax, ecx
   cmp eax, 4931
   ja .ck
   inc edi
   jmp short .c6
.ck:
inc edi
mov cl, [edi ]
   cmp cl, '9'
   ja .c8
   cmp cl, '0'
   jb .c8
```

```
    jmp short .ck
.c8:
cmp ebx, 1
jne .cf
neg eax
.cf:
    pop ebp
    pop esi
    pop edx
    pop ecx
    pop ebx
    ret

; input signed int a; output 10^a in st0
; utilizza due posizioni dello fpu−stack
; non inizializza niente; rounding mode per troncamento
; settato all'esterno
fpow10i:
    push eax
    push ebx
    push ecx
    cmp eax, −4931
    jl .ze
    cmp eax, 4931
    jg .oo
    jmp short .c0
.ze:
fldz
jmp .fine
.oo:
fld tword[piu_oo]
jmp .fine
.c0:
mov ebx, 0
test eax, 0x80000000
jz .c1
neg eax
mov ebx, 1
.c1: ; c= c*10 serve poiche' un
mov ecx, eax
and ecx, 0xF
lea ecx, [ecx+4*ecx]
lea ecx, [2*ecx]
    FLD tword[dieci_1 + ecx] ; elemento occupa 10 chars
    mov ecx, eax
    shr ecx, 4
    and ecx, 0xF
    lea ecx, [ecx+4*ecx]
    lea ecx, [2*ecx]
    FLD tword[dieci_16 + ecx]
    mov ecx, eax ; ok poiche' c<=4931
```

```
    shr ecx, 8
    FMULP st1
    lea ecx, [ecx+4*ecx] ; e 4931>>8 == 19
    lea ecx, [2*ecx]
    FLD tword[dieci_256 + ecx]
    cmp ebx, 1
    FMULP st1
    jnz .fine
    FLD1
    FDIVRP st1
.fine:
    pop ecx
    pop ebx
    pop eax
    ret

; segno in b
; numero nello stack del chiamante in num1:num0 numero
; di cifre in c, ultima posizione in j
; nel caso di errore o nessun numero stc altrimenti clc
; converti<origine i>
; 0 @sign, 4k, 8r, 12 i, 16 a, 20 Ra, 24 num0, 28 num1
converti:
    push eax
    push esi
    push edx
    push ebp ; i punta sempre al carattere
; corrente della stringa
    %define @sign [esp]
    sub esp, 4
    jmp short .c1 ; leva gli spazi
.c0:
inc esi
.c1:
cmp byte[esi], ' '
je .c0
    mov dword @sign, 0
    cmp byte[esi], '+'
    jne .b0
    inc esi
    jmp short .c2
.b0:
cmp byte[esi], '−'
jne .c2
inc esi
inc dword @sign
.c2:
cmp byte[esi], '9'
ja .ce
    cmp byte[esi], '0'
    jb .ce ; nel caso dopo eventuale + o −
```

```
   jmp short .b1 ; c'e' un non numero => errore
.ce:
mov edx, 0
mov ebp, 0
mov eax, 0
mov ecx, 0
mov ebx, 0
mov edi, [ esp + 12 ]
jmp .cf
.b1: ; leva gli zeri
jmp short .g1
.g0:
inc esi
.g1:
cmp byte[esi], '0'
je .g0
   mov eax, 0
   mov edx, 0
   mov ebp, 0
   mov edi, 0
   mov ecx, 0
.c3: ; prende parte non esponente
mov al, [esi ]
   cmp al, '9'
   ja .c4
   cmp al, '0'
   jb .c4
   call _mult10
   jc .r1
   sub al, '0'
   add ebp, eax
   jnc .b2
   inc edx
   jno .b2
   sub ebp, eax
   dec edx
   dec ecx
.r1:
jmp .c7
.b2:
inc esi
jmp short .c3
.c4:
cmp al, '.'
je .a1
jmp .c9
.a1:
cmp byte[esi+1], '9'
ja .a2
cmp byte[esi+1], '0'
jb .a2
```

