

## ascii to st0

**Source:** <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2004-11/0813.html>

---

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

**Date:** 11/18/04

Date: Thu, 18 Nov 2004 09:22:13 GMT

I don't know how to program fpu unit (I'm a poor beginner)  
How many errors do you see ? (expecily on fpu instruction)  
thank you

Ps

Where are those people that say "assembly" and "goto" are wrong and  
"C" and "structured programming" are good and right?

Why don't they write something near to routine ascii to st0 below in 3  
days in their favourite high level-structured language?

```
*****  
under Betov licence or under GNU General Public License  
; This program is distributed  
; WITHOUT ANY WARRANTY; without even the implied warranty of  
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  
this has origin from the fpu section of Betov assembly + Ng  
*****
```

```
alt-lang-asm  
/* 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
```

alt.lang.asm: ascii to st0

```
%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 ;

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

== SendMessageA, GetDlgItemTextA, LoadBitmapA, BeginPaint ==
user32.dll
== GetClientRect, EndPaint, GetDC, FillRect, RedrawWindow ==
user32.dll
== CreateCompatibleDC, SelectObject, BitBlt, DeleteDC, SetPixel ==
gdi32.dll

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_ERASEBKGDND 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
__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
lpzMenuName resd 1| lpzClassName 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, 6, 0,
WindowClassName
db, WindowClassName='Anything', WindowCaption='Base App';
dd, WindowHandle=0, MenuHandle=0, WindowX=50, WindowY=50,
WindowW=500, WindowH=350;
```

```
section .text
```

```
..start:
{pushad
[wstr+ hInstance] = *GetModuleHandleA(0);
[wstr+ hIcon] = *LoadIconA( D[wstr+hInstance] , IDI_ICON);
[wstr+ hCursor] = *LoadCursorA( 0, IDC_ARROW);
*RegisterClassA(wstr);
*MenuHandle = *LoadMenuA( D[wstr+hInstance], IDR_MENU );
*WindowHandle = *CreateWindowExA( WS_EX_CLIENTEDGE,
WindowClassName,
WindowCaption, _ WS_OVERLAPPEDWINDOW || WS_VISIBLE _
,
D*WindowX, D*WindowY, D*WindowW,
D*WindowH,
0, D*MenuHandle, D[wstr+hInstance],
0); /* 12 argomentii
*ShowWindow( D*WindowHandle, SW_SHOW);
*UpdateWindow( D*WindowHandle );
#.L1; finit;
.While:
{ *TranslateMessage(FirstMessage);
```

```

    *DispatchMessageA(FirstMessage);
.L1: *GetMessageA(FirstMessage, 0, 0, 0);
    a#.While
    }
popad
*ExitProcess(0);
}

[section .data]
dieci dd 10, 0
limiti dd 0x19999999, 0x99999999
/* se numero<=limiti => ok altrimenti e' troppo grande
base dd 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
seed dd 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
seed0 dd 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
index dd 0, 0
index1 dd 0, 0
buffer times 512 db 0
num_temp dt 0.0, 0.0
numero dt 12344444444444444444.0e6, 0.0, 0.0, 0.0
numero1 dt -12344444444444444444.0e5, 0.0
str_pnt dd 0
stringa db "12344444444444444444499999.0e001", 0
align 8, db 0
piu_oo dd 0, 0x80000000, 0x7FFF, 0
align 8, db 0
meno_oo dd 0, 0x80000000, 0xffff, 0
titolo db "Numero 80bits", 0
__SECT__

/* void MainWindowProc(Adr, msg, wpar, lpar)
/* 0 k, 4 Ra, 8 P_Adr, 12 P_msg, 16 P_wpar, 20 P_lpar
MainWindowProc:
{< k
    k = s;
    << @Adressee=[k+8], @Message=[k+12], @wParam=[k+16], @lParam=[k+20]
/*-----
    D @Message == WM_CLOSE !#.Else_If0
        { *DestroyWindow( D @Adressee ); a=1; ##.End_If; }
    .Else_If0: D @Message == WM_DESTROY !#.Else_If1
        { *PostQuitMessage(0); a=1; ##.End_If; }
    .Else_If1: D @Message == WM_COMMAND #.cont
        ##.Else
    .cont:
        { D @wParam == IDM_EXIT !#.Else_If2
            { *SendMessageA( D @Adressee, WM_CLOSE, 0, 0); a=1;
##.End_If}
    .Else_If2: D @wParam == IDM_RANDOM !#.Else_If3
        { fld T*numero; Tprinth(); a=1; ##.End_If }
    .Else_If3: D @wParam == IDM_MAX_RANDOM !#.Else_If4
        { fld T*numero1; Tprinth(); a=1; ##.End_If }

