```
;##########################################################################
;
;                               FpuAbs
;
;##########################################################################

  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameter and allow additional data types for storage.
  ;
  ;                               |Src| -> Dest
  ;
  ; This FpuAbs function changes the sign of a REAL number (Src) to
  ; positive with the FPU and returns the result at the specified
  ; destination (the FPU itself or a memory location), unless an invalid
  ; operation is reported by the FPU or the definition of the parameters
  ; (with uID) is invalid.
  ;
  ; The source can only be a REAL number from the FPU itself or either a
  ; REAL4, REAL8 or REAL10 from memory. (The absolute value of integers can
  ; be easily obtained with CPU instructions.)
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF the source is specified to be the FPU top data register, it would be
  ; removed. It would then be replaced by the result only if the FPU is
  ; specified as the destination.
  ;
  ; IF the source is from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ##########################################################################

FpuAbs proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;------------------------------
```

```
;check if top register is empty
;------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

dest0:
      fabs

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return error

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
```

```
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was Src taken from FPU
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            ffree st(7)       ;free it if not already empty
            fld   tempst      ;return the result on the FPU
      .endif

      or    al,1              ;to insure EAX!=0
      ret

FpuAbs endp

; ########################################################################

end



; ########################################################################
;
;                             FpuAdd
;
;########################################################################

  ; ----------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU.
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;
  ;                           Src1 + Src2 -> Dest
  ;
  ; This FpuAdd function adds the numbers from two sources (Src1 and Src2)
  ; with the FPU and returns the result as a REAL number at the specified
  ; destination (the FPU itself or a memory location), unless an invalid
  ; operation is reported by the FPU or the definition of the parameters
  ; (with uID) is invalid.
  ;
  ; Either of the two sources can be:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory, or
  ; one of the FPU constants.
  ;
  ; None of the sources are checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
```

```
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuAdd proc public lpSrc1:DWORD, lpSrc2:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU or SRC2_FPU      ;is any data taken from FPU?
      jz    continue

;------------------------------
;check if top register is empty
;------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src1 and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src1 is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   src2        ;check next parameter for Src2
      .endif

      mov   eax,lpSrc1
      test  uID,SRC1_CONST
      jnz   constant
      test  uID,SRC1_REAL
      .if !ZERO?              ;Src1 is an 80-bit REAL10 in memory?
            fld   tbyte ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_REAL8
      .if !ZERO?              ;Src1 is a 64-bit REAL10 in memory?
            fld   qword ptr [eax]
```

```
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_REAL4
      .if !ZERO?              ;Src1 is a 32-bit REAL10 in memory?
            fld   dword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src1 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src1 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif

      test  uID,SRC1_DIMM
      .if   !ZERO?            ;Src1 is an immediate 32-bit integer?
            fild  lpSrc1
            jmp   src2        ;check next parameter for Src2
      .endif

      ;otherwise no valid ID for Src1

srcerr:
      frstor content
srcerr1:
      xor   eax,eax           ;error code
      ret

constant:
      cmp   eax,FPU_PI
      jnz   @F
      fldpi
      jmp   src2
   @@:
      cmp   eax,FPU_NAPIER
      jnz   srcerr            ;no correct CONST for Src1
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

;---------------------------------------
;check source for Src2 and load it to FPU
;---------------------------------------

src2:
      test  uID,SRC2_FPU
      .if   !ZERO?            ;Src2 is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28] ;retrieve it from the stored data
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc2
      test  uID,SRC2_CONST
      jnz   constant2
      test  uID,SRC2_REAL
      .if   !ZERO?            ;Src2 is an 80-bit REAL10 in memory?
            fld   tbyte ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_REAL8
```

```
      .if   !ZERO?            ;Src2 is a 64-bit REAL10 in memory?
            fld   qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_REAL4
      .if   !ZERO?            ;Src2 is a 32-bit REAL10 in memory?
            fld   dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC2_DMEM
      .if   !ZERO?            ;Src2 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_QMEM
      .if   !ZERO?            ;Src2 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC2_DIMM
      .if   !ZERO?            ;Src2 is an immediate 32-bit integer?
            fild  lpSrc2
            jmp   dest0       ;go complete process
      .endif
      jmp   srcerr            ;no correct flag for Src2

constant2:
      cmp   eax,FPU_PI
      jnz   @F
      fldpi                   ;load pi (3.14159...) on FPU
      jmp   dest0             ;go complete process
   @@:
      cmp   eax,FPU_NAPIER
      jnz   srcerr            ;no correct CONST for Src2
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

dest0:
      fadd

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   eax,1             ;test for invalid operation
      jc    srcerr            ;clean-up and return error

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
```

```
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU or SRC2_FPU      ;was any data taken from FPU?
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            ffree st(7)       ;free it if not already empty
            fld   tempst      ;return the result on the FPU
      .endif

      or    al,1              ;to insure EAX!=0
      ret

FpuAdd endp

; ########################################################################

end



; ########################################################################
;
;                            FpuArccos
;
;########################################################################

  ; ----------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameter and allow additional data types for storage.
  ;
  ;                                 sqrt(1-Src^2)
  ;               acos(Src) = atan -------------  -> Dest
  ;                                      Src
  ;
  ; This FpuArccos function computes the angle in degrees or radians
  ; with the FPU corresponding to the cosine value provided in the source
  ; parameter (Src) and returns the result as a REAL number at the
  ; specified destination (the FPU itself or a memory location), unless
  ; an invalid operation is reported by the FPU or the definition of the
  ; parameters (with uID) is invalid.
  ;
  ; The source can only be a REAL number from the FPU itself or either a
  ; REAL4, REAL8 or REAL10 from memory. Its absolute value must be
  ; equal to or less than 1 (integers are thus not allowed).
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
```

```
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuArccos proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is data taken from FPU?
      jz    continue

;-------------------------------
;check if top register is empty
;-------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;----------------------------------------
;check source for Src and load it to FPU
;----------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
```

```
      .endif

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

   @@:
      mov   eax,lpSrc
      fld   tbyte ptr[eax]

dest0:
      fld   st                ;copy cosine value
      fmul  st,st             ;cos^2
      fld1
      fsubr                   ;1-cos^2 = sin^2
      fsqrt                   ;->sin
      fxch
      fpatan                  ;i.e. arctan(sin/cos)

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   eax,1             ;test for invalid operation
      jc    srcerr            ;clean-up and return error

      test  uID,ANG_RAD
      jnz   @F                ;jump if angle is required in radians
      pushd 180
      fimul dword ptr[esp]    ;*180 degrees
      fldpi                   ;load pi (3.14159...) on FPU
      fdiv                    ;*180/pi, angle now in degrees
      pop   eax               ;clean CPU stack

; store result as specified

@@:
      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was any data taken from FPU?
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            ffree st(7)       ;free it if not already empty
            fld   tempst      ;return the result on the FPU
      .endif

      or    al,1              ;to insure EAX!=0
```

```
      ret

FpuArccos endp

; ########################################################################

end



; ########################################################################
;
;                              FpuArccosh
;
;########################################################################

  ; ----------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameter and allow additional data types for storage.
  ;
  ;        acosh(Src) = ln[Src + sqrt(Src^2 - 1)] -> Dest
  ;
  ; This FpuArccosh function computes the number corresponding to the
  ; hyperbolic cosine value provided in the source parameter (Src) and
  ; returns the result as a REAL number at the specified destination
  ; (the FPU itself or a memory location), unless an invalid operation is
  ; reported by the FPU or the definition of the parameters (with uID) is
  ; invalid.
  ;
  ; The source can be either
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory.
  ; The source must be greater than 1, otherwise an invalid operation
  ; is reported.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ----------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuArccosh proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD
```

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;-------------------------------
;check if top register is empty
;-------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src1 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src1 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DIMM     ;is Src an immediate 32-bit integer?
      jnz   @F                ;last valid ID for Src
```

```
srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

   @@:
      fild  lpSrc

dest0:
      fld   st(0)             ;copy it
      fmul  st,st(0)          ;Src^2
      fld1
      fsub                    ;Src^2-1
      fsqrt                   ;sqrt(Src^2-1)
      fadd                    ;Src+sqrt(Src^2-1)
      fldln2                  ;->ln(2)=1/[log2(e)]
      fxch
      fyl2x                   ;->[log2(Src+sqrt(Src^2-1))]/[log2(e)]
                              ; = ln[Src+sqrt(Src^2-1)]

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   eax,1             ;test for invalid operation
      jc    srcerr            ;clean-up and return error

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was any data taken from FPU?
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            ffree st(7)       ;free it if not already empty
            fld   tempst      ;return the result on the FPU
      .endif

      or    al,1              ;to insure EAX!=0
      ret

FpuArccosh endp

; ########################################################################

end
```

```
; ########################################################################
;
;                             FpuArcsin
;
;########################################################################

  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ; used as source parameter and allow additional data types for storage.
  ;
  ;                                  Src
  ;            asin(Src) = atan -------------  -> Dest
  ;                             sqrt(1-Src^2)
  ;
  ; This FpuArcsin function computes the angle in degrees or radians
  ; with the FPU corresponding to the sine value provided in the source
  ; parameter (Src) and returns the result as a REAL number at the
  ; specified destination (the FPU itself or a memory location), unless
  ; an invalid operation is reported by the FPU or the definition of the
  ; parameters (with uID) is invalid.
  ;
  ; The source can only be a REAL number from the FPU itself or either a
  ; REAL4, REAL8 or REAL10 from memory. Its absolute value must be
  ; equal to or less than 1 (integers are thus not allowed).
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuArcsin proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE
```

```
      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;-------------------------------
;check if top register is empty
;-------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

   @@:
      mov   eax,lpSrc
      fld   tbyte ptr[eax]

dest0:
      fld   st(0)             ;copy sine value
      fmul  st,st(0)          ;sin^2
      fld1
      fsubr                   ;1-sin^2 = cos^2
      fsqrt                   ;->cos
      fpatan                  ;i.e. arctan(sin/cos) = arcsin

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   eax,1             ;test for invalid operation
      jc    srcerr            ;clean-up and return error
```

```
      test  uID,ANG_RAD
      jnz   @F                ;jump if angle is required in radians
      pushd 180
      fimul dword ptr[esp]    ;*180 degrees
      fldpi                   ;load pi (3.14159...) on FPU
      fdiv                    ;*180/pi, angle now in degrees
      pop   eax               ;clean CPU stack

      ftst                    ;check for negative value
      fstsw ax                ;retrieve status word from FPU
      fwait
      sahf
      jnc   @F                ;jump if positive number
      pushd 360
      fiadd dword ptr[esp]    ;angle now 0-360
      fwait
      pop   eax               ;clean CPU stack

; store result as specified

@@:
      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was any data taken from FPU?
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            ffree st(7)       ;free it if not already empty
            fld   tempst      ;return the result on the FPU
      .endif

      or    al,1              ;to insure EAX!=0
      ret

FpuArcsin endp

; ########################################################################

end



; ########################################################################
;
;                             FpuArcsinh
;
;########################################################################
```

```
  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameter and allow additional data types for storage.
  ;
  ;        asinh(Src) = ln[Src + sqrt(Src^2 + 1)] -> Dest
  ;
  ; This FpuArcsinh function computes the number corresponding to the
  ; hyperbolic sine value provided in the source parameter (Src) and
  ; returns the result as an 80-bit REAL number at the specified destination
  ; (the FPU itself or a memory location), unless an invalid operation is
  ; reported by the FPU or the definition of the parameters (with uID) is
  ; invalid.
  ;
  ; The source can be either
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuArcsinh proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;------------------------------
;check if top register is empty
;------------------------------
```

```
      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src1 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src1 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DIMM     ;is Src an immediate 32-bit integer?
      jnz   @F                ;last valid ID for Src

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

   @@:
      fild  lpSrc

dest0:
      ftst
      fstsw ax                ;retrieve status flags from FPU
      fwait
      sahf
      jpe   srcerr            ;indeterminate value
      pushf                   ;save flags
      fabs                    ;use positive value
```

```
      fld   st(0)             ;copy it
      fmul  st,st(0)          ;Src^2
      fld1
      fadd                    ;Src^2+1
      fsqrt                   ;sqrt(Src^2+1)
      fadd                    ;Src+sqrt(Src^2+1)
      fldln2                  ;->ln(2)=1/[log2(e)]
      fxch
      fyl2x                   ;->[log2(Src+sqrt(Src^2+1))]/[log2(e)]
                              ; = ln[Src+sqrt(Src^2+1)]
      popf
      .if   CARRY?            ;if value was originally negative
            fchs              ;make result negative
      .endif
      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   eax,1             ;test for invalid operation
      jc    srcerr            ;clean-up and return error

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was any data taken from FPU?
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            ffree st(7)       ;free it if not already empty
            fld   tempst      ;return the result on the FPU
      .endif

      or    al,1              ;to insure EAX!=0
      ret

FpuArcsinh endp

; ########################################################################

end



; ########################################################################
;
;                             FpuArctan
;
;########################################################################
```

```
  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameter and allow additional data types for storage.
  ;
  ;                        atan(Src) -> Dest
  ;
  ; This FpuArctan function computes the angle in degrees or radians
  ; with the FPU corresponding to the tangent value provided in the source
  ; parameter (Src) and returns the result as a REAL number at the
  ; specified destination (the FPU itself or a memory location), unless
  ; an invalid operation is reported by the FPU or the definition of the
  ; parameters (with uID) is invalid.
  ;
  ; The source can only be a REAL number from the FPU itself or either a
  ; REAL4, REAL8 or REAL10 from memory.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuArctan proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;------------------------------
;check if top register is empty
;------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
```

```
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;--------------------------------------
;check source for Src and load it to FPU
;--------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

   @@:
      mov   eax,lpSrc
      fld   tbyte ptr[eax]

dest0:
      fld1
      fpatan                  ;i.e. arctan(Src/1)

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   eax,1             ;test for invalid operation
      jc    srcerr            ;clean-up and return error

      test  uID,ANG_RAD
      jnz   @F                ;jump if angle is required in radians
      pushd 180
      fimul dword ptr[esp]    ;*180 degrees
      fldpi                   ;load pi (3.14159...) on FPU
      fdiv                    ;*180/pi, angle now in degrees
      pop   eax               ;clean CPU stack

      ftst                    ;check for negative angle
      fstsw ax                ;retrieve status word from FPU
      fwait
      sahf
      jnc   @F                ;jump if positive number
      pushd 360
      fiadd dword ptr[esp]    ;angle now 0-360
```

```
      fwait
      pop   eax               ;clean CPU stack

; store result as specified

@@:
      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was any data taken from FPU?
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            ffree st(7)       ;free it if not already empty
            fld   tempst      ;return the result on the FPU
      .endif

      or    al,1              ;to insure EAX!=0
      ret

FpuArctan endp

; ########################################################################

end



; ########################################################################
;
;                            FpuArctanh
;
;########################################################################

  ; ----------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameter and allow additional data types for storage.
  ;
  ;         atanh(Src) = 0.5 * ln[(1+Src)/(1-Src)] -> Dest
  ;
  ; This FpuArctanh function computes the number corresponding to the
  ; hyperbolic tangent value provided in the source parameter (Src) and
  ; returns the result as a REAL number at the specified destination
  ; (the FPU itself or a memory location), unless an invalid operation is
  ; reported by the FPU or the definition of the parameters (with uID) is
  ; invalid.
```

```
  ;
  ; The source can only be a REAL number from the FPU itself or either a
  ; REAL4, REAL8 or REAL10 from memory. Its absolute value must be
  ; lower than 1, otherwise an invalid operation is reported.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuArctanh proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;-------------------------------
;check if top register is empty
;-------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
```

