

## Re: my first .dll: why doesn't it work ...

**Source:** <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2004-11/1327.html>

---

**From:** Ro (*inp\_out\_at\_sim.tim*)

**Date:** 11/29/04

Date: Mon, 29 Nov 2004 09:31:43 GMT

Someone have found errors in fpow10i and in bcdtostring  
How many errors do you see?

```
; This program is free software; you can redistribute it and/or modify
; it under the terms of the GNU General Public License as published by
; the Free Software Foundation; either version 2 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU General Public License for more details.
;
; You should have received a copy of the GNU General Public License
; along with this program; if not, write to the Free Software
; Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
;
; *Non* e' consentito usare le funzioni atou, atoi, uPrint, utoa,
; itoa, utoh, hPrint, AsciiToSt0, converti_exp, converti, _mult10,
; St0ToAscii, BcdToString, round, IncArray scritte qui sotto
; per usi militari o di spionaggio o nei campi biologico (quindi
; anche genetica), farmaceutico, nucleare

; con a6
; nasm -f obj prog.asm
; alink -oPE prog.obj dlg.res
section .text use32
section .data use32
section .const use32
section .bss use32
section .text
section .data

%macro str 2+
    [section .data]
    align 4 , db 0
    %1 db %2
    dd 0
```

```
__SECT__
%endmacro

%define NL 13, 10
%define STDIN -10
%define STDOUT -11
%define IDI_ICON 100
%define IDR_MENU 2000
%define IDM_RANDOM 2001
%define IDM_MAX_RANDOM 2002
%define IDM_RANGE_RANDOM 2003
%define IDM_GAUSS 2004
%define IDM_LINEARE 2007
%define IDM_LINEARE1 2008

%define IDM_EXIT 2005
%define IDM_HELP_ABOUT 2006
%define IDC_ARROW 07f00h
    IDCANCEL equ 2h

extern ReadFile, WriteFile, GetStdHandle, ExitProcess
import ReadFile kernel32.dll
import WriteFile kernel32.dll
import GetStdHandle kernel32.dll
import ExitProcess kernel32.dll
extern GetModuleHandleA, GetTickCount
import GetModuleHandleA kernel32.dll
import GetTickCount kernel32.dll
extern DialogBoxParamA, EndDialog, MessageBoxA, SetDlgItemTextA
import DialogBoxParamA user32.dll
import EndDialog user32.dll
import MessageBoxA user32.dll
import SetDlgItemTextA user32.dll
extern LoadIconA, LoadCursorA, RegisterClassA, LoadMenuA,
ShowWindow
import LoadIconA user32.dll
import LoadCursorA user32.dll
import RegisterClassA user32.dll
import LoadMenuA user32.dll
import ShowWindow user32.dll
extern CreateWindowExA, DestroyWindow, PostQuitMessage,
UpdateWindow
import CreateWindowExA user32.dll
import DestroyWindow user32.dll
import PostQuitMessage user32.dll
import UpdateWindow user32.dll
extern TranslateMessage, DispatchMessageA, GetMessageA
import TranslateMessage user32.dll
import DispatchMessageA user32.dll
import GetMessageA user32.dll
extern DefWindowProcA, ReleaseDC, wsprintfA
```