```
   jmp short .a3
.a2: ; caso numero.?non_numero
jmp .c9
.a3: ; caso numero.?numero
mov edi, esi
.ca: ; elimina gli zeri superflui
inc edi
mov al, [edi ]
   cmp al, '9'
   ja .cb
   cmp al, '0'
   jb .cb
   jmp short .ca
.cb: ; j punta *all'ultimo numero* non zero
dec edi
mov al, [edi ]
cmp al, '0'
je .cb
   cmp esi, edi
   jne .cd ; tranne caso numero.000000000
   jmp .c9
.cd:
cmp esi, edi
je .a0
   inc esi
   mov al, [esi]
   call _mult10 ; elimina i restanti decimali
   jc .a0
   sub al, '0'
   dec ecx
   add ebp, eax
   jnc .a4
   inc edx
   jno .a4
   sub ebp, eax
   dec edx
   jmp short .a0
.a4:
jmp short .cd

.c7: ; caso 999etc
inc ecx
inc esi
mov al, [esi ]
   cmp al, '9'
   ja .c8
   cmp al, '0'
   jae .c7
.c8: ; caso 999etc.etc
cmp al, '.'
jne .c9
```

```
    cmp byte[esi+1], '9'
    ja .c9
    cmp byte[esi+1], '0'
    jb .c9
.a0:
inc esi
mov al, [esi ]
    cmp al, '9'
    ja .c9
    cmp al, '0'
    jae .a0
.c9:
mov edi, esi
mov ebx, @sign
.cf:
add esp, 4
    cmp edi, [esp+8]
    jne .cc
    stc
    jmp short .cz
.cc: ; [s+8]==i
clc
.cz: ; nello stack la risposta
mov [esp+20], ebp
mov [esp+24], edx
    %undef @sign
    pop ebp
    pop edx
    pop esi
    pop eax
    ret


; r:k mult10<r:k>
; ma (r:k)*10< 1 0 0 ? clearCarry: setCarry
; 0 j, 4 i, 8 c, 12 b, 16 a, 20 Ra
_mult10:
    push eax
    push ebx
    push ecx
    push esi
    push edi
;-------------------*/
    mov ecx, edx ; salva input in c:j
    mov edi, ebp
    mov eax, ebp
    mul dword[dieci] ; k*10
    mov esi, eax ; save result in i:k
    mov ebp, edx

    mov eax, ecx
```

```
    mul dword[dieci]
    add eax, ebp
    jnc .m0
    inc edx
.m0:
    cmp edx, 0
    jne .mfineset
    mov edx, eax
    mov ebp, esi
    test edx, 0x80000000
    jnz .mfineset
; qui perde un bit per il segno
    clc
    jmp short .mfine
.mfineset:
    mov edx, ecx
    mov ebp, edi
    stc
.mfine:
    pop edi
    pop esi
    pop ecx
    pop ebx
    pop eax
    ret


; eax St0ToAscii( char* string, uint intPart, uint decPart )
<st0>;
;
; Suppone che la stringa sia sufficientemente ampia, ritorna in eax il
; numero di caratteri scritti, usa due registri float, se intPart==0
; scrive il numero senza esponente se l'esponente permette (0<e<=17)
; altrimenti (intPart, decPart)=( 1, decPart ), se intPart o decPart
; sono fuori range li aggiusta privilegiando intPart
; k: 0k, 4 ra, 8 P_string, 12 P_intPart, 16 P_decPart
St0ToAscii:
    push ebp
    mov ebp, esp
    push ebx
    push ecx
    push edx
    push esi
    sub esp, 64 ; 10, 20; 0−9=num, 12−15=exp, 16−17=wc_temp
20−29=bcd
    %define @string [ebp+8]
    %define @exp [esp+12]
    %define @wc_temp [esp+16]
    %define @bcd [esp+20]
    fnstcw @wc_temp
    mov dword[esp+8], 0
```