```
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

   @@:
      mov   eax,lpSrc
      fld   tbyte ptr[eax]

dest0:
      fld   st(0)             ;copy it
      fld1
      fadd                    ;1+Src
      fxch
      fld1
      fsubr                   ;1-Src
      fdiv                    ;(1+Src)/(1-Src)
      fldln2                  ;->ln(2)=1/[log2(e)]
      fxch
      fyl2x                   ;->[log2((1+Src)/(1-Src))]/[log2(e)]
                              ; = ln[(1+Src)/(1-Src)]
      fld1
      fchs
      fxch
      fscale                  ;->0.5*ln[(1+Src)/(1-Src)] = atanh(Src)
      fstp  st(1)             ;store over scaling factor

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   eax,1             ;test for invalid operations
      jc    srcerr            ;clean-up and return error

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
```

```
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was any data taken from FPU?
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            ffree st(7)       ;free it if not already empty
            fld   tempst      ;return the result on the FPU
      .endif

      or    al,1              ;to insure EAX!=0
      ret

FpuArctanh endp

; ########################################################################

end



; ########################################################################
;
;                              FpuAtoFL
;
;########################################################################

  ; ----------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified January, 2004, to eliminate .data section and remove some
  ; redundant code.
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised December 2006 to avoid a minuscule error when processing strings
  ; which do not have any decimal digit.
  ; Revised January 2010 to allow additional data types for storage.
  ;
  ; This FpuAtoFL function converts a decimal number from a zero terminated
  ; alphanumeric string format (Src) to a REAL number and returns the
  ; result as a REAL number at the specified destination (the FPU itself
  ; or a memory location), unless an invalid operation is reported by
  ; the FPU or the definition of the parameters (with uID) is invalid.
  ;
  ; The source can be a string in regular numeric format or in scientific
  ; notation. The number of digits (excluding all leading 0's and trailing
  ; decimal 0's) must not exceed 18. If in scientific format, the exponent
  ; must be within +/-4931
  ;
  ; The source is checked for validity. The procedure returns an error if
  ; a character other than those acceptable is detected prior to the
  ; terminating zero or the above limits are exceeded.
  ;
  ; This procedure is based on converting the digits into a specific packed
  ; decimal format which can be used by the FPU and then adjusted for an
  ; exponent of 10.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
```

```
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuAtoFL proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE
LOCAL bcdstrf      :TBYTE
LOCAL bcdstri      :TBYTE

      fsave content

      push  ebx
      push  ecx
      push  edx
      push  esi
      push  edi
      xor   eax,eax
      xor   ebx,ebx
      xor   edx,edx
      lea   edi,bcdstri
      stosd
      stosd
      stosd
      stosd
      stosd
      lea   edi,bcdstri+8
      mov   esi,lpSrc
      mov   ecx,18
   @@:
      lodsb
      cmp   al," "
      jz    @B                ;eliminate leading spaces
      or    al,al             ;is string empty?
      jnz   @F

atoflerr:
      frstor content
atoflerr1:
      xor   eax,eax
      pop   edi
      pop   esi
      pop   edx
      pop   ecx
      pop   ebx
      ret

;----------------------
;check for leading sign
;----------------------

   @@:
      cmp   al,"+"
      jz    @F
      cmp   al,"-"
      jnz   integer
      mov   ah,80h
   @@:
```

```
      mov   [edi+1],ah        ;put sign byte in bcd strings
      mov   [edi+11],ah
      xor   eax,eax
      lodsb

;-----------------------------------------
;convert the integer digits to packed decimal
;-----------------------------------------

integer:
      cmp   al,"."
      jnz   @F
      lea   edi,bcdstri
      call  load_integer
      lodsb
      lea   edi,bcdstrf+8
      mov   cl,18
      and   bh,4
      jmp   decimals

   @@:
      cmp   al,"e"
      jnz   @F
      .if   cl == 18
            jmp   atoflerr    ;error if no digit other than 0 before e
      .endif
      lea   edi,bcdstri
      call  load_integer
      jmp   scient

   @@:
      cmp   al,"E"
      jnz   @F
      .if   cl == 18
            jmp   atoflerr    ;error if no digit other than 0 before E
      .endif
      lea   edi,bcdstri
      call  load_integer
      jmp   scient

   @@:
      or    al,al
      jnz   @F
      test  bh,4
      jz    atoflerr          ;error if no numerical digit before terminating 0
      lea   edi,bcdstri
      call  load_integer
      jmp   laststep

   @@:
      sub   al,"0"
      jc    atoflerr          ;unacceptable character
      jnz   @F
      test  bh,2
      jnz   @F
      or    bh,4              ;at least 1 numerical character
      lodsb
      jmp   integer
   @@:
      cmp   al,9
      ja    atoflerr          ;unacceptable character
      or    bh,6              ;at least 1 non-zero numerical character
      sub   ecx,1
      jc    atoflerr          ;more than 18 integer digits
      mov   ah,al

      lodsb
      cmp   al,"."
      jnz   @F
      mov   al,0
```

```
      ror   ax,4
      mov   [edi],al
      lea   edi,bcdstri
      call  load_integer
      lea   edi,bcdstrf+8
      mov   cl,18
      and   bh,4
      lodsb
      jmp   decimals

   @@:
      cmp   al,"e"
      jnz   @F
      mov   al,0
      ror   ax,4
      mov   [edi],al
      lea   edi,bcdstri
      call  load_integer
      jmp   scient

   @@:
      cmp   al,"E"
      jnz   @F
      mov   al,0
      ror   ax,4
      mov   [edi],al
      lea   edi,bcdstri
      call  load_integer
      jmp   scient

   @@:
      or    al,al
      jnz   @F
      ror   ax,4
      mov   [edi],al
      lea   edi,bcdstri
      call  load_integer
      jmp   laststep

   @@:
      sub   al,"0"
      jc    atoflerr          ;unacceptable character
      cmp   al,9
      ja    atoflerr          ;unacceptable character
      dec   ecx
      rol   al,4
      ror   ax,4
      mov   [edi],al
      dec   edi
      lodsb
      jmp   integer

;-------------------------------------------
;convert the decimal digits to packed decimal
;-------------------------------------------

decimals:
      cmp   al,"e"
      jnz   @F
      lea   edi,bcdstrf
      call  load_decimal
      jmp   scient

   @@:
      cmp   al,"E"
      jnz   @F
      lea   edi,bcdstrf
      call  load_decimal
      jmp   scient
```

```
   @@:
      or    al,al
      jnz   @F
      test  bh,4
      jz    atoflerr          ;error if no numerical digit before terminating 0
      lea   edi,bcdstrf
      call  load_decimal
      jmp   laststep

   @@:
      sub   al,"0"
      jc    atoflerr          ;unacceptable character
      cmp   al,9
      ja    atoflerr          ;unacceptable character
      or    bh,4              ;at least 1 numerical character
      .if   al != 0
            or    bh,2
      .endif
      sub   ecx,1
      jnc   @F
      .if   al == 0           ;if trailing decimal 0
            inc   ecx
            lodsb
            jmp   decimals
      .endif
      jmp   atoflerr
   @@:
      mov   ah,al

decimal1:
      lodsb
      cmp   al,"e"
      jnz   @F
      mov   al,0
      ror   ax,4
      mov   [edi],al
      lea   edi,bcdstrf
      call  load_decimal
      jmp   scient

   @@:
      cmp   al,"E"
      jnz   @F
      mov   al,0
      ror   ax,4
      mov   [edi],al
      lea   edi,bcdstrf
      call  load_decimal
      jmp   scient

   @@:
      or    al,al
      jnz   @F
      test  bh,4
      jz    atoflerr          ;error if no numerical digit before terminating 0
      mov   al,0
      ror   ax,4
      mov   [edi],al
      lea   edi,bcdstrf
      call  load_decimal
      jmp   laststep

   @@:
      sub   al,"0"
      jc    atoflerr          ;unacceptable character
      cmp   al,9
      ja    atoflerr          ;unacceptable character
      .if   al != 0
            or    bh,2        ;at least one non-zero decimal digit
      .endif
```

```
      dec   ecx
      rol   al,4
      ror   ax,4
      mov   [edi],al
      dec   edi
      lodsb
      jmp   decimals

laststep:
      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    atoflerr          ;clean-up and return error

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            ffree st(7)       ;free it if not already empty
            fld   tempst      ;return the result on the FPU
      .endif

      or    al,1              ;to insure EAX!=0
   @@:
      pop   edi
      pop   esi
      pop   edx
      pop   ecx
      pop   ebx
      ret

scient:
      xor   eax,eax
      xor   edx,edx
      lodsb
      cmp   al,"+"
      jz    @F
      cmp   al,"-"
      jnz   scient1
      stc
      rcr   eax,1     ;keep sign of exponent in most significant bit of EAX
   @@:
      lodsb                   ;get next digit after sign

scient1:
      push  eax
      and   eax,0ffh
      jnz   @F        ;continue if 1st byte of exponent is not terminating 0
```

```
scienterr:
      pop   eax
      jmp   atoflerr          ;no exponent

   @@:
      sub   al,30h
      jc    scienterr         ;unacceptable character
      cmp   al,9
      ja    scienterr         ;unacceptable character
      add   edx,edx           ;x2
      lea   edx,[edx+edx*4]   ;x2x5=x10
      add   edx,eax
      cmp   edx,4931
      ja    scienterr         ;exponent too large
      lodsb
      or    al,al
      jnz   @B
      pop   eax               ;retrieve exponent sign flag
      rcl   eax,1             ;is most significant bit set?
      jnc   @F
      neg   edx
   @@:
      call  XexpY
      fmul
      jmp   laststep

FpuAtoFL endp

; ########################################################################

;put 10 to the proper exponent (value in EDX) on the FPU

XexpY:
      push  edx
      fild  dword ptr[esp]    ;load the exponent
      fldl2t                  ;load log2(10)
      fmul                    ;->log2(10)*exponent
      pop   edx

;at this point, only the log base 2 of the 10^exponent is on the FPU
;the FPU can compute the antilog only with the mantissa
;the characteristic of the logarithm must thus be removed

      fld   st(0)             ;copy the logarithm
      frndint                 ;keep only the characteristic
      fsub  st(1),st          ;keeps only the mantissa
      fxch                    ;get the mantissa on top

      f2xm1                   ;->2^(mantissa)-1
      fld1
      fadd                    ;add 1 back

;the number must now be readjusted for the characteristic of the logarithm

      fscale                  ;scale it with the characteristic

;the characteristic is still on the FPU and must be removed

      fstp  st(1)             ;clean-up the register
      ret

;########################################################################

;shifts the packed BCD string of the integers to the integer position
;EDI points to the BCD string
;ECX = count of positions for shifting the BCD string

load_integer:
      push  esi
      .if   cl == 18
```

```
            fldz
      .else
            mov   esi,edi
            sub   ecx,18
            neg   ecx
            shr   ecx,1
            push  edi
            .if   !CARRY?     ;even number of integer digits
                  mov   edx,9
                  sub   edx,ecx
                  add   esi,edx
                  rep   movsb
            .else             ;odd number of integer digits
                  mov   edx,8
                  sub   edx,ecx
                  add   esi,edx
                  xor   eax,eax
                  lodsb
                  rol   ax,4
                  test  ecx,ecx
                  .if   !ZERO?
                     @@:
                        rol   ah,4
                        lodsb
                        rol   ax,4
                        stosb
                        dec   ecx
                        jnz   @B
                  .endif
                  mov   [edi],ah
                  inc   edi
            .endif
            mov   ecx,edx
            xor   eax,eax
            rep   stosb
            pop   edi
            fbld  tbyte ptr[edi]
      .endif
      pop   esi
      ret

;###########################################################################

;converts the decimal portion in the packed BCD string to binary
;EDI points to the BCD string

load_decimal:
      test  bh,2
      jnz   @F
      ret
   @@:
      .if   cl == 18
            fldz
      .else
            fbld  tbyte ptr[edi]
            mov   edx,-18
            call  XexpY
            fmul
      .endif
      fadd
      ret

;###########################################################################

end



; ###########################################################################
;
```

```
;                               FpuChs
;
;#########################################################################

  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameter and allow additional data types for storage.
  ;
  ;                               -(Src) -> Dest
  ;
  ; This FpuChs function changes the sign of a REAL number (Src) and returns
  ; the result at the specified destination (the FPU itself or a memory
  ; variable), unless an invalid operation is reported by the FPU or the
  ; definition of the parameters (with uID) is invalid.
  ;
  ; The source can only be a REAL number from the FPU itself or either a
  ; REAL4, REAL8 or REAL10 from memory. (The sign of integers can be changed
  ; easily with a CPU instruction.)
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; #########################################################################

FpuChs proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;-------------------------------
;check if top register is empty
;-------------------------------
```

```
      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

   @@:
      mov   eax,lpSrc
      fld   tbyte ptr[eax]

dest0:
      fchs

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return error

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
```

```
      test  uID,DEST_MEM8
      .if   !ZERO?           ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was Src taken from FPU
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            ffree st(7)       ;free it if not already empty
            fld   tempst      ;return the result on the FPU
      .endif

      or    al,1              ;to insure EAX!=0
      ret

FpuChs endp

; ########################################################################

end



; ########################################################################
;
;                             FpuComp
;
;########################################################################

  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters.
  ;
  ; This FpuComp function compares one number (Src1) to another (Src2)
  ; with the FPU and returns the result in EAX as coded bits:
  ;         EAX = 0     comparison impossible
  ;         bit 0       1 = Src1 = Src2
  ;         bit 1       1 = Src1 > Src2
  ;         bit 2       1 = Src1 < Src2
  ;
  ; Either of the two sources can be:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory, or
  ; one of the FPU constants.
  ;
  ; None of the sources are checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return the result. All other CPU registers are
  ; preserved. All FPU registers are also preserved.
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive
```

```
    include Fpu.inc

    .code

; ########################################################################

FpuComp proc public lpSrc1:DWORD, lpSrc2:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU or SRC2_FPU      ;is data taken from FPU?
      jz    @F                ;continue if not

;------------------------------
;check if top register is empty
;------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   @F                ;not empty if Carry flag not set
      jpe   @F                ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

   @@:
      fsave content

;----------------------------------------
;check source for Src1 and load it to FPU
;----------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src1 is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   src2        ;check next parameter for Src2
      .endif

      mov   eax,lpSrc1
      test  uID,SRC1_CONST
      jnz   constant
      test  uID,SRC1_REAL
      .if !ZERO?              ;Src1 is an 80-bit REAL10 in memory?
            fld   tbyte ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_REAL8
      .if !ZERO?              ;Src1 is a 64-bit REAL10 in memory?
            fld   qword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_REAL4
      .if !ZERO?              ;Src1 is a 32-bit REAL10 in memory?
            fld   dword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src1 is a 32-bit integer in memory?
            fild  dword ptr [eax]
```

```
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src1 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif

      test  uID,SRC1_DIMM
      .if   !ZERO?            ;Src1 is an immediate 32-bit integer?
            fild  lpSrc1
            jmp   src2        ;check next parameter for Src2
      .endif

      ;otherwise no valid ID for Src1

srcerr:
      frstor content
srcerr1:
      xor   eax,eax           ;error code
      ret

constant:
      cmp   eax,FPU_PI
      jnz   @F
      fldpi
      jmp   src2
@@:
      cmp   eax,FPU_NAPIER
      jnz   srcerr            ;no correct CONST for Src1
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