alt.lang.asm: Re: my first .dll: why doesn't it work ...

```
dt 1.0e128 , 1.0e144 , 1.0e160 , 1.0e176 , 1.0e192 , 1.0e208 ,
1.0e224 , 1.0e240
dieci_256 dt 1.0 , 1.0e256 , 1.0e512 , 1.0e768 , 1.0e1024 ,
1.0e1280 , 1.0e1536 , 1.0e1792
dt 1.0e2048 , 1.0e2304 , 1.0e2560 , 1.0e2816 , 1.0e3072 ,
1.0e3328 , 1.0e3584 , 1.0e3840
dt 1.0e4096 , 1.0e4352 , 1.0e4608 , 1.0e4864 ; 20=0..19
```

```
WM_INITDIALOG equ 0110h
WM_COMMAND equ 0111h
WM_PAINT equ 0fh
WM_DESTROY equ 2h
WM_CLOSE equ 10h
WM_NCPAINT equ 085h
WM_ERASEBKGD equ 14h
WM_CTLCOLOREDIT equ 133h
WM_MOUSEMOVE equ 200h
WM_NCMOUSEMOVE equ 0A0h
```

```
MB_OK equ 0
MB_SYSTEMMODAL equ 01000h
EM_SETSEL equ 0b1h
SRCCOPY equ 0cc0020h
SW_SHOW equ 5h
RDW_INVALIDATE equ 01h
RDW_INTERNALPAINT equ 02h
RDW_ERASE equ 04h
WS_EX_CLIENTEDGE equ 200h
WS_OVERLAPPEDWINDOW equ 0cf0000h
WS_VISIBLE equ 10000000h
```

```
FirstMessage dd 0 , 0 , 0 , 0 , 0 , 0 , 0
```

```
struc rect1
Left resd 1
Top resd 1
Right resd 1
Bottom resd 1
Stw resd 1
Sth resd 1
endstruc
```

```
%macro rect1_fill 7
[section .data]
%1 dd %2
dd %3
dd %4
dd %5
dd %6
dd %7
```

Re: my first .dll: why doesn't it work ...

```
__SECT__
%endm

    struc WindowClass
    style resd 1
    lpfnWndProc resd 1
    cbClsExtra resd 1
    cbWndExtra resd 1
    hInstance resd 1
    hIcon resd 1
    hCursor resd 1
    hbrBackground resd 1
    lpszMenuName resd 1
    lpszClassName resd 1
    endstruc

%macro window_fill 11
    [section .data]
    %1
    dd %2
    dd %3
    dd %4
    dd %5
    dd %6
    dd %7
    dd %8
    dd %9
    dd %10
    dd %11
    __SECT__
%endm

    window_fill wstr , 3 , MainWindowProc , 0 , 0 , 0 , 0 , 0 , 6 , 0
, WindowClassName
    [section .data]
    WindowClassName db 'Anything' , 0
    WindowCaption db 'Base App' , 0
    __SECT__
    [section .data]
    WindowHandle dd 0 ,0
    MenuHandle dd 0 ,0
    WindowX dd 50 ,0
    WindowY dd 50 ,0
    WindowW dd 500 ,0
    WindowH dd 350 ,0
    __SECT__

    [section .text]

..start:
    pushad
```

```
push 0
call [GetModuleHandleA]
mov [wstr+ hInstance], eax
push IDI_ICON
push dword[wstr+hInstance]
call [LoadIconA]
mov [wstr+ hIcon] , eax
push IDC_ARROW
push 0
call [LoadCursorA]
mov [wstr+ hCursor] , eax
push wstr
call [RegisterClassA]
push IDR_MENU
push dword[wstr+hInstance]
call [LoadMenuA]
mov [MenuHandle] , eax
push 0
push dword[wstr+hInstance]
push dword[MenuHandle]
push 0
push dword[WindowH]
push dword[WindowW]
push dword[WindowY]
push dword[WindowX]
push WS_OVERLAPPEDWINDOW | WS_VISIBLE
push WindowCaption
push WindowClassName
push WS_EX_CLIENTEDGE
call [CreateWindowExA]
mov [WindowHandle], eax ; 12 argomenti
push SW_SHOW
push dword[WindowHandle]
call [ShowWindow]
push dword[WindowHandle]
call [UpdateWindow]
finit
jmp short .L1
.While:
  push FirstMessage
  call [TranslateMessage]
  push FirstMessage
  call [DispatchMessageA]
.L1:
push 0
push 0
push 0
push FirstMessage
call [GetMessageA]
  cmp eax, 0