```

alt.lang.asm: ascii to st0

```

.Else_If4: D @wParam == IDM_RANGE_RANDOM !#.Else_If5
  {AsciiToSt0(stringa , str_pnt); Tprinth(); a=1; ##.End_If
}
.Else_If5: D @wParam == IDM_GAUSS !#.Else_If5a
  {a=1; ##.End_If }
.Else_If5a: D @wParam == IDM_LINEARE !#.Else_If5b
  {a=1; ##.End_If }
.Else_If5b: D @wParam == IDM_LINEARE1 !#.Else_If6
  {a=1; ##.End_If }
.Else_If6: D @wParam == IDM_HELP_ABOUT !#.End_Ifn
  {a=1; ##.End_If }
.End_Ifn:
  a=0; ##.End_If;
}
.Else:
  { *DefWindowProcA( D @Adressee, D @Message, D @wParam, D
@lParam); }
.End_If:
>> @Adressee, @Message, @wParam, @lParam
  s = k
> k;
ret 16
}

```

str DecString, " in decimale"

```

/* uso: str .titolo, "Numero " ; uPrint(titolo, D*num);
/* utoaz( a < char* Title, unsigned n)
/* 0 a, 4 Ra, 8 1P_Title, 12 1P_n
uPrint:
{< a
<< @Title=[s+8], @n=[s+12]
/*-----*/
a=@n; utoa(DecString, a); a=@Title;
*MessageBoxA( 0, DecString, a, _ MB_OK || MB_SYSTEMMODAL _ );
>> @Title, @n
> a
ret 8
}

```

```

/* utoa(char* string, unsigned n)
/* 0 k, 4 j, 8 i, 12 r, 16 b, 20 Ra, 24 1P_string, 28 1P_n
utoa:
{< b, r, i, j, k
<< @string=[s+24], @n=[s+28]
/*-----*/
i=@string; b=10; a=@n; j=i; k=0;
.utoaz_l:
  {r ^= r
  div b /* a= (r:a)/b r= (r:a)%b
  rl+='0'; *j=rl; ++j; ++k

```

```

        a#.utoaz_l
    }
    B*j=0; --j | #.vai
.utoaz_f:
    { al=*i; ah=*j;
      *j=al; *i=ah;
      ++i | --j;
    .vai:
      i<j #.utoaz_f
    }
    a=k;
.utoaz_z:
>> @string, @n
> b, r, i, j, k
ret 8
}

```

```

/* void utoh(char* Title, uint n)
/* 0 a, 4 c, 8 b, 12 i, 16 Ra, 20 1P_Title, 24 1P_n
utoh:
{< a, c, b, i
<< @n=[s+24], @Title=[s+20]
    b=@n; c=@Title; i=c; i+=7;
    .10:
        { al=bl; al&=0Fh; al+='0';
          al>'9'!.11; al+= 7;
        .11:
            B*i=al; --i; b>>=4; c<=i#.10
        }
>> @n, @Title
> a, c, b, i
ret 8
}

```

```
str HexprintString, " h"
```

```

/* uso: str titolo, "Numero " ; hPrint(titolo, D*num);
/* utoaz( a < char* Title, unsigned n)
/* 0 a, 4 Ra, 8 1P_Title, 12 1P_n
hPrint:
{< a
<< @Title=[s+8], @n=[s+12]
/*-----*/
a=@n; utoh(HexprintString, a);
a=@Title;
*MessageBoxA( 0, HexprintString, a, _MB_OK || MB_SYSTEMMODAL _ );
>> @Title, @n
> a
ret 8
}