;---------------------------------------
;check source for Src2 and load it to FPU
;---------------------------------------

src2:
      test  uID,SRC2_FPU
      .if   !ZERO?            ;Src2 is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28] ;retrieve it from the stored data
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc2
      test  uID,SRC2_CONST
      jnz   constant2
      test  uID,SRC2_REAL
      .if   !ZERO?            ;Src2 is an 80-bit REAL10 in memory?
            fld   tbyte ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_REAL8
      .if   !ZERO?            ;Src2 is a 64-bit REAL10 in memory?
            fld   qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_REAL4
      .if   !ZERO?            ;Src2 is a 32-bit REAL10 in memory?
            fld   dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC2_DMEM
```

```
      .if   !ZERO?            ;Src2 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_QMEM
      .if   !ZERO?            ;Src2 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC2_DIMM
      .if   !ZERO?            ;Src2 is an immediate 32-bit integer?
            fild  lpSrc2
            jmp   dest0       ;go complete process
      .endif
      jmp   srcerr            ;no correct flag for Src2

constant2:
      cmp   eax,FPU_PI
      jnz   @F
      fldpi                   ;load pi (3.14159...) on FPU
      jmp   dest0             ;go complete process
@@:
      cmp   eax,FPU_NAPIER
      jnz   srcerr            ;no correct CONST for Src2
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

dest0:
      fxch
      fcom
      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return result
      sahf                    ;transfer to the CPU flags
      jpe   srcerr            ;error if non comparable
      ja    greater
      jc    @F
      mov   eax,CMP_EQU       ;Src2 = Src1
      jmp   finish

   @@:
      mov   eax,CMP_LOWER     ;Src1 < Src2
      jmp   finish

greater:
      mov   eax,CMP_GREATER   ;Src1 > Src2

finish:
      frstor content

      ret

FpuComp endp

; ########################################################################


end



; ########################################################################
;
```

```
;                               FpuCos
;
;##############################################################################

  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified January 2004 to prevent stack faults and to adjust
  ; angles outside the acceptable range if necessary.
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;
  ;                               cos(Src) -> Dest
  ;
  ; This FpuCos function computes the cosine of an angle in degrees or radians
  ; (Src) with the FPU and returns the result as a REAL number at the
  ; specified destination (the FPU itself or a memory location), unless
  ; an invalid operation is reported by the FPU or the definition of the
  ; parameters (with uID) is invalid.
  ;
  ; The source can be either:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ##############################################################################

FpuCos proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue
```

```
;------------------------------
;check if top register is empty
;------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src1 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src1 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DIMM     ;is Src an immediate 32-bit integer?
      jnz   @F                ;last valid ID for Src

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

   @@:
      fild  lpSrc

dest0:
      test  uID,ANG_RAD
      jnz   @F                ;jump if angle already in radians
```

```
      fldpi                   ;load pi (3.14159...) on FPU
      fmul
      pushd 180
      fidiv word ptr[esp]     ;value now in radians
      fwait
      pop   eax               ;clean the stack
   @@:
      fldpi
      fadd  st,st             ;->2pi
      fxch
   @@:
      fprem                   ;reduce the angle
      fcos
      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return error
      sahf                    ;transfer to the CPU flags
      jpe   @B                ;reduce angle again if necessary
      fstp  st(1)             ;get rid of the 2pi

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was Src taken from FPU
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            ffree st(7)       ;free it if not already empty
            fld   tempst      ;return the result on the FPU
      .endif

      or    al,1              ;to insure EAX!=0
      ret

FpuCos endp

; ########################################################################

end



; ########################################################################
;
;                             FpuCosh
;
```

```
;##########################################################################

  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;
  ;     cosh(Src) = [e^(Src) + e^(-Src)]/2 -> Dest    (see FpuEexpX for e^x)
  ;
  ; This FpuCosh function computes the hyperbolic cosine of a number (Src)
  ; with the FPU and returns the result as a REAL number at the specified
  ; destination (the FPU itself or a memory location), unless an invalid
  ; operation is reported by the FPU or the definition of the parameters
  ; (with uID) is invalid.
  ;
  ; The source can be either:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ##########################################################################

FpuCosh proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;------------------------------
;check if top register is empty
;------------------------------
```

```
      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;--------------------------------------
;check source for Src and load it to FPU
;--------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src1 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src1 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DIMM     ;is Src an immediate 32-bit integer?
      jnz   @F                ;otherwise no correct flag for Src

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

   @@:
      fild  lpSrc

dest0:
      fldl2e
      fmul                    ;log2(e)*Src
      fld   st(0)
      frndint
      fxch
      fsub  st,st(1)
```

```
      f2xm1
      fld1
      fadd
      fscale                  ;-> antilog[log2(e)*Src] = e^(Src)
      fstp  st(1)             ;get rid of scaling factor

      fld   st                ;copy it to get the reciprocal
      fld1
      fdivrp st(1),st         ;1/e^(Src) = e^(-Src)
      fadd                    ;e^(Src) + e^(-Src)
      fld1
      fchs                    ;-1
      fxch
      fscale                  ;-> [e^(Src) + e^(-Src)]/2 = cosh(Src)
      fstp  st(1)             ;get rid of scaling factor

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return error

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was Src taken from FPU
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU
      jz    @F                ;the result has been stored in memory
                              ;none of the FPU data was modified

      ffree st(7)             ;free it if not already empty
      fld   tempst            ;load the result on the FPU
   @@:
      or    al,1              ;to insure EAX!=0
      ret

FpuCosh endp

; ########################################################################

end



; ########################################################################
;
;                             FpuDiv
```

```
;
;##########################################################################

  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;
  ;                            Src1 / Src2 -> Dest
  ;
  ; This FpuDiv function divides the Src1 number by the Src2 number
  ; with the FPU and returns the result as an 80-bit REAL number at the
  ; specified destination (the FPU itself or a memory location), unless an
  ; invalid operation is reported by the FPU or the definition of the
  ; parameters (with uID) is invalid.
  ;
  ; Either of the two sources can be:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory, or
  ; one of the FPU constants.
  ;
  ; None of the sources are checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ##########################################################################

FpuDiv proc public lpSrc1:DWORD, lpSrc2:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU or SRC2_FPU      ;is data taken from FPU?
      jz    continue

;------------------------------
```

```
;check if top register is empty
;------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src1 and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src1 is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   src2        ;check next parameter for Src2
      .endif

      mov   eax,lpSrc1
      test  uID,SRC1_CONST
      jnz   constant
      test  uID,SRC1_REAL
      .if !ZERO?              ;Src1 is an 80-bit REAL10 in memory?
            fld   tbyte ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_REAL8
      .if !ZERO?              ;Src1 is a 64-bit REAL10 in memory?
            fld   qword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_REAL4
      .if !ZERO?              ;Src1 is a 32-bit REAL10 in memory?
            fld   dword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src1 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src1 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif

      test  uID,SRC1_DIMM
      .if   !ZERO?            ;Src1 is an immediate 32-bit integer?
            fild  lpSrc1
            jmp   src2        ;check next parameter for Src2
      .endif

      ;otherwise no valid ID for Src1

srcerr:
      frstor content
srcerr1:
      xor   eax,eax           ;error code
      ret

constant:
```

```
      cmp   eax,FPU_PI
      jnz   @F
      fldpi
      jmp   src2
   @@:
      cmp   eax,FPU_NAPIER
      jnz   srcerr            ;no correct CONST for Src1
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

;---------------------------------------
;check source for Src2 and load it to FPU
;---------------------------------------

src2:
      test  uID,SRC2_FPU
      .if   !ZERO?            ;Src2 is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28] ;retrieve it from the stored data
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc2
      test  uID,SRC2_CONST
      jnz   constant2
      test  uID,SRC2_REAL
      .if   !ZERO?            ;Src2 is an 80-bit REAL10 in memory?
            fld   tbyte ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_REAL8
      .if   !ZERO?            ;Src2 is a 64-bit REAL10 in memory?
            fld   qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_REAL4
      .if   !ZERO?            ;Src2 is a 32-bit REAL10 in memory?
            fld   dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC2_DMEM
      .if   !ZERO?            ;Src2 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_QMEM
      .if   !ZERO?            ;Src2 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC2_DIMM
      .if   !ZERO?            ;Src2 is an immediate 32-bit integer?
            fild  lpSrc2
            jmp   dest0       ;go complete process
      .endif
      jmp   srcerr            ;no correct flag for Src2

constant2:
      cmp   eax,FPU_PI
      jnz   @F
      fldpi                   ;load pi (3.14159...) on FPU
      jmp   dest0             ;go complete process
   @@:
```

```
      cmp   eax,FPU_NAPIER
      jnz   srcerr            ;no correct CONST for Src2
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

dest0:
      fdiv

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return error

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU or SRC2_FPU      ;was any data taken from FPU?
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU
      jz    @F                ;the result has been stored in memory
                              ;none of the FPU data was modified

      ffree st(7)             ;free it if not already empty
      fld   tempst            ;load the result on the FPU
   @@:
      or    al,1              ;to insure EAX!=0
      ret

FpuDiv endp

; ########################################################################

end



; ########################################################################
;
;                             FpuEexpX
;
;########################################################################
```

```
  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;
  ;              e^(Src) = antilog2[ log2(e) * Src ] -> Dest
  ;
  ; This FpuEexpX function computes the Naperian antilogarithm of a number.
  ; It raises the Naperian constant to the power of the Src number with
  ; the FPU and returns the result as a REAL number at the specified
  ; destination (the FPU itself or a memory location), unless an invalid
  ; operation is reported by the FPU or the definition of the parameters
  ; (with uID) is invalid.
  ;
  ; The exponent can be either:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory, or
  ; one of the FPU constants.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuEexpX proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;------------------------------
;check if top register is empty
;------------------------------
```

```
      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;--------------------------------------
;check source for Src and load it to FPU
;--------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_CONST
      jnz   constant
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DIMM
      .if   !ZERO?            ;Src is an immediate 32-bit integer?
            fild  lpSrc
            jmp   dest0       ;go complete process
      .endif

      ;otherwise no correct flag for Src

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

constant:
      cmp   eax,FPU_PI
      jz    @F
```

```
      fldpi                   ;load pi (3.14159...) on FPU
      jmp   dest0             ;go complete process
   @@:
      cmp   eax,FPU_NAPIER
      jz    srcerr            ;no correct CONST for Src
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

dest0:
      fldl2e                  ;->log2(e)
      fmul                    ;->log2(e)*Src

;the FPU can compute the antilog only with the mantissa
;the characteristic of the logarithm must thus be removed

      fld   st(0)             ;copy the logarithm
      frndint                 ;keep only the characteristic
      fsub  st(1),st          ;keeps only the mantissa
      fxch                    ;get the mantissa on top

      f2xm1                   ;->2^(mantissa)-1
      fld1
      fadd                    ;add 1 back

;the number must now be readjusted for the characteristic of the logarithm

      fscale                  ;scale it with the characteristic

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return if error

;the characteristic is still on the FPU and must be removed

      fstp  st(1)             ;get rid of the characteristic

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was Src taken from FPU
      jz    @F
      fstp  st                ;remove source

   @@:
```

```
      test  uID,DEST_FPU
      jz    @F                ;the result has been stored in memory
                              ;none of the FPU data was modified

      ffree st(7)             ;free it if not already empty
      fld   tempst            ;load the result on the FPU
   @@:
      or    al,1              ;to insure EAX!=0
      ret

FpuEexpX endp

; ########################################################################

end



; ########################################################################
;
;                             FpuExam
;
;########################################################################

  ; ----------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters. Also corrected a potential bug.
  ;
  ; This FpuExam function examines a REAL number (Src) for its validity,
  ; its sign, a value of zero, an absolute value less than 1, and a value
  ; of infinity.
  ; The result is returned in EAX as coded bits:
  ;         EAX = 0     invalid number
  ;         bit 0       1 = valid number
  ;         bit 1       1 = number is equal to zero
  ;         bit 2       1 = number is negative
  ;         bit 3       1 = number less than 1 but not zero
  ;         bit 4       1 = number is infinity
  ; If the source was on the FPU, it will be preserved if no error is
  ; reported.
  ;
  ; The source can only be a REAL number from the FPU itself or either a
  ; REAL4, REAL8 or REAL10 from memory.
  ;
  ; Only EAX is used to return the result. All other CPU registers are
  ; preserved. All FPU registers are also preserved.
  ;
  ; ----------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuExam proc public uses edx lpSrc:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
LOCAL content[108] :BYTE

      test  uID,SRC1_FPU      ;is data taken from FPU?
      jz    @F               ;continue if not

;------------------------------
;check if top register is empty
;------------------------------

      fxam                   ;examine its content
      fstsw ax               ;store results in AX
      fwait                  ;for precaution
      sahf                   ;transfer result bits to CPU flag
      jnc   @F               ;not empty if Carry flag not set
      jpe   @F               ;not empty if Parity flag set
      jz    srcerr1          ;empty if Zero flag set

   @@:
      fsave content

;---------------------------------------
;check source for Src and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?           ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0      ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?           ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0      ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?           ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0      ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?           ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0      ;go complete process
      .endif

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

dest0:
      ftst                   ;test number
      fstsw ax               ;retrieve exception flags from FPU
      fwait
      shr   al,1             ;invalid operation?
      jc    srcerr

      xor   edx,edx
      sahf                   ;transfer flags to CPU flag register
      jnz   @F               ;if not 0 value or NAN
      jc    examine          ;go check for infinity or NAN value
      or    edx,XAM_ZERO or XAM_SMALL
      jmp   finish           ;no need for checking for sign or size if 0

   @@:
```

```
      jnc   @F                ;number is not negative
      or    edx,XAM_NEG

;check for size smaller than 1 by comparing the absolute value to 1

   @@:
      fabs                    ;make sure it is positive
      fld1                    ;for comparing to 1
      fcompp                  ;compare 1 to absolute value and pop both
      fstsw ax                ;retrieve result flags from FPU
      fwait
      sahf                    ;transfer flags to CPU flag register
      jc    finish            ;src>1
      jz    finish            ;src=1
      or    edx,XAM_SMALL     ;value less than 1

finish:
      frstor content
      mov   eax,edx
      or    al,1              ;to indicate source was a valid number
      ret

examine:                      ;strictly for infinity value
      fxam
      fstsw ax                ;retrieve result of fxam
      fwait
      sahf                    ;transfer flags to CPU flag register
      jpo   srcerr            ;must be NAN
      or    edx,XAM_INFINIT
      jmp   finish


FpuExam endp

; ########################################################################

end



; ########################################################################
;
;                             FpuFLtoA
;
;########################################################################

  ; ----------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; and modified April 2003. A minor flaw was corrected in July 2003.
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters.
  ; Revised May 2011 to correct a minor flaw causing the function to return
  ; an error when the input value was smaller than 3.36e-4917.
  ;
  ; This FpuFLtoA function converts a REAL number (Src) to its decimal
  ; representation as a zero terminated alphanumeric string which
  ; is returned at the specified memory destination unless an invalid
  ; operation is reported by the FPU or the definition of the parameters
  ; (with uID) is invalid.
  ;
  ; The format of the string can be specified as regular (default) or
  ; scientific notation. The number of decimal places returned must also be
  ; specified but the total number of significant digits must not exceed 16.
  ; When the regular format is specified, the integer portion can also be
  ; padded with preceding spaces to position the decimal point at a
  ; specified location from the start of the string.
  ;
  ; The source can be a REAL number from the FPU itself or either a
  ; REAL4, REAL8 or REAL10 from memory.
```