  jne .While
```

```
popad
push 0
call [ExitProcess]
```

```
; void MainWindowProc(Adr, msg, wpar, lpar)
; 0 k, 4 Ra, 8 P_Adr, 12 P_msg, 16 P_wpar, 20 P_lpar
MainWindowProc:
```

```
    push ebp
    mov  ebp, esp
    %define @Adressee [ebp+8]
    %define @Message [ebp+12]
    %define @wParam [ebp+16]
    %define @lParam [ebp+20]
```

```
-----
;
    cmp  dword @Message, WM_CLOSE
    jne  .Else_If0
    push dword @Adressee
    call [DestroyWindow]
    mov  eax, 1
    jmp  .End_If
```

```
.Else_If0:
    cmp  dword @Message, WM_DESTROY
    jne  .Else_If1
    push 0
    call [PostQuitMessage]
    mov  eax, 1
    jmp  .End_If
```

```
.Else_If1:
    cmp  dword @Message, WM_COMMAND
    je   .cont
    jmp  .Else
```

```
.cont:
    cmp  dword @wParam, IDM_EXIT
    jne  .Else_If2
    push 0
    push 0
    push WM_CLOSE
    push dword @Adressee
    call [SendMessageA]
    mov  eax, 1
    jmp  .End_If
```

```
.Else_If2:
    cmp  dword @wParam, IDM_RANDOM
    jne  .Else_If3
    fld  tword[numero]
    call Tprinth
```

```
    mov eax, 1
    jmp .End_If
.Else_If3:
cmp dword @wParam, IDM_MAX_RANDOM
jne .Else_If4
    fld tword[numero1]
    call Tprinth
    mov eax, 1
    jmp .End_If
.Else_If4:
cmp dword @wParam, IDM_RANGE_RANDOM
jne .Else_If5
    push str_pnt
    push stringa
    call AsciiToSt0
    call Tprinth
    mov eax, 1
    jmp .End_If
.Else_If5:
cmp dword @wParam, IDM_GAUSS
jne .Else_If5a
    fld tword[numero]
    push 2
    push 6
    push base
    call St0ToAscii
    fstp st0
; *MessageBoxA( 0, base, base, _ MB_OK || MB_SYSTEMMODAL _ );
    push eax
    push base
    call uPrint
    mov eax, 1
    jmp .End_If

.Else_If5a:
cmp dword @wParam, IDM_LINEARE
jne .Else_If5b
    mov eax, 1
    jmp .End_If
.Else_If5b:
cmp dword @wParam, IDM_LINEARE1
jne .Else_If6
    mov eax, 1
    jmp .End_If
.Else_If6:
cmp dword @wParam, IDM_HELP_ABOUT
jne .End_Ifn
    mov eax, 1
    jmp .End_If
.End_Ifn:
    mov eax, 0
```

```
jmp .End_If
```

```
.Else:
```

```
    push dword @iParam  
    push dword @wParam  
    push dword @Message  
    push dword @Addressee  
    call [DefWindowProcA]
```

```
.End_If:
```

```
    %undef @Addressee  
    %undef @Message  
    %undef @wParam  
    %undef @iParam  
    mov esp, ebp  
    pop ebp  
    ret 16
```

```
; 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 .c4
    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);
; utoaz( a < 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_1:
    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_1

    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_1

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)
; 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
.10:
mov al, bl
and al, 0Fh
add al, '0'
cmp al, '9'
jbe .11
add al, 7
```

```
.11:
    mov byte[esi], al
    dec esi
    shr ebx, 4
    cmp ecx, esi
    jbe .10

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