```

```

Tprintu:
< b
  fstp T*num_temp /* il valore in st0 e levato dallo stack
  b=buffer;
  utoa(b, D*num_temp ); b+=a; B*b='#'; ++b /* 32
  utoa(b, D[num_temp+4]); b+=a; B*b='#'; ++b /* +32
  utoa(b, D[num_temp+8]); b+=a; B*b='#'; ++b /* +16
  B*b=0; ++b;
  *MessageBoxA( 0, buffer, titolo, _MB_OK || MB_SYSTEMMODAL _ );
> b
ret

```

```

Tprinth:
< b, a
  fstp T*num_temp /* il valore in st0 e levato dallo stack
  b=buffer;
  utoh(b, D*num_temp ); b+=8; B*b='#'; ++b /* 32
  utoh(b, D[num_temp+4]); b+=8; B*b='#'; ++b /* +32
  utoh(b, D[num_temp+8]); b+=8; B*b='#'; ++b; B*b='#'; ++b /* +16
  B*b=0; ++b;
  *MessageBoxA( 0, buffer, titolo, _MB_OK || MB_SYSTEMMODAL _ );
> b, a
ret

```

```

[section .bss]
  align 16, db 0
  cw_temp resw 12
__SECT__

```

```

[section .data]
  cw_data dw 0000001101111111b
  num0 dd 0
  num1 dd 0
  num00 dd 0
  num11 dd 0
  sign dd 0
  nim dd 0
__SECT__

```

```

/* 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 st0, e st1
/* 0k, 4j, 8i, 12r, 16c, 20b, 24Ra, 28P_string, 32P_address
AsciiToSt0:
< b, c, r, i, j, k
  fnstcw [cw_temp];
  fldcw [cw_data]; /* resetta tutto + troncamento
  i ^= 28; converti(); jnc .a0
  a ^= 28; b ^= 32; *b = a; fldz; ##.cx
.a0: [num0]=k; [num1]=r; [sign]=b; a=0;

```

```

    #.c1 /* leva gli spazi
.c0: ++j| .c1: B*j==' '#.c0;
    bl=B*j;
    bl=='e' #.c2 /* 44.44 e -12455
    bl!='E' #.c3
.c2: converti_exp(); /* in a esponente
    a> 4931 ?#.ce_piu_oo
    c> 4931 ?#.ce_piu_oo
    a< -4931 ?#.ce_zero
.c3: a+=c
    /* uPrint(titolo, a);
    a> 4931 ?#.ce_piu_oo
    a< -4931 ?#.ce_zero
    #.c4
.ce_piu_oo: D[sign]==0!#.seielmeno |fld T*piu_oo |##.cq;
.seielmeno: fld T*meno_oo; ##.cq;
.ce_zero: fldz; ##.cf;
.c4: fpow10(); /* 10^a
    /* wait
    /* fist D*nim
    /* uPrint(titolo, D*nim);

.c5: fld Q[num0];
    /* wait
    /* fist D*nim
    /* uPrint(titolo, D*nim);

    fmulp st1
.cf: D*sign==0 #.cq; FCHS;
.cq: a=^32; *a=j;
.cx: fldcw [cw_temp];
> b, c, r, i, j, k
ret 8

/* <j>pos <a>num+sign converti_exp<j>
converti_exp:
< b, c, r, i, k
    i=j; b=0; r=0;
.c0: ++j| B*j==' '#.c0; /* leva gli spazi
    B*j=='+' #.c1 | ++j; #.c2
.c1: B*j=='-' #.c2 | ++j; ++b;
.c2: B*j>'9' #.ce
    B*j<'0' #.ce /* nel caso dopo eventuale + o -
    #.c3
.ce: j=i; a=0; ##.cf
.c3: #.c5 /* leva gli zeri
.c4: ++j| .c5: B*j=='0' #.c4;
    a=0; c=0; k=0;
.c6: cl=*j
    cl<='9' !#.c8
    cl>='0' !#.c8