```
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; This procedure is based on using an FPU instruction to convert the
  ; REAL number into a specific packed decimal format. After unpacking,
  ; the decimal point is positioned as required.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved. All FPU registers are preserved.
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

      .code

; ########################################################################

FpuFLtoA proc public lpSrc:DWORD, lpDecimal:DWORD, lpDest:DWORD, uID:DWORD

LOCAL tempdw       :DWORD
LOCAL esize        :DWORD
LOCAL Padding      :DWORD
LOCAL Decimal      :DWORD
LOCAL extra10x     :DWORD
LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE
LOCAL bcdstr       :TBYTE
LOCAL oldcw        :WORD
LOCAL truncw       :WORD
LOCAL unpacked[20] :BYTE

;get the specified number of decimals for result
;and make corrections if necessary

      mov   eax,lpDecimal
      test  uID,SRC2_DMEM
      jz    @F
      mov   eax,[eax]         ;get the decimals from memory
   @@:
      push  eax
      movzx eax,al            ;low byte - number of decimal digits
      cmp   eax,15
      jbe   @F
      mov   eax,15            ;a maximum of 15 decimals is allowed
   @@:
      mov   Decimal,eax
      pop   eax
      movzx eax,ah            ;2nd byte - number of char before decimal point
      cmp   eax,17
      jbe   @F
      mov   eax,17            ;a maximum of 17 characters is allowed
   @@:
      mov   Padding,eax

      test  uID,SRC1_FPU      ;is data taken from FPU?
      jz    @F                ;continue if not

;------------------------------
;check if top register is empty
;------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
```

```
      sahf                    ;transfer result bits to CPU flag
      jnc   @F                ;not empty if Carry flag not set
      jpe   @F                ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

   @@:
      fsave content

;---------------------------------------
;check source for Src and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

srcerr:
      frstor content
srcerr1:
      push  edi
      mov   edi,lpDest
      mov   eax,"ORRE"
      stosd
      mov   ax,"R"
      stosw
      pop   edi
      xor   eax,eax
      ret

dest0:

;-----------------------------------------
;first examine the value on FPU for validity
;-----------------------------------------

      fxam                    ;examine value on FPU
      fstsw ax                ;get result
      fwait
      sahf                    ;transfer to CPU flags
      jz    maybezero
      jpo   srcerr            ;C3=0 and C2=0 would be NAN or unsupported
      jnc   getnumsize        ;continue if normal finite number

;--------------------------------
;value to be converted = INFINITY
;--------------------------------

      push  ecx
      push  esi
      push  edi
```

```
      mov   edi,lpDest
      mov   al,"+"
      test  ah,2              ;C1 field = sign
      jz    @F
      mov   al,"-"
   @@:
      stosb
      mov   eax,"IFNI"
      stosd
      mov   eax,"YTIN"
      stosd
      jmp   finish

;-------------------------
;value to be converted = 0
;-------------------------

maybezero:
      jpe   getnumsize        ;would be denormalized number
      fstp  st(0)             ;flush that 0 value off the FPU
      push  ecx
      push  esi
      push  edi
      mov   edi,lpDest
      test  uID,STR_SCI       ;scientific notation?
      jnz   @F                ;no padding
      mov   ecx,Padding
      sub   ecx,2
      jle   @F                ;no padding specified or necessary
      mov   al," "
      rep   stosb
   @@:
      mov   ax,3020h          ;" 0" szstring
      stosw                   ;write it
      jmp   finish

;---------------------------
; get the size of the number
;---------------------------

getnumsize:
      fldlg2                  ;log10(2)
      fld   st(1)             ;copy Src
      fabs                    ;insures a positive value
      fyl2x                   ;->[log2(Src)]*[log10(2)] = log10(Src)

      fstcw oldcw             ;get current control word
      fwait
      mov   ax,oldcw
      or    ax,0c00h          ;code it for truncating
      mov   truncw,ax
      fldcw truncw            ;insure rounding code of FPU to truncating

      fist  esize             ;store characteristic of logarithm
      fldcw oldcw             ;load back the former control word

      ftst                    ;test logarithm for its sign
      fstsw ax                ;get result
      fwait
      sahf                    ;transfer to CPU flags
      sbb   esize,0           ;decrement esize if log is negative
      fstp  st(0)             ;get rid of the logarithm

;------------------------------------------------------------------------
; get the power of 10 required to generate an integer with the specified
; number of significant digits
;------------------------------------------------------------------------

      mov   eax,uID
      and   eax,STR_SCI
```

```
      mov   extra10x,0
      .if   eax == 0          ;regular decimal notation
            mov   eax,esize
            or    eax,eax     ;check if number is < 1
            js    @F
            .if   eax > 15    ;if number is >= 10^16
                  or    uID,STR_SCI ;switch to scientific notation
                  mov   Decimal,15  ;insures 15 decimal places in result
                  jmp   scific
            .endif
            add   eax,Decimal
            .if   eax > 15    ;if integer + decimal digits > 16
                  sub   eax,15
                  sub   Decimal,eax ;reduce decimal digits as required
            .endif
         @@:
            push  Decimal
            pop   tempdw
      .else                   ;scientific notation
scific:
            mov   eax,Decimal
            sub   eax,esize
;-------------------------------------------------------------------------
; added to v2.34
            cmp   eax,4931  ;check if power of 10 would exceed REAL10 limit
            jle   @F
            mov   ecx,4931
            sub   eax,ecx
            xchg  eax,ecx
            mov   extra10x,ecx
          @@:
;-------------------------------------------------------------------------
            mov   tempdw,eax
      .endif

;-----------------------------------------------------------------------------------------
; multiply the number by the power of 10 to generate required integer and store it as BCD
;-----------------------------------------------------------------------------------------

.if   tempdw != 0
      fild  tempdw
      fldl2t
      fmul                    ;->log2(10)*exponent
      fld   st
      frndint                 ;get the characteristic of the log
      fxch
      fsub st,st(1)           ;get only the fractional part but keep the characteristic
      f2xm1                   ;->2^(fractional part)-1
      fld1
      fadd                    ;add 1 back
      fscale                  ;re-adjust the exponent part of the REAL number
      fstp  st(1)             ;get rid of the characteristic of the log
      fmul                    ;->16-digit integer
;-------------------------------------------------------------------------
; added to v2.34
      .if   extra10x != 0
; convert the extra 10^x to binary
            fild  extra10x
            fldl2t            ;->log2(10)
            fmul              ;->log2(10)*Src
            fld   st          ;copy the logarithm
            frndint           ;keep only the characteristic
            fxch              ;get the mantissa on top
            fsub  st,st(1)    ;keeps only the mantissa
            f2xm1             ;->2^(mantissa)-1
            fld1
            fadd              ;add 1 back
            fscale            ;scale it with the characteristic
            fstp  st(1)       ;overwrite the characteristic
```

```
            fmul              ;will provide required digits for display
      .endif
;-------------------------------------------------------------------------------
.endif

      fbstp bcdstr            ;->TBYTE containing the packed digits
      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   eax,1             ;test for invalid operation
      jc    srcerr            ;clean-up and return error

;-------------------------------------------------------------------------------
; unpack BCD, the 10 bytes returned by the FPU being in the little-endian style
;-------------------------------------------------------------------------------

      push  ecx
      push  esi
      push  edi
      lea   esi,bcdstr+9      ;go to the most significant byte (sign byte)
      lea   edi,unpacked
      mov   eax,3020h
      mov   cl,byte ptr[esi]  ;sign byte
      .if   cl == 80h
            mov   al,"-"      ;insert sign if negative number
      .endif
      stosw
      mov   ecx,9
   @@:
      dec   esi
      movzx eax,byte ptr[esi]
      ror   ax,4
      ror   ah,4
      add   ax,3030h
      stosw
      dec   ecx
      jnz   @B

      mov   edi,lpDest
      lea   esi,unpacked
      test  uID,STR_SCI       ;scientific notation?
      jnz   scientific

;************************
; REGULAR STRING NOTATION
;************************

;------------------------------
; check if padding is specified
;------------------------------

      mov   ecx,Padding
      or    ecx,ecx           ;check if padding is specified
      jz    nopadding

      mov   edx,2             ;at least 1 integer + sign
      mov   eax,esize
      or    eax,eax
      js    @F                ;only 1 integer digit if size is < 1
      add   edx,eax           ;->number of integer digits
   @@:
      sub   ecx,edx
      jle   nopadding
      mov   al," "
      rep   stosb

nopadding:
      pushfd                  ;save padding flags
      movsb                   ;insert sign
      mov   ecx,1             ;at least 1 integer digit
      mov   eax,esize
```

```
      or    eax,eax           ;is size negative (i.e. number smaller than 1)
      js    @F
      add   ecx,eax
   @@:
      mov   eax,Decimal
      add   eax,ecx           ;->total number of digits to be displayed
      sub   eax,19
      sub   esi,eax           ;address of 1st digit to be displayed
      pop   eax               ;retrieve padding flags in EAX
      .if   byte ptr[esi-1] == "1"
            dec   esi
            inc   ecx
            push  eax         ;transfer padding flags through stack
            popfd             ;retrieve padding flags
            jle   @F          ;no padding was necessary
            dec   edi         ;adjust for one less padding byte
      .endif
   @@:
      rep   movsb             ;copy required integer digits
      mov   ecx,Decimal
      or    ecx,ecx
      jz    @F
      mov   al,"."
      stosb
      rep   movsb             ;copy required decimal digits
   @@:
      jmp   finish

;********************
; SCIENTIFIC NOTATION
;********************

scientific:
      movsb                   ;insert sign
      mov   ecx,Decimal
      mov   eax,18
      sub   eax,ecx
      add   esi,eax
      cmp   byte ptr[esi-1],"1"
      pushfd                  ;save flags for extra "1"
      jnz   @F
      dec   esi
   @@:
      movsb                   ;copy the integer
      mov   al,"."
      stosb
      rep   movsb             ;copy the decimal digits

      mov   al,"E"
      stosb
      mov   al,"+"
      mov   ecx,esize
      popfd                   ;retrieve flags for extra "1"
      jnz   @F                ;no extra "1"
      inc   ecx               ;adjust exponent
   @@:
      or    ecx,ecx
      jns   @F
      mov   al,"-"
      neg   ecx               ;make number positive
   @@:
      stosb                   ;insert proper sign

;Note: the absolute value of the size could not exceed 4931

      mov   eax,ecx
      mov   cl,100
      div   cl                ;->thousands & hundreds in AL, tens & units in AH
      push  eax
      and   eax,0ffh          ;keep only the thousands & hundreds
```

```
      mov   cl,10
      div   cl                ;->thousands in AL, hundreds in AH
      add   ax,3030h          ;convert to characters
      stosw                   ;insert them
      pop   eax
      shr   eax,8             ;get the tens & units in AL
      div   cl                ;tens in AL, units in AH
      add   ax,3030h          ;convert to characters
      stosw                   ;insert them

finish:
      xor   eax,eax
      stosb                   ;string terminating 0
      pop   edi
      pop   esi
      pop   ecx

      frstor content

      or    al,1              ;to insure EAX!=0
      ret

FpuFLtoA endp

; ########################################################################

end



; ########################################################################
;
;                             FpuLnx
;
;########################################################################

  ; ----------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;
  ;                ln(Src) = log2(Src) * ln(2) -> Dest
  ;
  ; This FpuLnx function computes the natural logarithm of a number (Src)
  ; with the FPU and returns the result as a REAL number at the
  ; specified destination (the FPU itself or a memory location), unless
  ; an invalid operation is reported by the FPU or the definition of the
  ; parameters (with uID) is invalid.
  ;
  ; The source can be either:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory, or
  ; one of the FPU constants.
  ; Negative or zero values of Src will return an error.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
```

```
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuLnx proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;------------------------------
;check if top register is empty
;------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;--------------------------------------
;check source for Src and load it to FPU
;--------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_CONST
      jnz   constant
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
```

```
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DIMM
      .if   !ZERO?            ;Src is an immediate 32-bit integer?
            fild  lpSrc
            jmp   dest0       ;go complete process
      .endif

      ;otherwise no correct flag for Src

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

constant:
      cmp   eax,FPU_PI
      jz    @F
      fldpi                   ;load pi (3.14159...) on FPU
      jmp   dest0             ;go complete process
@@:
      cmp   eax,FPU_NAPIER
      jz    srcerr            ;no correct CONST for Src
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

dest0:
      fldln2
      fxch
      fyl2x                   ;->[log2(Src)]*ln(2) = ln(Src)

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return error

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
```

```
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was Src taken from FPU
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU
      jz    @F                ;the result has been stored in memory
                              ;none of the FPU data was modified

      ffree st(7)             ;free it if not already empty
      fld   tempst            ;load the result on the FPU
   @@:
      or    al,1              ;to insure EAX!=0
      ret

FpuLnx endp

; ########################################################################

end



; ########################################################################
;
;                             FpuLogx
;
;########################################################################

  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;
  ;               log(Src) = log2(Src) * log10(2) -> Dest
  ;
  ; This FpuLog function computes the logarithm base 10 of a number (Src)
  ; with the FPU and returns the result as a REAL number at the
  ; specified destination (the FPU itself or a memory location), unless
  ; an invalid operation is reported by the FPU or the definition of the
  ; parameters (with uID) is invalid.
  ;
  ; The source can be either:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory, or
  ; one of the FPU constants.
  ; Negative or zero values of Src will return an error.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
```

```
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuLogx proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;-------------------------------
;check if top register is empty
;-------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_CONST
      jnz   constant
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
```

```
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DIMM
      .if   !ZERO?            ;Src is an immediate 32-bit integer?
            fild  lpSrc
            jmp   dest0       ;go complete process
      .endif

      ;otherwise no correct flag for Src

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

constant:
      cmp   eax,FPU_PI
      jz    @F
      fldpi                   ;load pi (3.14159...) on FPU
      jmp   dest0             ;go complete process
   @@:
      cmp   eax,FPU_NAPIER
      jz    srcerr            ;no correct CONST for Src
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

dest0:
      fldlg2
      fxch
      fyl2x                   ;->[log2(Src)]*log10(2) = log(Src) base 10

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return error

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
```

```
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was Src taken from FPU
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU
      jz    @F                ;the result has been stored in memory
                              ;none of the FPU data was modified

      ffree st(7)             ;free it if not already empty
      fld   tempst            ;load the result on the FPU
@@:
      or    al,1              ;to insure EAX!=0
      ret

FpuLogx endp

; ########################################################################

end



; ########################################################################
;
;                             FpuMod
;
;########################################################################

  ; ------------------------------------------------------------------------
  ; This procedure was originally provided by E^cube, January 2010.
  ; Modified January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;
  ;                             Src1 mod Src2 -> Dest
  ;
  ; This FpuMod computes the modulo of the Src1 number by the Src2 number
  ; with the FPU and returns the result as a REAL number at the specified
  ; destination (the FPU itself or a memory location), unless an invalid
  ; operation is reported by the FPU or the definition of the parameters
  ; (with uID) is invalid.
  ; In essence, Result = Src1 - Q*Src2, where Q is an integer.
  ;
  ; Either of the two sources can be:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory, or
  ; one of the FPU constants.
  ;
  ; None of the sources are checked for validity. This is the programmer's
  ; responsibility.
  ;
```

```
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

FpuMod proc public lpSrc1:DWORD, lpSrc2:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU or SRC2_FPU      ;is data taken from FPU?
      jz    continue