```
   fldcw [cw_data] ; resetta tutto + troncamento
   fxam ; controlla numero in st0 setta c0, c1, c2,
c3
   mov esi, @string
   fnstsw ax ; store in ax content of stauts reg
; 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
; c0 c1 c2 c3
; c1= sign; c3, c2, c0
; non suppo. 0 0 0 formato
; NaN 0 0 1
; Finite num. 0 1 0
; Infinity 0 1 1
; Zero 1 0 0
; Empty reg. 1 0 1
; Denormal 1 1 0
   mov al, ah
   test al, 00000010b
   jz .c0
   mov byte[esi], '−'
   jmp short .c1
.c0:
mov byte[esi], '+'
.c1: ; leva il bit del segno
inc esi
and al, 01000101b
   cmp al, 00000000b
   jne .c2 ; in .cf nessuna operazione in fpu−stack
   dec esi
   mov dword[esi], 'Ns'
   mov eax, 2
   jmp .cf
.c2:
cmp al, 00000001b
jne .c3
   dec esi
   mov dword[esi], 'NaN'
   mov eax, 3
   jmp .cf
.c3:
cmp al, 00000101b
jne .c4
   mov dword[esi], 'InF'
   mov eax, 4
   jmp .cf
.c4:
cmp al, 01000000b
jne .c5
   mov dword[esi], '0.0'
   mov eax, 4
   jmp .cf
.c5:
```

```
cmp al, 01000001b
jne .c7
   dec esi
   mov dword[esi], 'Nul'
   mov eax, 3
   jmp .cf
.c6:
mov dword[esi], '0.0'
mov eax, 4
jmp .ca
.c7: ; st = num, num
fld st0
   fstp tword[esp] ; st = num ; salva il numero num
   fxtract ; st = f(num), e(num)
   fstp st0 ; st = e(num)
   fldlg2 ; st = log_10(2),e(num)
   fmulp st1 ; st = log_10(2)*e(num)
   fistp dword @exp ; st = vuoto; salva x in 10^x, pop;
; salva l'esponente
   mov eax, @exp ; in .c6 0.0
   neg eax
   add eax, 17
   cmp eax, 4931
   jg .c6
; 1/10^x * 10^17 = 10^(17−x)=10^a
   call fpow10i ; st = 10^a
   fld tword[esp] ; st = num,10^a
   fmulp st1 ; st = num*10^a
   fbstp tword @bcd ; st = vuoto; numero in memoria
; ^^^^^^^^^^^^^^^^ salva numero in @bcd=[s+20] *come bcd*
; −−−−−−−−−−−−−−−− scrive nella stringa
   lea eax, @bcd
   mov ebx, @exp
   push dword[ebp+16]
   push dword[ebp+12]
   push ebx
   push eax
   push esi
   call BcdToString
; −−−−−−−−−−−−−−−− ripristino
.ca: ; ricarica il numero nello fpu−stack
fld tword[esp]
; st = num
   inc eax ; per il '−' o il '+' iniziale
.cf:
fldcw @wc_temp
   %undef @string
   %undef @exp
   %undef @wc_temp
   %undef @bcd
   add esp, 64
```

```
   pop esi
   pop edx
   pop ecx
   pop ebx
   mov esp, ebp
   pop ebp
   ret 12


; eax:len
; BcdToString(char* string, char* bcd_number, uint exp, uint intN,
uint decN)
; k == 0k, 4ra, 8@string, 12@bcd_number, 16@exp, 20IntN, 24decN
BcdToString:
   push ebp
   mov ebp, esp
   push ebx
   push ecx
   push edx
   push esi
   push edi
   sub esp, 64
   %define @string [ebp+8]
   %define @bcd_number [ebp+12]
   %define @exp [ebp+16]
   %define @IntN [ebp+20]
   %define @DecN [ebp+24]
   mov edi, esp
   mov esi, @bcd_number
   mov ebx, @exp
   mov edx, esi
   add esi, 8
   mov byte[edi], '0' ; serve per eventuale arrotondamento
   inc edi
   mov al, [esi]
   mov cl, [esi]
   and cl, 0xf
   shr al, 4
   add al, '0'
   add cl, '0'
   cmp al, '0'
   jne .c0
   mov [edi], cl
   inc edi
   dec esi
   jmp short .c1
.c0:
mov [edi], al
inc ebx
mov [edi+1], cl
dec esi
```

```
add edi, 2
.c1:
mov al, [esi]
mov cl, [esi]
and cl, 0xf
shr al, 4
add al, '0'
add cl, '0'
   mov [edi], al
   inc edi
   mov [edi], cl
   dec esi
   inc edi
   cmp esi, edx
   jae .c1
   mov byte[edi], 0 ; ora in s[0..17]
```