```

```

    mul D*dieci;
    cl--'0'; a+=c;
    a>4931#.ck
    ++j | #.c6;
.ck: ++j; cl=*j
    cl<='9' !#.c8
    cl>='0' !#.c8 | #.ck
.c8: b==1!#.cf | neg a
.cf:
> b, c, r, i, k
ret

[section .data]
/* 0 1 2 3 4 5
dieci_1 dt 1.0 , 10.0 , 100.0, 1000.0, 10000.0, 100000.0
    dt 1.0e6 , 1.0e7, 1.0e8, 1.0e9 , 1.0e10 , 1.0e11, 1.0e12,
1.0e13
    dt 1.0e14, 1.0e15
dieci_16 dt 1.0 , 1.0e16 , 1.0e32 , 1.0e48 , 1.0e64 ,
1.0e80 , 1.0e96, 1.0e112
    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, 1.0e4096
    dt 1.0e4352, 1.0e4608, 1.0e4864
__SECT__

/* input a; output 10^a in st0
/* utilizza due posizioni dello fpw-stack
/* non inizializza niente rounding mode per troncamento
/* settato all'esterno
fpow10:
< a, b, c
    a< -4931 ?#.err
    a> 4931 ?#.err
    #.c0
.err: fldz; ##.fine
.c0: b=0; a&0x80000000 | jz .c1 | neg a | b=1
.c1: c=a; c&=0xF; c= &[c+4*c]; c= &[2*c];
    FLD T[dieci_1 + c];
    c=a; c>>= 4; c&=0xF; c= &[c+4*c]; c= &[2*c];
    FLD T[dieci_16 + c];
    c=a; cl=ch; FMULP st1; c&=0xF;
    c= &[c+4*c]; c= &[2*c];
    FLD T[dieci_256 + c];
    b<>1; FMULP st1; jnz .fine
    FLD1 | FDIVRP st1
.fine:
> a, b, c

```

```

ret

/* segno in b
/* numero in r:k, numero cifre in c, ultima posizione in j
/* nel caso di errore o nessun numero stc altrimenti clc
/* converti<origine i>
/* 0 @sign, 4 i, 8 a, 12 Ra
converti:
< a, i /* i punta sempre al carattere
    /* corrente della stringa
<< @sign=[s]
    s--4
    #.c1 /* leva gli spazi
.c0: ++i| .c1: B*i==' ' #.c0;
    D @sign=0;
    B*i=='+' !#.b0 | ++i; #.c2
.b0: B*i=='-' !#.c2 | ++i; ++D @sign;
.c2: B*i>'9' #.ce
    B*i<'0' #.ce /* nel caso dopo eventuale + o -
    #.b1; /* c'e' un non numero => errore
.ce: r=0; a=0; c=0; b=0; j=^4; ##.cf
.b1: #.g1 /* leva gli zeri
.g0: ++i| .g1: B*i=='0' #.g0;
    a=0; r=0; k=0; j=0; c=0;
.c3: al=*i /* prende parte non esponente
    al<='9' !#.c4
    al>='0' !#.c4
    D*num00=k; D*num11=r;
    _mult10(); jc .c7
    al='0'; k+=a | jnc .b2; ++r | jo .c7
.b2: ++i| #.c3;
.c4: al==' ' #.a1 | ##.c9;
.a1: B[i+1]>'9'#.a2 | B[i+1]<'0'#.a2
    #.a3
.a2: ##.c9; /* caso numero.?non_numero
.a3: j=i; /* caso numero.?numero
.ca: ++j; al=*j /* elimina gli zeri superflui
    al>'9'#.cb;
    al<'0'#.cb;
    #.ca;
.cb: --j; al=*j | al=='0'#.cb
    i=j!#.cd; --i; ##.c9 /* caso numero.000000000
.cd: i=j#.a0
    ++i; al=*i
    al<='9' !#.c9
    al>='0' !#.c9
    --c;
    _mult10(); jc .a0 /* elimina i restanti decimali
    al='0'; k+=a | jnc .a4; ++r | jo .a0
.a4: #.cd;