;--------------------------------
;check if top register is empty
;--------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src1 is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   src2        ;check next parameter for Src2
      .endif

      mov   eax,lpSrc1
      test  uID,SRC1_CONST
      jnz   constant
      test  uID,SRC1_REAL
      .if !ZERO?              ;Src1 is an 80-bit REAL10 in memory?
            fld   tbyte ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_REAL8
```

```
      .if !ZERO?              ;Src1 is a 64-bit REAL10 in memory?
            fld   qword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_REAL4
      .if !ZERO?              ;Src1 is a 32-bit REAL10 in memory?
            fld   dword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src1 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src1 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif

      test  uID,SRC1_DIMM
      .if   !ZERO?            ;Src1 is an immediate 32-bit integer?
            fild  lpSrc1
            jmp   src2        ;check next parameter for Src2
      .endif

      ;otherwise no valid ID for Src1

srcerr:
      frstor content
srcerr1:
      xor   eax,eax           ;error code
      ret

constant:
      test  eax,FPU_PI
      jz    @F
      fldpi
      jmp   src2
   @@:
      test  eax,FPU_NAPIER
      jz    srcerr            ;no correct CONST flag for Src1
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

;---------------------------------------
;check source for Src2 and load it to FPU
;---------------------------------------

src2:
      test  uID,SRC2_FPU      ;is Src2 taken from FPU?
      .if   !ZERO?            ;Src2 is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28] ;retrieve it from the stored data
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc2
      test  uID,SRC2_CONST
      jnz   constant2
      test  uID,SRC2_REAL
      .if   !ZERO?            ;Src2 is an 80-bit REAL10 in memory?
            fld   tbyte ptr [eax]
            jmp   dest0       ;go complete process
```

```
      .endif
      test  uID,SRC2_REAL8
      .if   !ZERO?            ;Src2 is a 64-bit REAL10 in memory?
            fld   qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_REAL4
      .if   !ZERO?            ;Src2 is a 32-bit REAL10 in memory?
            fld   dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC2_DMEM
      .if   !ZERO?            ;Src2 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_QMEM
      .if   !ZERO?            ;Src2 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC2_DIMM
      .if   !ZERO?            ;Src2 is an immediate 32-bit integer?
            fild  lpSrc2
            jmp   dest0       ;go complete process
      .endif
      jmp   srcerr            ;no correct flag for Src2

constant2:
      cmp   eax,FPU_PI
      jnz   @F
      fldpi                   ;load pi (3.14159...) on FPU
      jmp   dest0             ;go complete process
   @@:
      cmp   eax,FPU_NAPIER
      jnz   srcerr            ;no correct CONST for Src2
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

dest0:
      fxch
   @@:
      fprem
      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return error
      sahf                    ;copy to the CPU flags
      jpe   @B             ;continue reducing if C2=PF=1 (reduction incomplete)

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
```

```
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU or SRC2_FPU      ;was any data taken from FPU?
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU
      jz    @F                ;the new value has been stored in memory
                              ;none of the FPU data was modified

      ffree st(7)             ;free it if not already empty
      fld   tempst            ;load the new value on the FPU
   @@:
      or    al,1              ;to insure EAX!=0
      ret

FpuMod endp

; ########################################################################

end



; ########################################################################
;
;                             FpuMul
;
;########################################################################

  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;
  ;                             Src1 * Src2 -> Dest
  ;
  ; This FpuMul function multiplies the numbers from two sources (Src1 and
  ; Src2) with the FPU and returns the result as a REAL number at the specified
  ; destination (the FPU itself or a memory location), unless an invalid
  ; operation is reported by the FPU or the definition of the parameters
  ; (with uID) is invalid.
  ;
  ; Either of the two sources can be:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory, or
  ; one of the FPU constants.
  ;
  ; None of the sources are checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
```

```
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuMul proc public lpSrc1:DWORD, lpSrc2:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU or SRC2_FPU      ;is data taken from FPU?
      jz    continue

;-------------------------------
;check if top register is empty
;-------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src1 and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src1 is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   src2        ;check next parameter for Src2
      .endif

      mov   eax,lpSrc1
      test  uID,SRC1_CONST
      jnz   constant
      test  uID,SRC1_REAL
      .if !ZERO?              ;Src1 is an 80-bit REAL10 in memory?
            fld   tbyte ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_REAL8
```

```
      .if !ZERO?              ;Src1 is a 64-bit REAL10 in memory?
            fld   qword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_REAL4
      .if !ZERO?              ;Src1 is a 32-bit REAL10 in memory?
            fld   dword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src1 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src1 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif

      test  uID,SRC1_DIMM
      .if   !ZERO?            ;Src1 is an immediate 32-bit integer?
            fild  lpSrc1
            jmp   src2        ;check next parameter for Src2
      .endif

      ;otherwise no valid ID for Src1

srcerr:
      frstor content
srcerr1:
      xor   eax,eax           ;error code
      ret

constant:
      cmp   eax,FPU_PI
      jnz   @F
      fldpi
      jmp   src2
   @@:
      cmp   eax,FPU_NAPIER
      jnz   srcerr            ;no correct CONST for Src1
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

;---------------------------------------
;check source for Src2 and load it to FPU
;---------------------------------------

src2:
      test  uID,SRC2_FPU
      .if   !ZERO?            ;Src2 is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28] ;retrieve it from the stored data
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc2
      test  uID,SRC2_CONST
      jnz   constant2
      test  uID,SRC2_REAL
      .if   !ZERO?            ;Src2 is an 80-bit REAL10 in memory?
            fld   tbyte ptr [eax]
            jmp   dest0       ;go complete process
```

```
      .endif
      test  uID,SRC2_REAL8
      .if   !ZERO?            ;Src2 is a 64-bit REAL10 in memory?
            fld   qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_REAL4
      .if   !ZERO?            ;Src2 is a 32-bit REAL10 in memory?
            fld   dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC2_DMEM
      .if   !ZERO?            ;Src2 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_QMEM
      .if   !ZERO?            ;Src2 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC2_DIMM
      .if   !ZERO?            ;Src2 is an immediate 32-bit integer?
            fild  lpSrc2
            jmp   dest0       ;go complete process
      .endif
      jmp   srcerr            ;no correct flag for Src2

constant2:
      cmp   eax,FPU_PI
      jnz   @F
      fldpi                   ;load pi (3.14159...) on FPU
      jmp   dest0             ;go complete process
   @@:
      cmp   eax,FPU_NAPIER
      jnz   srcerr            ;no correct CONST for Src2
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

dest0:
      fmul

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   eax,1             ;test for invalid operation
      jc    srcerr            ;clean-up and return error

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
```

```
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU or SRC2_FPU      ;was any data taken from FPU?
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU
      jz    @F                ;the result has been stored in memory
                              ;none of the FPU data was modified

      ffree st(7)             ;free it if not already empty
      fld   tempst            ;load the result on the FPU
   @@:
      or    al,1              ;to insure EAX!=0
      ret

FpuMul endp

; ########################################################################

end



; ########################################################################
;
;                              FpuRound
;
;########################################################################

  ; ------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified January 2004 to remove data section
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;    Added checking for overflow when storing the result as an integer.
  ;
  ; This FpuRound function rounds a REAL number (Src) to the nearest
  ; integer and returns the integer portion at the specified destination,
  ; unless an invalid operation is reported by the FPU or the definition
  ; of the parameters (with uID) is invalid.
  ;
  ; The source can only be a REAL number from the FPU itself or either a
  ; REAL4, REAL8 or REAL10 from memory.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------
```

```
    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuRound proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;------------------------------
;check if top register is empty
;------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
```

```
srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

dest0:
      push  eax               ;reserve space on CPU stack
      fstcw [esp]             ;get current control word
      fwait
      mov   ax,[esp]
      and   ax,0f3ffh         ;code it for rounding
      push  eax
      fldcw [esp]             ;change rounding code of FPU to round

      frndint                 ;round the number
      pop   eax               ;get rid of last push
      fldcw [esp]             ;load back the former control word

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      pop   eax               ;clean CPU stack
      jc    srcerr            ;clean-up and return error

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_IMEM
      .if   !ZERO?            ;store as DWORD at specified address
            fistp dword ptr[eax]
            jmp   integersave
      .endif
      test  uID,DEST_IMEM8
      .if   !ZERO?            ;store as QWORD at specified address
            fistp qword ptr[eax]
            jmp   integersave
      .endif
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was any data taken from FPU?
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU
      jz    @F                ;the result has been stored in memory
                              ;none of the FPU data was modified

      ffree st(7)             ;free it if not already empty
      fld   tempst            ;load the result on the FPU
   @@:
```

```
      or    al,1              ;to insure EAX!=0
      ret

integersave:
      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return error
      jmp   restore

FpuRound endp

; ########################################################################

end



; ########################################################################
;
;                             FpuSin
;
;########################################################################

  ; ----------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified January 2004 to prevent stack faults and to adjust
  ; angles outside the acceptable range if necessary.
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;
  ;                       sin(Src) -> Dest
  ;
  ; This FpuSin function computes the sine of an angle in degrees or radians
  ; (Src) with the FPU and returns the result as a REAL number at the
  ; specified destination (the FPU itself or a memory location), unless
  ; an invalid operation is reported by the FPU or the definition of the
  ; parameters (with uID) is invalid.
  ;
  ; The source can be either:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ----------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc
```

```
    .code

; ########################################################################

FpuSin proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;------------------------------
;check if top register is empty
;------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src1 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_QMEM
```

```
      .if !ZERO?              ;Src1 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DIMM     ;is Src an immediate 32-bit integer?
      jz    srcerr            ;no correct flag for Src
      fild  lpSrc
      jmp   dest0             ;go complete process

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

dest0:
      test  uID,ANG_RAD
      jnz   @F                ;jump if angle already in radians
      fldpi                   ;load pi (3.14159...) on FPU
      fmul
      pushd 180
      fidiv word ptr[esp]     ;value now in radians
      fwait
      pop   eax               ;clean the stack
   @@:
      fldpi
      fadd  st,st             ;->2pi
      fxch
   @@:
      fprem                   ;reduce the angle
      fsin
      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return error
      sahf                    ;transfer to the CPU flags
      jpe   @B                ;reduce angle again if necessary
      fstp  st(1)             ;get rid of the 2pi

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was Src taken from FPU
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU
      jz    @F                ;the result has been stored in memory
```

```
                              ;none of the FPU data was modified

      ffree st(7)             ;free it if not already empty
      fld   tempst            ;load the result on the FPU
   @@:
      or    al,1              ;to insure EAX!=0
      ret

FpuSin endp

; ########################################################################

end



; ########################################################################
;
;                             FpuSinh
;
;########################################################################

  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;
  ;     sinh(Src) = [e^(Src) - e^(-Src)]/2 -> Dest    (see FpuEexpX for e^x)
  ;
  ; This FpuSinh function computes the hyperbolic sine of a number (Src)
  ; with the FPU and returns the result as a REAL number at the specified
  ; destination (the FPU itself or a memory location), unless an invalid
  ; operation is reported by the FPU or the definition of the parameters
  ; (with uID) is invalid.
  ;
  ; The source can be either:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################
```

```
FpuSinh proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;------------------------------
;check if top register is empty
;------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;--------------------------------------
;check source for Src and load it to FPU
;--------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src1 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src1 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
```

```
      test  uID,SRC1_DIMM     ;is Src an immediate 32-bit integer?
      jz    srcerr            ;no correct flag for Src
      fild  lpSrc
      jmp   dest0             ;go complete process

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

dest0:
      fldl2e
      fmul                    ;log2(e)*Src
      fld   st(0)
      frndint
      fxch
      fsub  st,st(1)
      f2xm1
      fld1
      fadd
      fscale                  ;-> antilog[log2(e)*Src] = e^(Src)
      fstp  st(1)             ;get rid of scaling factor

      fld   st(0)             ;copy it to get the reciprocal
      fld1
      fdivrp st(1),st         ;1/e^(Src) = e^(-Src)
      fsub                    ;e^(Src) - e^(-Src)
      fld1
      fchs                    ;-1
      fxch
      fscale                  ;-> [e^(Src) - e^(-Src)]/2 = sinh(Src)
      fstp  st(1)             ;get rid of scaling factor

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return error

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was Src taken from FPU
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU
      jz    @F                ;the result has been stored in memory
```

```
                              ;none of the FPU data was modified

      ffree st(7)             ;free it if not already empty
      fld   tempst            ;load the result on the FPU
   @@:
      or    al,1              ;to insure EAX!=0
      ret

FpuSinh endp

; ########################################################################

end



; ########################################################################
;
;                             FpuSize
;
;########################################################################