```
Tprinth:
    push ebx
    push eax
    fstp tword[num_temp] ; il valore in st0 e levato dallo
stack
    mov ebx, buffer
    push dword[num_temp]
    push ebx
    call utoh ; 32
    add ebx, 8
    mov byte[ebx], '#'
    inc ebx
    push dword[num_temp+4]
    push ebx
    call utoh ; +32
    add ebx, 8
    mov byte[ebx], '#'
    inc ebx
    push dword[num_temp+8]
    push ebx
    call utoh ; +16
    add ebx, 8
    mov byte[ebx], '#'
    inc ebx
    mov byte[ebx], '#'
    inc ebx
    mov byte[ebx], 0
    inc ebx
    push MB_OK | MB_SYSTEMMODAL
    push titolo
    push buffer
    push 0
    call [MessageBoxA]
```

```

pop eax
pop ebx
ret

```

```

; uso: str titolo, "Numero " ; hPrint(titolo, D*num);
; utoaz( a < 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
fstcw [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: ; 10^a
call fpow10i
;fist D*nim; uPrint(titolo, D*nim);

.c5: ; num, 10^a
fild qword[esp]
;fist D*nim; uPrint(titolo, D*nim);

    fmulp st1 ; 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:
mov ecx, eax
and ecx, 0xF
lea ecx, [ecx+4*ecx]
lea ecx, [2*ecx]
    FLD tword[dieci_1 + ecx]
    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 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 .c7
    sub al, '0'
    add ebp, eax
    jnc .b2
    inc edx
    jno .b2
    sub ebp, eax
    dec edx
    dec ecx
    jnz .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
    cld
    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
; per intPart=0 scrive il numero senza esponente se l'esponente
; permette (e<17 && e>0) 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, 32 ; 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 st0 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
```



```
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]"0num" il numero di bcd
mov eax, @IntN
cmp eax, 0
je .c2
jmp .c4

.c2: ; b=exp=2<17 c=2, r=15
cmp ebx, 17
ja .c3
    cmp ebx, 0
    je .c3 ; c=1..17
    mov edx, @DecN
    mov ecx, ebx
    jmp short .c5
.c3:
mov ecx, 1
mov edx, @DecN
jmp short .c5
; b=exp, c=IntN, r=DecN
.c4:
mov ecx, @IntN
mov edx, @DecN
.c5:
cmp ecx, 17
jb .c6
mov ecx, 17
.c6: ; a=17-IntN
mov eax, 17
sub eax, ecx
    cmp edx, eax
    jb .c7
    mov edx, eax
.c7:
mov esi, esp
lea edi, [esp+32]
    sub ebx, ecx ; i=s b==exp

; nota: c->1..17 r->17-c=0..16
; arrotonda l'array in s nella cifra scelta
mov eax, edx ; nessun arrotondamento
add eax, ecx
cmp eax, 17
je .c8
push 5
push eax
push esi
push edi
```

```
    call round
    mov esi, edi
; posiziona il punto nell'array
.c8:
mov edi, @string

.cm: ; puo' essere 0000num in un bcd?
cmp byte[esi], '0'
jne .cn
    inc esi
    jmp short .cm
.cn:
cmp byte[esi], '0'
jb .co
cmp byte[esi], '9'
ja .co
    jmp short .c9
.co:
mov dword[edi], '0.0'
add edi, 3
jmp short .cc

.c9:
mov al, [esi]
mov [edi], al
cmp al, 0
je .cb
inc esi
inc edi
dec ecx
jnz .c9
    cmp byte[esi], 0
    je .cb
    cmp edx, 0
    je .cb ; c=17: fine
    mov byte[edi], '.'
    inc edi
.ca:
mov al, [esi]
mov [edi], al
cmp al, 0
je .cb
inc esi
inc edi
dec edx
jnz .ca
.cb:
cmp ebx, 0
je .cc
    mov byte[edi], 'e'
    inc edi
```

```
    push ebx
    push edi
    call itoa
    add edi, eax
.cc:
mov byte[edi], 0
    mov eax, @string
    sub edi, eax
    mov eax, edi
    inc eax
    %undef @string
    %undef @bcd_number
    %undef @exp
    %undef @IntN
    %undef @DecN
    add esp, 64
    pop edi
    pop esi
    pop edx
    pop ecx
    pop ebx
    mov esp, ebp
    pop ebp
    ret 20

; eax:len
; round(char* array, char* origin, uint decimal[1..16], uint min)
; k = 0k, 4ra, 8@array, 12@origin, 16@decimal, 20@min
round:
    push ebp
    mov ebp, esp
    push ebx
    push ecx
    push esi
    push edi
    %define @array [ebp+8]
    %define @origin [ebp+12]
    %define @decimal [ebp+16]
    %define @min [ebp+20]
;-----
    mov esi, @origin
    mov edi, @array
    mov ebx, @decimal
    mov ecx, @min
    inc ebx
    cmp ecx, 9
    ja .ce
    cmp ecx, 0
    jb .ce
    cmp ebx, 1
    jle .ce
```

```
    cmp edi, 0
    je .ce
    cmp esi, 0
    je .ce
    jmp .c0
.ce:
mov eax, 0
jmp .cf
.c0:
add cl, '0'
add esi, ebx
add edi, ebx
    cmp [esi], cl
    jbe .c1
    dec ebx
    push ebx
    push dword @origin
    push dword @array
    call IncArray
    mov eax, ebx
    inc eax
&
```