```

```
.c7: ++c; ++i; al=*i /* caso 999etc
      al<='9' !#.c8
      al>='0' #.c7
.c8: al=='!' !#.c9 /* caso 999etc.etc
      B[i+1]>'9'#.c9;
      B[i+1]<'0'#.c9;
      .a0: ++i; al=*i
      al<='9' !#.c9
      al>='0' #.a0
.c9: j=i; b=@sign;
.cf: s+=4 /* se va bene questo set clear flag==0
      j==[s] !#.cz |stc; /* [s]==i
.cz:
>> @sign
> a, i
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:
{< a, b, c, i, j
/*-----*/
  c=r; j=k; a=k; /* salva input in c:j
  mul D*dieci; /* k*10
  i=a; k=r; /* save result in i:k

  a=c; mul D*dieci;
  a+=k | jnc .m0 | ++r;
.m0:
  r!=0 #.mfineset;
  r=a; k=i;
  r&0x80000000|jnz .mfineset
  /* qui perde un bit per il segno
  clc; #.mfine
.mfineset:
  r=c; k=j;
  stc;
.mfine:
> a, b, c, i, j
ret
}
```

---

```
; con a6
; nasm -f obj prog.asm
; alink -oPE prog.obj dlq.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
import DefWindowProcA user32.dll
import ReleaseDC user32.dll
import wsprintfA user32.dll

```

```

extern SendMessageA, GetDlgItemTextA, LoadBitmapA, BeginPaint
import SendMessageA user32.dll
import GetDlgItemTextA user32.dll
import LoadBitmapA user32.dll
import BeginPaint user32.dll
extern GetClientRect, EndPaint, GetDC, FillRect, RedrawWindow
import GetClientRect user32.dll
import EndPaint user32.dll
import GetDC user32.dll
import FillRect user32.dll
import RedrawWindow user32.dll

```

```

extern CreateCompatibleDC, SelectObject, BitBlt, DeleteDC,
SetPixel

```

```

import CreateCompatibleDC gdi32.dll
import SelectObject gdi32.dll
import BitBlt gdi32.dll
import DeleteDC gdi32.dll
import SetPixel gdi32.dll

```

```

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, 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
__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
lpzMenuName resd 1
lpzClassName 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

```



\_\_SECT\_\_

```
; 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
    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
.Else_Ifn:
mov eax, 0
jmp .End_If

.Else:
push dword @lParam
push dword @wParam
push dword @Message
push dword @Adressee
call [DefWindowProcA]

.End_If:
%undef @Adressee
%undef @Message
%undef @wParam
%undef @lParam
mov esp, ebp
pop ebp

```

```
ret 16
```

```
str DecString , " in decimale"
```

```
; uso: str .titolo, "Numero " ; uPrint(titolo, D*num);
; utoaz( a < char* Title, unsigned n)
; 0 a, 4 Ra, 8 1P_Title, 12 1P_n
uPrint:
    push eax
    %define @Title [esp+8]
    %define @n [esp+12]
;-----*/
    mov eax, @n
    push eax
    push DecString
    call utoa
    mov eax, @Title
    push MB_OK | MB_SYSTEMMODAL
    push eax
    push DecString
    push 0
    call [MessageBoxA]
    %undef @Title
    %undef @n
    pop eax
    ret 8
```

```
; utoa(char* string, unsigned n)
; 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_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

```

```

; void utoh(char* Title, uint n)
; 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
.i0:
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

    str HexprintString , " h"

; uso: str titolo, "Numero " ; hPrint(titolo, D*num);
; utoaz( a < char* Title, unsigned n)
; 0 a, 4 Ra, 8 1P_Title, 12 1P_n
hPrint:
    push eax
    %define @Title [esp+8]
    %define @n [esp+12]
;-----*/
    mov eax, @n
    push eax
    push HexprintString
    call utoh
    mov eax, @Title
    push MB_OK | MB_SYSTEMMODAL
    push eax
    push HexprintString
    push 0
    call [MessageBoxA]
    %undef @Title
    %undef @n
    pop eax
    ret 8

Tprintu:
    push ebx
    fstp tword[num_temp] ; il valore in st0 e levato dallo
stack
    mov ebx, buffer
    push dword[num_temp]

```

```

push ebx
call utoa ; 32
add ebx, eax
mov byte[ebx], '#'
inc ebx
push dword[num_temp+4]
push ebx
call utoa ; +32
add ebx, eax
mov byte[ebx], '#'
inc ebx
push dword[num_temp+8]
push ebx
call utoa ; +16
add ebx, eax
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 ebx
ret