  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002.
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameter.
  ;
  ; This FpuSize function computes the exponent of a number (Src) as if it
  ; were expressed in scientific notation and returns the result as a LONG
  ; integer at the specified destination, unless an invalid operation
  ; is reported by the FPU or the definition of the parameters (with uID)
  ; is invalid.
  ;
  ; The source can be either:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory.
  ;
  ; If the source is taken from the FPU, its value will be preserved.
  ; The destination must be a pointer to a 32-bit integer memory variable.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; This function simply computes the common logarithm (base 10) of the
  ; absolute value of the number and returns the characteristic
  ; (i.e. power of 10), adjusted if necessary for a negative log value.
  ; For example,
  ; 5432    would return +3  (log =  3.735)
  ; 5.432   would return  0  (log =  0.735)
  ; 0.05432 would return -2  (log = -1.265)
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved. All FPU registers are preserved.
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuSize proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD
```

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue          ;go check for potential overflow

;-------------------------------
;check if top register is empty
;-------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src1 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src1 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
```

```
      test  uID,SRC1_DIMM     ;is Src an immediate 32-bit integer?
      jz    srcerr            ;no correct flag for Src
      fild  lpSrc
      jmp   dest0             ;go complete process

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

dest0:
      fabs                    ;insures a positive value
      ftst                    ;check the value on the FPU
      fstsw ax                ;store the result flags in AX
      fwait
      sahf                    ;transfer flags to CPU flag register
      jnz   @F                ;not NAN or zero
      jc    srcerr            ;invalid number or infinity
      mov   eax,lpDest
      mov   dword ptr[eax],80000000h     ;code it for 0 value
      jmp   finish            ;this avoids an invalid operation if computing
                              ;the logarithm of zero was attempted

   @@:
      fldlg2                  ;load log10(2)
      fxch                    ;set up registers for next operation
      fyl2x                   ;->[log2(x)]*[log10(2)] = log(x) base 10
      push  eax               ;reserve space on CPU stack
      fstcw [esp]             ;get current control word
      fwait
      mov   ax,[esp]
      and   ax,0f3ffh         ;clear RC field
      or    ax,0400h          ;code it for rounding down
      push  eax
      fldcw [esp]             ;change rounding code of FPU to rounding down
                              ;towards -INFINITY
      mov   eax,lpDest
      fistp dword ptr[eax]    ;store integer result at specified address
      fldcw [esp+4]           ;load back the former control word
      add   esp,8             ;restore stack pointer

finish:
      frstor content
      or    al,1              ;to insure EAX!=0
      ret

FpuSize endp

; ########################################################################

end



; ########################################################################
;
;                             FpuSqrt
;
;########################################################################

  ; ------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified January 2004 to remove data section
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;
  ;                           sqrt(Src) -> Dest
```

```
  ;
  ; This FpuSqrt function extracts the square root of a number (Src)
  ; with the FPU and returns the result as a REAL number at the specified
  ; destination (the FPU itself or a memory location), unless an invalid
  ; operation is reported by the FPU or the definition of the parameters
  ; (with uID) is invalid.
  ;
  ; (If the source is negative, an invalid operation will obviously
  ; be reported.)
  ;
  ; The source can be either:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory, or
  ; one of the FPU constants.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuSqrt proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;-------------------------------
;check if top register is empty
;-------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
```

```
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_CONST
      jnz   constant
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DIMM
      .if   !ZERO?            ;Src is an immediate 32-bit integer?
            fild  lpSrc
            jmp   dest0       ;go complete process
      .endif

      ;otherwise no correct flag for Src

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

constant:
      cmp   eax,FPU_PI
      jz    @F
      fldpi                   ;load pi (3.14159...) on FPU
      jmp   dest0             ;go complete process
   @@:
      cmp   eax,FPU_NAPIER
      jz    srcerr            ;no correct CONST for Src
      fld1
```

```
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

dest0:
      fsqrt                   ;get the square root of the number

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return error

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was Src taken from FPU
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU
      jz    @F                ;the result has been stored in memory
                              ;none of the FPU data was modified

      ffree st(7)             ;free it if not already empty
      fld   tempst            ;load the result on the FPU
   @@:
      or    al,1              ;to insure EAX!=0
      ret

FpuSqrt endp

; ########################################################################

end



; ########################################################################
;
;                             FpuState
;
;########################################################################

  ; ----------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, March 2004
  ; Modified April, 2005, to include an ID of the call in the report
```

```
  ;
  ; This FpuState function converts the content of all the FPU registers
  ; to a null-terminated alphanumeric string at the specified destination
  ; address.
  ;
  ; All the CPU and FPU registers are preserved.
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc
    includelib Fpu.lib

    .code

; ########################################################################

FpuState proc public lpDest:DWORD, uID:DWORD

LOCAL content[108] :BYTE
LOCAL buffer[4]    :BYTE

      pushfd
      pushad
      fsave content
      fwait
      lea   esi,content
      mov   edi,lpDest

;caller ID

      mov   eax,"  DI"
      stosd                   ;write "ID  " to destination
      mov   eax,uID           ;get ID parameter
      xor   ecx,ecx
      push  ecx               ;to be used as a terminator
      mov   cl,10

;convert ID from binary to ASCII

   @@:
      xor   edx,edx
      div   ecx
      add   dl,30h            ;convert remainder to ASCII
      push  edx               ;store it on the stack
      or    eax,eax           ;is conversion completed
      jnz   @B                ;continue if not

;recover ASCII characters one by one and store in buffer

   @@:
      pop   eax
      or    eax,eax           ;is it the end
      jz    @F                ;jump out if it is
      stosb
      jmp   @B

   @@:
      mov   ax,0A0Dh
      stosw                   ;crlf

;Control Word

      mov   eax,"  WC"
      stosd                   ;write "CW  " to destination
      lodsd                   ;get Control Word in AX
      shl   eax,16            ;shift it to the H.O. word
```

```
      mov   ecx,4
   @@:
      mov   al,0
      rol   eax,1
      mov   ah," "
      add   al,"0"
      stosw
      dec   ecx
      jnz   @B                ;last one is the IC field

      xor   ax,ax
      rol   eax,2
      ror   ax,1
      rol   ah,1
      add   ax,3030h
      stosw                   ;write the RC field
      mov   al," "
      stosb

      xor   ax,ax
      rol   eax,2
      ror   ax,1
      rol   ah,1
      add   ax,3030h
      stosw                   ;write the PC field
      mov   ax,"  "
      stosw

      mov   ecx,8
   @@:
      mov   al,0
      rol   eax,1
      mov   ah," "
      add   al,"0"
      stosw
      dec   ecx
      jnz   @B                ;write the interrupt masks
      dec   edi
      mov   ax,0A0Dh
      stosw                   ;crlf

;Status Word

      mov   eax,"  WS"
      stosd                   ;write "SW  " to destination
      lodsd                   ;get Status Word in AX
      shl   eax,16            ;shift it to the H.O. word
      mov   ecx,2
   @@:
      mov   al,0
      rol   eax,1
      mov   ah," "
      add   al,"0"
      stosw
      dec   ecx
      jnz   @B                ;last one is the C3 field

      xor   ax,ax
      rol   eax,3
      mov   buffer,al         ;TOP field
      push  eax
      and   eax,7
      ror   ax,1
      rol   eax,1
      ror   ax,2
      rol   ah,1
      add   eax,20303030h
      stosd                   ;write the TOP field
      pop   eax
```

```
      mov   ecx,3
   @@:
      mov   al,0
      rol   eax,1
      mov   ah," "
      add   al,"0"
      stosw
      dec   ecx
      jnz   @B                ;write the C2, C1, C0 field
      mov   al," "
      stosb

      mov   al,0
      rol   eax,1
      add   al,30h
      mov   ah," "
      stosw
      mov   ecx,7
      jmp   @F
      szflags     db    " IDZOUPS"
   @@:
      mov   al,0
      rol   eax,1
      mov   ah," "
      .if   al == 1
            mov   al,szflags[ecx]
      .else
            mov   al,szflags[ecx]
            or    al,20h
      .endif
      stosw
      dec   ecx
      jnz   @B                ;write the interrupt flags
      dec   edi
      mov   ax,0A0Dh
      stosw                   ;crlf

;Tag Word

      mov   eax,"  WT"
      stosd                   ;write "TW  " to destination
      lodsd                   ;get Tag Word in AX
      mov   cl,buffer
      shl   cl,1
      ror   ax,cl
      shl   eax,16            ;shift it to the H.O. word
      mov   ecx,8
   @@:
      mov   al,0
      ror   eax,2
      rol   ax,2
      push  eax
      .if   al == 0
            mov   eax," LAV"
      .elseif al == 1
            mov   eax," LUN"
      .elseif al == 2
            mov   eax," NaN"
      .else
            mov   eax," ERF"
      .endif
      stosd
      pop   eax
      dec   ecx
      jnz   @B
      dec   edi
      mov   ax,0A0Dh
      stosw                   ;crlf

;Instruction pointer
```

```
      mov   eax,"  PI"
      stosd                   ;write "IP  " to destination
      lodsd                   ;get Instruction pointer in EAX
      mov   ecx,8
   @@:
      rol   eax,4
      push  eax
      and   al,0fh
      add   al,30h
      .if   al > "9"
            add   al,7
      .endif
      stosb                   ;write each nibble as a hex number
      pop   eax
      dec   ecx
      jnz   @B
      mov   ax,0A0Dh
      stosw                   ;crlf

;Code segment

      mov   eax,"  SC"
      stosd                   ;write "CS  " to destination
      lodsd                   ;get Code segment in AX
      shl   eax,16            ;shift it to the H.O. word
      mov   ecx,4
   @@:
      rol   eax,4
      push  eax
      and   al,0fh
      add   al,30h
      .if   al > "9"
            add   al,7
      .endif
      stosb                   ;write each nibble as a hex number
      pop   eax
      dec   ecx
      jnz   @B
      mov   ax,0A0Dh
      stosw                   ;crlf

;Operand address

      mov   eax,"  AO"
      stosd                   ;write "OA  " to destination
      lodsd                   ;get Operand address in EAX
      mov   ecx,8
   @@:
      rol   eax,4
      push  eax
      and   al,0fh
      add   al,30h
      .if   al > "9"
            add   al,7
      .endif
      stosb                   ;write each nibble as a hex number
      pop   eax
      dec   ecx
      jnz   @B
      mov   ax,0A0Dh
      stosw                   ;crlf

;Data segment

      mov   eax,"  SD"
      stosd                   ;write "DS  " to destination
      lodsd                   ;get Data segment in AX
      shl   eax,16            ;shift it to the H.O. word
      mov   ecx,4
```

```
   @@:
      rol   eax,4
      push  eax
      and   al,0fh
      add   al,30h
      .if   al > "9"
            add   al,7
      .endif
      stosb                   ;write each nibble as a hex number
      pop   eax
      dec   ecx
      jnz   @B

      mov   ax,0A0Dh
      stosw                   ;crlf

;Data registers

      xor   ecx,ecx           ;count for registers
datareg:
      mov   ax,0A0Dh
      stosw                   ;crlf

      push  ecx
      mov   eax," 0TS"
      shl   ecx,16
      add   eax,ecx
      stosd                   ;write "STx " to destination

      shr   ecx,16
      add   cl,buffer
      and   cl,7
      shl   cl,1
      lea   eax,content+8
      mov   ax,[eax]          ;get Tag Word in AX
      shr   eax,cl
      and   al,3              ;get Tag of register in AL
      .if   al == 3           ;register is FREE
            mov   eax,"PME "
            stosd
            mov   ax,"YT"
            stosw
      .elseif al == 1         ;register == 0
            invoke FpuExam,esi,SRC1_REAL
            mov   dl,"+"
            test  eax,XAM_NEG
            jz    @F
            mov   dl,"-"
         @@:
            mov   eax," 0  "
            mov   ah,dl
            stosd
      .elseif al == 0         ;valid non-zero number
            invoke FpuFLtoA,esi,15,edi,SRC1_REAL or SRC2_DIMM or STR_SCI
            add   edi,24
      .else
            invoke FpuExam,esi,SRC1_REAL
            test  eax,XAM_VALID
            jnz   @F          ;valid = INFINITY, if not = INDEFINITE
            mov   eax,"DNI "
            stosd
            mov   eax,"NIFE"
            stosd
            mov   eax," ETI"
            stosd
            jmp   nextST
         @@:
            mov   dx,"+ "
            test  eax,XAM_NEG
            jz    @F
```

```
            mov   dh,"-"
         @@:
            mov   ax,dx
            stosw
            mov   eax,"IFNI"
            stosd
            mov   eax,"YTIN"
            stosd
      .endif

nextST:
      add   esi,10
      pop   ecx
      inc   ecx
      cmp   cl,8
      jb    datareg

      mov   byte ptr[edi],0

      frstor content
      fwait
      popad
      popfd
      ret

FpuState endp

; ##########################################################################

end



; ##########################################################################
;
;                              FpuSub
;
;##########################################################################

  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;
  ;                              Src1 - Src2 -> Dest
  ;
  ; This FpuSub function subtracts the Src2 number from the Src1 number
  ; with the FPU and returns the result as a REAL number at the specified
  ; destination (the FPU itself or a memory location), unless an invalid
  ; operation is reported by the FPU or the definition of the parameters
  ; (with uID) is invalid.
  ;
  ; Either of the two sources can be:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory, or
  ; one of the FPU constants.
  ;
  ; None of the sources are checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
```

```
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuSub proc public lpSrc1:DWORD, lpSrc2:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU or SRC2_FPU      ;is data taken from FPU?
      jz    continue

;-------------------------------
;check if top register is empty
;-------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src1 and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src1 is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   src2        ;check next parameter for Src2
      .endif

      mov   eax,lpSrc1
      test  uID,SRC1_CONST
      jnz   constant
      test  uID,SRC1_REAL
      .if !ZERO?              ;Src1 is an 80-bit REAL10 in memory?
            fld   tbyte ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_REAL8
```

```
      .if !ZERO?              ;Src1 is a 64-bit REAL10 in memory?
            fld   qword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_REAL4
      .if !ZERO?              ;Src1 is a 32-bit REAL10 in memory?
            fld   dword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src1 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src1 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif

      test  uID,SRC1_DIMM
      .if   !ZERO?            ;Src1 is an immediate 32-bit integer?
            fild  lpSrc1
            jmp   src2        ;check next parameter for Src2
      .endif

      ;otherwise no valid ID for Src1

srcerr:
      frstor content
srcerr1:
      xor   eax,eax           ;error code
      ret

constant:
      cmp   eax,FPU_PI
      jnz   @F
      fldpi
      jmp   src2
   @@:
      cmp   eax,FPU_NAPIER
      jnz   srcerr            ;no correct CONST for Src1
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

;---------------------------------------
;check source for Src2 and load it to FPU
;---------------------------------------

src2:
      test  uID,SRC2_FPU
      .if   !ZERO?            ;Src2 is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28] ;retrieve it from the stored data
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc2
      test  uID,SRC2_CONST
      jnz   constant2
      test  uID,SRC2_REAL
      .if   !ZERO?            ;Src2 is an 80-bit REAL10 in memory?
            fld   tbyte ptr [eax]
            jmp   dest0       ;go complete process
```

```
      .endif
      test  uID,SRC2_REAL8
      .if   !ZERO?            ;Src2 is a 64-bit REAL10 in memory?
            fld   qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_REAL4
      .if   !ZERO?            ;Src2 is a 32-bit REAL10 in memory?
            fld   dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC2_DMEM
      .if   !ZERO?            ;Src2 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_QMEM
      .if   !ZERO?            ;Src2 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC2_DIMM
      .if   !ZERO?            ;Src2 is an immediate 32-bit integer?
            fild  lpSrc2
            jmp   dest0       ;go complete process
      .endif
      jmp   srcerr            ;no correct flag for Src2

constant2:
      cmp   eax,FPU_PI
      jnz   @F
      fldpi                   ;load pi (3.14159...) on FPU
      jmp   dest0             ;go complete process
   @@:
      cmp   eax,FPU_NAPIER
      jnz   srcerr            ;no correct CONST for Src2
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

dest0:
      fsub
      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   eax,1             ;test for invalid operation
      jc    srcerr            ;clean-up and return error

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
```

```
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU or SRC2_FPU      ;was any data taken from FPU?
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU
      jz    @F                ;the result has been stored in memory
                              ;none of the FPU data was modified

      ffree st(7)             ;free it if not already empty
      fld   tempst            ;load the result on the FPU
   @@:
      or    al,1              ;to insure EAX!=0
      ret

FpuSub endp

; ########################################################################

end



; ########################################################################
;
;                             FpuTan
;
;########################################################################

  ; ----------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified January 2004 to prevent stack faults and to adjust
  ; angles outside the acceptable range if necessary.
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;
  ;                           a = tan(x)
  ;
  ; This FpuTan function computes the tangent of an angle in degrees or radians
  ; (Src) with the FPU and returns the result as a REAL number at the
  ; specified destination (the FPU itself or a memory location), unless
  ; an invalid operation is reported by the FPU or the definition of the
  ; parameters (with uID) is invalid.
  ;
  ; The source can be either:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
```

```
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuTan proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;--------------------------------
;check if top register is empty
;--------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;----------------------------------------
;check source for Src and load it to FPU
;----------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
```

```
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src1 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src1 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DIMM     ;is Src an immediate 32-bit integer?
      jz    srcerr            ;no correct flag for Src
      fild  lpSrc
      jmp   dest0             ;go complete process
srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

dest0:
      fldpi                   ;load pi (3.14159...) on FPU
      fadd  st,st             ;->2pi
      fxch
      test  uID,ANG_RAD
      jnz   @F                ;jump if angle already in radians
      fmul  st,st(1)
      pushd 360
      fidiv word ptr[esp]     ;value now in radians
      fwait
      pop   eax               ;clean the stack
   @@:
      fprem                   ;reduce the angle
      fptan
      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return error
      sahf                    ;transfer to the CPU flags
      jpe   @B                ;reduce angle again if necessary
      fstp  st                ;get rid of the 1
      fstp  st(1)             ;get rid of the 2pi

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
```

```
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was Src taken from FPU
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU
      jz    @F                ;the result has been stored in memory
                              ;none of the FPU data was modified

      ffree st(7)             ;free it if not already empty
      fld   tempst            ;load the result on the FPU
   @@:
      or    al,1              ;to insure EAX!=0
      ret

FpuTan endp

; ########################################################################

end



; ########################################################################
;
;                             FpuTanh
;
;########################################################################

  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;
  ;                             sinh(Src)
  ;               tanh(Src) = ---------  -> Dest     (see related functions)
  ;                             cosh(Src)
  ;
  ; This FpuTanh function computes the hyperbolic tangent of a number (Src)
  ; with the FPU and returns the result as a REAL number at the
  ; specified destination (the FPU itself or a memory location), unless
  ; an invalid operation is reported by the FPU or the definition of the
  ; parameters (with uID) is invalid.
  ;
  ; The source can be either:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
```

```
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc
    includelib Fpu.lib

    .code

; ########################################################################

FpuTanh proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;-------------------------------
;check if top register is empty
;-------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;----------------------------------------
;check source for Src and load it to FPU
;----------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
```

```
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src1 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src1 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DIMM     ;is Src an immediate 32-bit integer?
      jz    srcerr            ;no correct flag for Src
      fild  lpSrc
      jmp   dest0             ;go complete process

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

dest0:
      fld   st(0)

      invoke FpuSinh,0,0,SRC1_FPU or DEST_FPU
      or    eax,eax
      jz    srcerr
      fxch
      invoke FpuCosh,0,0,SRC1_FPU or DEST_FPU
      or    eax,eax
      jz    srcerr
      fdiv                    ;sinh/cosh=tanh

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was Src taken from FPU
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU
      jz    @F                ;the result has been stored in memory
                              ;none of the FPU data was modified

      ffree st(7)             ;free it if not already empty
      fld   tempst            ;load the result on the FPU
```

```
   @@:
      or    al,1              ;to insure EAX!=0
      ret

FpuTanh endp

; ########################################################################

end



; ########################################################################
;
;                              FpuTexpX
;
;########################################################################

  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;
  ;            10^(Src) = antilog2[ log2(10) * Src ] -> Dest
  ;
  ; This FpuTexpX function computes the base 10 antilogarithm of a number.
  ; It raises 10 to the power of the Src number with the FPU and returns
  ; the result as a REAL number at the specified destination (the FPU
  ; itself or a memory location), unless an invalid operation is reported
  ; by the FPU or the definition of the parameters (with uID) is invalid.
  ;
  ; The exponent can be either:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory, or
  ; one of the FPU constants.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuTexpX proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD
```

```
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;-------------------------------
;check if top register is empty
;-------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_CONST
      jnz   constant
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
```

```
      test  uID,SRC1_DIMM
      .if   !ZERO?           ;Src is an immediate 32-bit integer?
            fild  lpSrc
            jmp   dest0      ;go complete process
      .endif

      ;otherwise no correct flag for Src

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

constant:
      cmp   eax,FPU_PI
      jz    @F
      fldpi                  ;load pi (3.14159...) on FPU
      jmp   dest0            ;go complete process
   @@:
      cmp   eax,FPU_NAPIER
      jz    srcerr           ;no correct CONST for Src
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

dest0:
      fldl2t                 ;->log2(10)
      fmul                   ;->log2(10)*Src

;the FPU can compute the antilog only with the mantissa
;the characteristic of the logarithm must thus be removed

      fld   st               ;copy the logarithm
      frndint                ;keep only the characteristic
      fsub  st(1),st         ;keeps only the mantissa
      fxch                   ;get the mantissa on top

      f2xm1                  ;->2^(mantissa)-1
      fld1
      fadd                   ;add 1 back

;the number must now be readjusted for the characteristic of the logarithm

      fscale                 ;scale it with the characteristic

      fstsw ax               ;retrieve exception flags from FPU
      fwait
      shr   al,1             ;test for invalid operation
      jc    srcerr           ;clean-up and return error

;the characteristic is still on the FPU and must be removed

      fstp  st(1)            ;overwrite the characteristic

; store result as specified

      test  uID,DEST_FPU     ;check where result should be stored
      .if   !ZERO?           ;destination is the FPU
            fstp  tempst     ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?           ;store as REAL4 at specified address
            fstp  dword ptr[eax]
```

```
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was Src taken from FPU
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU
      jz    @F                ;the result has been stored in memory
                              ;none of the FPU data was modified

      ffree st(7)             ;free it if not already empty
      fld   tempst            ;load the result on the FPU
   @@:
      or    al,1              ;to insure EAX!=0
      ret

FpuTexpX endp

; ##########################################################################

end



; ##########################################################################
;
;                             FpuTrunc
;
;##########################################################################

  ; ------------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified January 2004 to eliminate data section
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;    Added checking for overflow when storing the result as an integer.
  ;
  ; This FpuTrunc function discards the fractional part of a REAL
  ; number (Src) and returns the integer portion at the specified
  ; destination, unless an invalid operation is reported by the FPU
  ; or the definition of the parameters (with uID) is invalid.
  ;
  ; The source can be either:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory.
  ;
  ; The source is not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
```

```
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
  ; ------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuTrunc proc public lpSrc:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU      ;is Src taken from FPU?
      jz    continue

;------------------------------
;check if top register is empty
;------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc
      test  uID,SRC1_REAL
      .if   !ZERO?            ;Src is an 80-bit REAL10 in memory?
            fld   tbyte ptr[eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC1_REAL8
      .if   !ZERO?            ;Src is a 64-bit REAL8 in memory?
            fld   qword ptr[eax]
            jmp   dest0       ;go complete process
      .endif
```

```
      test  uID,SRC1_REAL4
      .if   !ZERO?            ;Src is a 32-bit REAL4 in memory?
            fld   dword ptr[eax]
            jmp   dest0       ;go complete process
      .endif

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

dest0:
      push  eax               ;reserve space on stack
      fstcw [esp]             ;get current control word
      fwait
      mov   ax,[esp]
      or    ax,0c00h          ;code it for truncating
      push  eax
      fldcw [esp]             ;change rounding code of FPU to truncate

      frndint                 ;truncate the number
      pop   eax               ;remove modified CW from CPU stack
      fldcw [esp]             ;load back the former control word
      fwait
      pop   eax               ;clean CPU stack

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return error

; store result as specified

      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_IMEM
      .if   !ZERO?            ;store as DWORD at specified address
            fistp dword ptr[eax]
            jmp   integersave
      .endif
      test  uID,DEST_IMEM8
      .if   !ZERO?            ;store as QWORD at specified address
            fistp qword ptr[eax]
            jmp   integersave
      .endif
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU      ;was any data taken from FPU?
      jz    @F
      fstp  st                ;remove source

   @@:
```

```
      test  uID,DEST_FPU
      jz    @F                ;the result has been stored in memory
                              ;none of the FPU data was modified

      ffree st(7)             ;free it if not already empty
      fld   tempst            ;load the result on the FPU
   @@:
      or    al,1              ;to insure EAX!=0
      ret

integersave:
      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return error
      jmp   restore

FpuTrunc endp

; ########################################################################

end



; ########################################################################
;
;                             FpuXexpY
;
;########################################################################

  ; ----------------------------------------------------------------------
  ; This procedure was written by Raymond Filiatreault, December 2002
  ; Modified March 2004 to avoid any potential data loss from the FPU
  ; Revised January 2005 to free the FPU st7 register if necessary.
  ; Revised January 2010 to allow additional data types from memory to be
  ;    used as source parameters and allow additional data types for storage.
  ;
  ;            Src1^Src2 = antilog2[ log2(Src1) * Src2 ] -> Dest
  ;
  ; This FpuXexpY function raises a number (Src1) to a power (Src2)
  ; with the FPU and returns the result as a REAL number at the specified
  ; destination (the FPU itself or a memory location), unless an invalid
  ; operation is reported by the FPU or the definition of the parameters
  ; (with uID) is invalid.
  ;
  ; Either of the two sources can be:
  ; a REAL number from the FPU itself, or
  ; a REAL4, REAL8 or REAL10 from memory, or
  ; an immediate DWORD integer value, or
  ; a DWORD or QWORD integer from memory, or
  ; one of the FPU constants.
  ; The base number (Src1) must be positive.
  ;
  ; The sources are not checked for validity. This is the programmer's
  ; responsibility.
  ;
  ; Only EAX is used to return error or success. All other CPU registers
  ; are preserved.
  ;
  ; IF a source is specified to be the FPU top data register, it would be
  ; removed from the FPU. It would be replaced by the result only if the
  ; FPU is specified as the destination.
  ;
  ; IF source data is only from memory
  ; AND the FPU is specified as the destination for the result,
  ;       the st7 data register will become the st0 data register where the
  ;       result will be returned (any valid data in that register would
  ;       have been trashed).
  ;
```

```
  ; ------------------------------------------------------------------------

    .386
    .model flat, stdcall  ; 32 bit memory model
    option casemap :none  ; case sensitive

    include Fpu.inc

    .code

; ########################################################################

FpuXexpY proc public lpSrc1:DWORD, lpSrc2:DWORD, lpDest:DWORD, uID:DWORD

;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;
; Because a library is assembled before its functions are called, all
; references to external memory data must be qualified for the expected
; size of that data so that the proper code is generated.
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LOCAL content[108] :BYTE
LOCAL tempst       :TBYTE

      test  uID,SRC1_FPU or SRC2_FPU      ;is data taken from FPU?
      jz    continue

;-------------------------------
;check if top register is empty
;-------------------------------

      fxam                    ;examine its content
      fstsw ax                ;store results in AX
      fwait                   ;for precaution
      sahf                    ;transfer result bits to CPU flag
      jnc   continue          ;not empty if Carry flag not set
      jpe   continue          ;not empty if Parity flag set
      jz    srcerr1           ;empty if Zero flag set

continue:
      fsave content

;---------------------------------------
;check source for Src1 and load it to FPU
;---------------------------------------

      test  uID,SRC1_FPU
      .if   !ZERO?            ;Src1 is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28]
            jmp   src2        ;check next parameter for Src2
      .endif

      mov   eax,lpSrc1
      test  uID,SRC1_CONST
      jnz   constant
      test  uID,SRC1_REAL
      .if !ZERO?              ;Src1 is an 80-bit REAL10 in memory?
            fld   tbyte ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_REAL8
      .if !ZERO?              ;Src1 is a 64-bit REAL10 in memory?
            fld   qword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_REAL4
      .if !ZERO?              ;Src1 is a 32-bit REAL10 in memory?
            fld   dword ptr [eax]
```

```
            jmp   src2        ;check next parameter for Src2
      .endif

      test  uID,SRC1_DMEM
      .if !ZERO?              ;Src1 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif
      test  uID,SRC1_QMEM
      .if !ZERO?              ;Src1 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   src2        ;check next parameter for Src2
      .endif

      test  uID,SRC1_DIMM
      .if   !ZERO?            ;Src1 is an immediate 32-bit integer?
            fild  lpSrc1
            jmp   src2        ;check next parameter for Src2
      .endif

      ;otherwise no correct flag for Src1

srcerr:
      frstor content
srcerr1:
      xor   eax,eax
      ret

constant:
      cmp   eax,FPU_PI
      jnz   @F
      fldpi                   ;load pi (3.14159...) on FPU
      jmp   src2              ;check next parameter Src2 for exponent
   @@:
      cmp   eax,FPU_NAPIER
      jnz   srcerr            ;no correct CONST for Src1
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

;----------------------------------------
;check source for Src2 and load it to FPU
;----------------------------------------

src2:
      test  uID,SRC2_FPU
      .if   !ZERO?            ;Src2 is taken from FPU?
            lea   eax,content
            fld   tbyte ptr[eax+28] ;retrieve it from the stored data
            jmp   dest0       ;go complete process
      .endif

      mov   eax,lpSrc2
      test  uID,SRC2_CONST
      jnz   constant2
      test  uID,SRC2_REAL
      .if   !ZERO?            ;Src2 is an 80-bit REAL10 in memory?
            fld   tbyte ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_REAL8
      .if   !ZERO?            ;Src2 is a 64-bit REAL10 in memory?
            fld   qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_REAL4
```

```
      .if   !ZERO?            ;Src2 is a 32-bit REAL10 in memory?
            fld   dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC2_DMEM
      .if   !ZERO?            ;Src2 is a 32-bit integer in memory?
            fild  dword ptr [eax]
            jmp   dest0       ;go complete process
      .endif
      test  uID,SRC2_QMEM
      .if   !ZERO?            ;Src2 is a 64-bit integer in memory?
            fild  qword ptr [eax]
            jmp   dest0       ;go complete process
      .endif

      test  uID,SRC2_DIMM
      .if   !ZERO?            ;Src2 is an immediate 32-bit integer?
            fild  lpSrc2
            jmp   dest0       ;go complete process
      .endif
      jmp   srcerr            ;no correct flag for Src2

constant2:
      cmp   eax,FPU_PI
      jnz   @F
      fldpi                   ;load pi (3.14159...) on FPU
      jmp   dest0             ;go complete process
   @@:
      cmp   eax,FPU_NAPIER
      jnz   srcerr            ;no correct CONST for Src1
      fld1
      fldl2e
      fsub  st,st(1)
      f2xm1
      fadd  st,st(1)
      fscale
      fstp  st(1)

dest0:
      fxch                    ;set up FPU registers for next operation
      fyl2x                   ;->log2(Src1)*exponent

;the FPU can compute the antilog only with the mantissa
;the characteristic of the logarithm must thus be removed

      fld   st(0)             ;copy the logarithm
      frndint                 ;keep only the characteristic
      fsub  st(1),st          ;keeps only the mantissa
      fxch                    ;get the mantissa on top

      f2xm1                   ;->2^(mantissa)-1
      fld1
      fadd                    ;add 1 back

;the number must now be readjusted for the characteristic of the logarithm

      fscale                  ;scale it with the characteristic

      fstsw ax                ;retrieve exception flags from FPU
      fwait
      shr   al,1              ;test for invalid operation
      jc    srcerr            ;clean-up and return error

;the characteristic is still on the FPU and must be removed

      fstp  st(1)             ;overwrite the characteristic

; store result as specified
```

```
      test  uID,DEST_FPU      ;check where result should be stored
      .if   !ZERO?            ;destination is the FPU
            fstp  tempst      ;store it temporarily
            jmp   restore
      .endif
      mov   eax,lpDest
      test  uID,DEST_MEM4
      .if   !ZERO?            ;store as REAL4 at specified address
            fstp  dword ptr[eax]
            jmp   restore
      .endif
      test  uID,DEST_MEM8
      .if   !ZERO?            ;store as REAL8 at specified address
            fstp  qword ptr[eax]
            jmp   restore
      .endif
      fstp  tbyte ptr[eax]    ;store as REAL10 at specified address (default)

restore:
      frstor  content         ;restore all previous FPU registers

      test  uID,SRC1_FPU or SRC2_FPU      ;was any data taken from FPU?
      jz    @F
      fstp  st                ;remove source

   @@:
      test  uID,DEST_FPU
      jz    @F                ;the result has been stored in memory
                              ;none of the FPU data was modified

      ffree st(7)             ;free it if not already empty
      fld   tempst            ;load the result on the FPU
   @@:
      or    al,1              ;to insure EAX!=0
      ret

FpuXexpY endp

; ########################################################################

end



; ########################################################################
;
;                               smdtoa
;
; Procedure for converting a 32-bit signed integer to ASCII without
; any leading "0"
;
; Return:   Null-terminated ASCII string in memory
;           EAX = address of first character of the string
;           ECX = number of characters in string (excluding terminating 0)
;
; Usage:    invoke smdtoa,src,lpBuf
;
;   where:
;   src   = dword to be converted
;   lpBuf = pointer to a memory buffer for the null-terminated ASCII
;           string (a 12-byte buffer is sufficient for the largest of
;           numbers, i.e. 10 digits maximum, sign, and the terminating 0)
;
; EAX, ECX and EDX are trashed. EBX, ESI and EDI are preserved.
;
; NOTE: This procedure cannot determine if the source dword is a valid
;       signed DWORD INTEGER. It is the responsibility of the
;       programmer to insure that. No error code can be returned.
;
;                       by Raymond Filiatreault
```

```
;                           Otober 2009
;                           MASM  syntax
;
; The "m" in the procedure name is to indicate that this procedure is
; based on the principal of multiplying by the reciprocal of 10 instead
; of dividing by 10, i.e. the use of "magic numbers".
;
; ########################################################################

      .686                  ; minimum processor needed for 32 bit
      .model flat, stdcall  ; FLAT memory model & STDCALL calling
      option casemap :none  ; set code to case sensitive

; ########################################################################

;smdtoa PROTO :DWORD, :DWORD

.code

smdtoa proc uses ebx edi src:DWORD, lpdest:DWORD

      mov   edi,lpdest        ;buffer address
      mov   eax,src
      add   edi,11            ;=>12th byte of the buffer
      push  edi
      mov   ecx,0CCCCCCCDh    ;"magic number" multiplier for division by 10
      mov   ebx,10
      mov   byte ptr[edi],0   ;string terminating 0
      test  eax,eax
      .if   SIGN?
            neg   eax
      .endif
   @@:
      mul   ecx               ;multiply by magic number
      shrd  eax,edx,3         ;binary fractional "remainder" back in EAX
      shr   edx,3             ;EDX = quotient
      inc   eax               ;precaution against occasional "underflow"
      push  edx               ;save current quotient for next division
      mul   ebx               ;x10 gets "decimal" remainder into EDX
      dec   edi               ;back to previous digit in buffer
      add   dl,30h            ;convert remainder to ascii
      mov   [edi],dl          ;insert it into buffer
      pop   eax               ;retrieve current quotient
      test  eax,eax           ;test if done
      jnz   @B                ;continue if not done
      mov   eax,src           ;retrieve original dword
      test  eax,eax
      .if   SIGN?
            dec   edi
            mov   byte ptr[edi],"-"
      .endif
      pop   ecx
      mov   eax,edi           ;EAX = address of first character
      sub   ecx,edi           ;ECX = number of characters
      ret

smdtoa endp

end



; ########################################################################
;
;                               smqtoa
;
; Procedure for converting a 64-bit signed integer from memory to ASCII
; without any leading "0"
;
; Return:   Null-terminated ASCII string in memory
```

```
;           EAX = address of first character of the string
;           ECX = number of characters in string (excluding terminating 0)
;
; Usage:    invoke smqtoa,lpQW,lpBuf
;
;   where:
;   lpQW  = pointer to the location of the qword in memory
;   lpBuf = pointer to a memory buffer for the null-terminated ASCII
;           string (a 22-byte buffer is sufficient for the largest of
;           numbers, i.e. 20 digits maximum, sign and the terminating 0)
;
; EAX, ECX and EDX are trashed. EBX, ESI and EDI are preserved.
;
; NOTE: This procedure cannot determine if the data pointed to by
;       lpQW is a valid signed QWORD INTEGER. It is the responsibility
;       of the programmer to insure that. No error code can be returned.
;
;                       by Raymond Filiatreault
;                            Otober 2009
;                            MASM  syntax
;
; The "m" in the procedure name is to indicate that this procedure is
; based on the principal of multiplying by the reciprocal of 10 instead
; of dividing by 10, i.e. the use of "magic numbers".
;
; ########################################################################

      .686                    ; minimum processor needed for 32 bit
      .model flat, stdcall    ; FLAT memory model & STDCALL calling
      option casemap :none    ; set code to case sensitive

; ########################################################################

;smqtoa PROTO :DWORD, :DWORD

.code

smqtoa proc uses ebx esi edi lpsrc:DWORD, lpdest:DWORD

LOCAL x10 :DWORD
LOCAL r10L :DWORD
LOCAL r10H :DWORD
LOCAL t1 :DWORD
LOCAL t2 :DWORD

      mov   esi,lpsrc         ;memory address of QWORD
      mov   edi,lpdest        ;buffer address
      mov   ebx,[esi]         ;EBX = low DWORD of QWORD
      mov   esi,[esi+4]       ;ESI = high DWORD of QWORD
      add   edi,21            ;=>22th byte of the buffer
      push  edi
      mov   byte ptr[edi],0   ;string terminating 0
      mov   r10L,0CCCCCCCDh   ;low DWORD of "magic number" multiplier
      mov   r10H,0CCCCCCCCh   ;high DWORD of "magic number" multiplier
      mov   x10,10

      test  esi,esi
      push  esi               ;save for retrieving sign later
      .if   SIGN?
            not   esi
            not   ebx
            add   ebx,1
            adc   esi,0
      .endif

; multiplications of two 64-bit numbers will be required as long as
; the "quotient" is greater than a DWORD
; the result of such a multiplication has potentially up to 127 bits

   @@:
```

```
      .if   esi != 0          ;if still greater than a DWORD
            xor   ecx,ecx     ;for the 96-127 bits

;multiply the "magic number" by the low DWORD

            mov   eax,r10L
            mul   ebx         ;multiply by the low DWORD
            mov   t1,edx      ;keep only the higher 32 bits (32-63)
            mov   eax,r10H
            mul   ebx
            add   t1,eax      ;add to the previous 32 bits
            adc   edx,0       ;add any overflow to the 64-95 bits
            mov   t2,edx      ;store those bits

;multiply the "magic number" by the high DWORD and add

            mov   eax,r10L
            mul   esi         ;multiply by the high DWORD
            add   t1,eax      ;add the 32-63 bits
            adc   t2,edx      ;add with carry the 64-95 bits
            adc   ecx,0       ;transfer the carry to the 96-127 bits
            mov   eax,r10H
            mul   esi
            add   eax,t2      ;add with the previous 64-95 bits
            adc   edx,ecx     ;add with carry with the previous 96-127 bits
            mov   ecx,t1      ;retrieve the lower 32-63 bits
            shrd  ecx,eax,3   ;binary fractional "remainder" back in ECX
            shrd  eax,edx,3   ;low DWORD of quotient in EAX
            inc   ecx         ;precaution against occasional "underflow"
            shr   edx,3       ;high DWORD of quotient in EDX
            mov   ebx,eax     ;low DWORD of quotient back in EBX
            mov   esi,edx     ;high DWORD of quotient back in ESI
            mov   eax,ecx     ;binary fractional "remainder" back in EAX
            mul   x10         ;=>"decimal" remainder into EDX
            add   dl,30h      ;convert remainder to ascii
            dec   edi         ;back to previous digit in buffer
            mov   [edi],dl    ;insert it into buffer
            jmp   @B
      .endif

; multiplications of DWORDs will be sufficient after "quotient" is
; reduced to a DWORD.

   @@:
      mov   eax,ebx           ;current DWORD quotient
      mul   r10L              ;multiply by "magic number" for DWORD
      shrd  eax,edx,3         ;binary fractional "remainder" back in EAX
      shr   edx,3             ;EDX = quotient
      inc   eax               ;precaution against occasional "underflow"
      mov   ebx,edx           ;save current quotient in EBX
      mul   x10               ;=> "decimal" remainder into EDX
      dec   edi               ;back to previous digit in buffer
      add   dl,30h            ;convert remainder to ascii
      mov   [edi],dl          ;insert it into buffer
      test  ebx,ebx           ;test if done
      jnz   @B                ;continue if not done
      pop   esi               ;retrieve high DWORD of original QWORD
      test  esi,esi
      jns   @F
      dec   edi
      mov   byte ptr[edi],"-"
   @@:
      pop   ecx
      mov   eax,edi           ;EAX = address of first character
      sub   ecx,edi           ;ECX = number of characters
      ret

smqtoa endp

end
```

```
; ##########################################################################
;
;                               umdtoa
;
; Procedure for converting a 32-bit unsigned integer to ASCII without
; any leading "0"
;
; Return:   Null-terminated ASCII string in memory
;           EAX = address of first character of the string
;           ECX = number of characters in string (excluding terminating 0)
;
; Usage:    invoke umdtoa,src,lpBuf
;
;   where:
;   src   = dword to be converted
;   lpBuf = pointer to a memory buffer for the null-terminated ASCII
;           string (an 11-byte buffer is sufficient for the largest of
;           numbers, i.e. 10 digits maximum, and the terminating 0)
;
; EAX, ECX and EDX are trashed. EBX, ESI and EDI are preserved.
;
; NOTE: This procedure cannot determine if the source dword
;       is a valid unsigned DWORD INTEGER. It is the responsibility
;       of the programmer to insure that. No error code can be returned.
;
;                       by Raymond Filiatreault
;                            Otober 2009
;                            MASM  syntax
;
; The "m" in the procedure name is to indicate that this procedure is
; based on the principal of multiplying by the reciprocal of 10 instead
; of dividing by 10, i.e. the use of "magic numbers".
;
; ##########################################################################

      .686                    ; minimum processor needed for 32 bit
      .model flat, stdcall    ; FLAT memory model & STDCALL calling
      option casemap :none    ; set code to case sensitive

; ##########################################################################

;umdtoa PROTO :DWORD, :DWORD

.code

umdtoa proc uses ebx edi src:DWORD, lpdest:DWORD

      mov   edi,lpdest        ;buffer address
      mov   eax,src
      add   edi,10            ;=>11th byte of the buffer
      push  edi
      mov   ecx,0CCCCCCCDh    ;"magic number" multiplier for division by 10
      mov   ebx,10
      mov   byte ptr[edi],0   ;string terminating 0
   @@:
      mul   ecx               ;multiply by magic number
      shrd  eax,edx,3         ;binary fractional "remainder" back in EAX
      shr   edx,3             ;EDX = quotient
      inc   eax               ;precaution against occasional "underflow"
      push  edx               ;save current quotient for next division
      mul   ebx               ;x10 gets "decimal" remainder into EDX
      dec   edi               ;back to previous digit in buffer
      add   dl,30h            ;convert remainder to ascii
      mov   [edi],dl          ;insert it into buffer
      pop   eax               ;retrieve current quotient
      test  eax,eax           ;test if done
      jnz   @B                ;continue if not done
```

```
      pop   ecx
      mov   eax,edi           ;EAX = address of first character
      sub   ecx,edi           ;ECX = number of characters
      ret

umdtoa endp

end



; ########################################################################
;
;                               umqtoa
;
; Procedure for converting a 64-bit unsigned integer from memory to ASCII
; without any leading "0"
;
; Return:   Null-terminated ASCII string in memory
;           EAX = address of first character of the string
;           ECX = number of characters in string (excluding terminating 0)
;
; Usage:    invoke umqtoa,lpQW,lpBuf
;
;   where:
;   lpQW  = pointer to the location of the qword in memory
;   lpBuf = pointer to a memory buffer for the null-terminated ASCII
;           string (a 21-byte buffer is sufficient for the largest of
;           numbers, i.e. 20 digits maximum, and the terminating 0)
;
; EAX, ECX and EDX are trashed. EBX, ESI and EDI are preserved.
;
; NOTE: This procedure cannot determine if the data pointed to by
;       lpQW is a valid unsigned QWORD INTEGER. It is the responsibility
;       of the programmer to insure that. No error code can be returned.
;
;                        by Raymond Filiatreault
;                             Otober 2009
;                             MASM  syntax
;
; The "m" in the procedure name is to indicate that this procedure is
; based on the principal of multiplying by the reciprocal of 10 instead
; of dividing by 10, i.e. the use of "magic numbers".
;
; ########################################################################

      .686                    ; minimum processor needed for 32 bit
      .model flat, stdcall    ; FLAT memory model & STDCALL calling
      option casemap :none    ; set code to case sensitive

; ########################################################################

;umqtoa PROTO :DWORD, :DWORD

.code

umqtoa proc uses ebx esi edi lpsrc:DWORD, lpdest:DWORD

LOCAL x10 :DWORD
LOCAL r10L :DWORD
LOCAL r10H :DWORD
LOCAL t1 :DWORD
LOCAL t2 :DWORD

      mov   esi,lpsrc         ;memory address of QWORD
      mov   edi,lpdest        ;buffer address
      mov   ebx,[esi]         ;EBX = low DWORD of QWORD
      mov   esi,[esi+4]       ;ESI = high DWORD of QWORD
      add   edi,20            ;=>21th byte of the buffer
```

```
      push  edi
      mov   byte ptr[edi],0   ;string terminating 0
      mov   r10L,0CCCCCCCDh   ;low DWORD of "magic number" multiplier
      mov   r10H,0CCCCCCCCh   ;high DWORD of "magic number" multiplier
      mov   x10,10

; multiplications of two 64-bit numbers will be required as long as
; the "quotient" is greater than a DWORD
; the result of such a multiplication has potentially up to 128 bits

   @@:
      .if   esi != 0          ;if still greater than a DWORD
            xor   ecx,ecx     ;for the 96-127 bits

;multiply the "magic number" by the low DWORD

            mov   eax,r10L
            mul   ebx         ;multiply by the low DWORD
            mov   t1,edx      ;keep only the higher 32 bits (32-63)
            mov   eax,r10H
            mul   ebx
            add   t1,eax      ;add to the previous 32 bits
            adc   edx,0       ;add any overflow to the 64-95 bits
            mov   t2,edx      ;store those bits

;multiply the "magic number" by the high DWORD and add

            mov   eax,r10L
            mul   esi         ;multiply by the high DWORD
            add   t1,eax      ;add the 32-63 bits
            adc   t2,edx      ;add with carry the 64-95 bits
            adc   ecx,0       ;transfer the carry to the 96-127 bits
            mov   eax,r10H
            mul   esi
            add   eax,t2      ;add with the previous 64-95 bits
            adc   edx,ecx     ;add with carry with the previous 96-127 bits
            mov   ecx,t1      ;retrieve the lower 32-63 bits
            shrd  ecx,eax,3   ;binary fractional "remainder" back in ECX
            shrd  eax,edx,3   ;low DWORD of quotient in EAX
            inc   ecx         ;precaution against occasional "underflow"
            shr   edx,3       ;high DWORD of quotient in EDX
            mov   ebx,eax     ;low DWORD of quotient back in EBX
            mov   esi,edx     ;high DWORD of quotient back in ESI
            mov   eax,ecx     ;binary fractional "remainder" back in EAX
            mul   x10         ;=>"decimal" remainder into EDX
            add   dl,30h      ;convert remainder to ascii
            dec   edi         ;back to previous digit in buffer
            mov   [edi],dl    ;insert it into buffer
            jmp   @B
      .endif

; multiplications of DWORDs will be sufficient after "quotient" is
; reduced to a DWORD.

   @@:
      mov   eax,ebx           ;current DWORD quotient
      mul   r10L              ;multiply by "magic number" for DWORD
      shrd  eax,edx,3         ;binary fractional "remainder" back in EAX
      shr   edx,3             ;EDX = quotient
      inc   eax               ;precaution against occasional "underflow"
      mov   ebx,edx           ;save current quotient in EBX
      mul   x10               ;=> "decimal" remainder into EDX
      dec   edi               ;back to previous digit in buffer
      add   dl,30h            ;convert remainder to ascii
      mov   [edi],dl          ;insert it into buffer
      test  ebx,ebx           ;test if done
      jnz   @B                ;continue if not done

      pop   ecx
      mov   eax,edi           ;EAX = address of first character
```

```
      sub   ecx,edi           ;ECX = number of characters
      ret

umqtoa endp

end
```