```

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

```

```

[section .bss]
align 16 , db 0
cw_temp resw 12
__SECT__

```

```

[section .data]
cw_data dw 0000001101111111b
num0 dd 0
num1 dd 0
num00 dd 0
num11 dd 0
sign dd 0
nim dd 0
__SECT__

```

```

; 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 st0 e st1
; 0k, 4j, 8i, 12r, 16c, 20b, 24Ra, 28P_string, 32P_address

```

```

AsciiToSt0:
  push ebx
  push ecx
  push edx
  push esi
  push edi
  push ebp
  fnstcw [cw_temp]
  fldcw [cw_data] ; resetta tutto + troncamento
  mov esi, [ esp + 28 ]
  call converti
  jnc .a0
  mov eax, [ esp + 28 ]
  mov ebx, [ esp + 32 ]
  mov [ebx], eax
  fldz

```

```

    jmp .cx
.a0:
mov [num0], ebp
mov [num1], edx
mov [sign], 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[sign], 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 fpow10
; wait
; fist D*nim
; uPrint(titolo, D*nim);

.c5:

```

```

fild qword[num0]
; wait
; fist D*nim
; uPrint(titolo, D*nim);

    fmulp st1
.cf:
cmp dword[sign], 0
je .cq
FCHS
.cq:
mov eax, [ esp + 32 ]
mov [eax], edi
.cx:
fldcw [cw_temp]
    pop ebp
    pop edi
    pop esi
    pop edx
    pop ecx
    pop ebx
    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
.c3:
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

```

```

[section .data]
; 0 1 2 3 4 5

```

## alt.lang.asm: ascii to st0

```
dieci_1 dt 1.0 , 10.0 , 100.0 , 1000.0 , 10000.0 , 100000.0
dt 1.0e6 , 1.0e7 , 1.0e8 , 1.0e9 , 1.0e10 , 1.0e11 , 1.0e12 ,
1.0e13
dt 1.0e14 , 1.0e15
dieci_16 dt 1.0 , 1.0e16 , 1.0e32 , 1.0e48 , 1.0e64 ,
1.0e80 , 1.0e96 , 1.0e112
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 , 1.0e4096
dt 1.0e4352 , 1.0e4608 , 1.0e4864
__SECT__
```

```
; input a; output 10^a in st0
; utilizza due posizioni dello fpu-stack
; non inizializza niente rounding mode per troncamento
; settato all'esterno
```

```
fpow10:
```

```
push eax
push ebx
push ecx
cmp eax, -4931
jl .err
cmp eax, 4931
jg .err
jmp short .c0
```

```
.err:
```

```
fldz
```

```
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
```

```

mov cl, ch
FMULP st1
and ecx, 0xF
lea ecx, [ecx+4*ecx]
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 in r:k, numero cifre in c, ultima posizione in j
; nel caso di errore o nessun numero stc altrimenti clc
; converti<origine i>
; 0 @sign, 4 i, 8 a, 12 Ra
converti:
push eax
push esi ; 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 ; 'e' un non numero => errore
.ce:

```

```

mov edx, 0
mov eax, 0
mov ecx, 0
mov ebx, 0
mov edi, [ esp + 4 ]
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
    mov dword[num00], ebp
    mov dword[num11], edx
    call _mult10
    jc .c7
    sub al, '0'
    add ebp, eax
    jnc .b2
    inc edx
    jo .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:
dec edi
mov al, [edi ]
cmp al, '0'
je .cb
  cmp esi, edi
  jne .cd ; caso numero.000000000
  dec esi
  jmp .c9
.cd:
cmp esi, edi
je .a0
  inc esi
  mov al, [esi ]
  cmp al, '9'
  ja .c9
  cmp al, '0'
  jb .c9
  dec ecx
  call _mult10 ; elimina i restanti decimali
  jc .a0
  sub al, '0'
  add ebp, eax
  jnc .a4
  inc edx
  jo .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: ; se va bene questo set clear flag==0
add esp, 4
  cmp edi, [esp]
  jne .cz ; [s]==i
  stc
.cz:
  %undef @sign
  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
```