. .

AmoK - The Art of Coding
http://www.amok.am

presents

AmoK x86Asm Integer BigLib

Version 1.0

This is the documentation of a library providing you an easy to use programmer's interface for big numbers from 64 to 16384 bits. It comes with a mainly complete instruction set of arithmetical and binary operations oriented by the x86 CPU instructions.

The library is available for dynamic and static linking in COEFF format. There are seperate packages of includefiles for C/C++ and Masm32. See also the code examples at the end of this document.

I would like to thank Slash for his great support on testing and debugging.

Feel free to contact me. See `http://www.amok.am` for details.

Bratalarm

1 BigLib - Framework

HBIG **BigCreate** (DWORD bigsize, DWORD biginitflag, DWORD bigopt1, DWORD bigopt2);

This method creates a new BigNum of given size *bigsize* and returns a handle `HBIG` of the BigNum wich can be seen as an identifier of the BigNum.

bigsize The size of the BigNum that will be created. Use one of the following flags: `BIG_64, BIG_128, BIG_256, BIG_512, BIG_1024, BIG_2048, BIG_4096, BIG_8192` or `BIG_16384` where the value x of `BIG_x` means the size in bits.

biginitflag Defines the way of initialization. The use of the optional arguments **bigopt1** and **bigopt2** depends on one of the following flags:

`BIG_ZERO` BigNum is initialized to 0. bigopt1 and bigopt2 are ignored.

`BIG_INT` BigNum is initialized with the value of an unsigned integer bigopt1.

`BIG_STRING` bigopt1 points to an alphanumerical string with number-base bigopt2. The number-base is a value between 2 and 64. For a base of x the first x-chars of the following chartable are used:

```
0123456789ABCDEFGHIJKLMNOPQRSTUV
WXYZabcdefghijklmnopqrstuvwxyz{|
```

`BIG_BYTES` BigNum is initialized by a big-endian bytearray referenced by *bigopt1*. *bigopt2* is the length of the bytearray. If bigopt2 is greater than the maximum possible number of bytes, not all bytes will be used.

`BIG_BIG` *bigop1* is a handle to a BigNum whose value will be the init-value of the new BigNum.

DWORD **BigDestroy** (HBIG a);

This will free the memory of the given BigNum a. Destroy unused BigNums to avoid memory leakage.

DWORD **BigFromString** (HBIG a, char *pString, unsigned int numbase); $a \leftarrow$ convert(pString)

Movs the interpreted value of the base *numbase* alphanumerical string *pString* to a.

DWORD **BigToString** (HBIG a, char *pString, unsigned int numbase); pString $\leftarrow$ convert(a)

Converts the value of a into an alphanumerical string of base *numbase* referenced by *pString*.

DWORD **BigConvertMov** (HBIG a, HBIG b); $a \leftarrow b$

This is the only function where a and b can have different sizes. If a has a smaller size than b a truncated value of b will be moved into a.

2 BigLib - Instructionset

DWORD	**BigAdd** (HBIG a, HBIG b);	$a \leftarrow a + b$
DWORD	**BigAnd** (HBIG a, HBIG b);	$a \leftarrow a$ AND b
DWORD	**BigCmp** (HBIG a, HBIG b);	
DWORD	**BigDec** (HBIG a);	$a \leftarrow a - 1$
DWORD	**BigDiv** (HBIG a, HBIG b, HBIG r);	$a \leftarrow \lfloor a/b \rfloor,\ r \leftarrow a \bmod b$
DWORD	**BigGcd** (HBIG a, HBIG b, HBIG c);	$c \leftarrow \ \mathrm{gcd}(a,\ b)$
DWORD	**BigInc** (HBIG a);	$a \leftarrow a + 1$
DWORD	**BigIntAdd** (HBIG a, DWORD i);	$a \leftarrow a + i$
DWORD	**BigIntDiv** (HBIG a, DWORD i, unsigned int *r);	$a \leftarrow \lfloor a/i \rfloor,\ r \leftarrow a \bmod i$
DWORD	**BigIntMov** (HBIG a, DWORD i);	$a \leftarrow i$
DWORD	**BigIntMul** (HBIG a, DWORD i);	$a \leftarrow a \cdot i$
DWORD	**BigIsPrime** (HBIG a);	
DWORD	**BigMov** (HBIG a, HBIG b);	$a \leftarrow b$
DWORD	**BigMul** (HBIG a, HBIG b);	$a \leftarrow a \cdot b$
DWORD	**BigMulMod** (HBIG a, HBIG b, HBIG m);	$a \leftarrow a \cdot b \bmod m$
DWORD	**BigNot** (HBIG a);	$a \leftarrow \ \mathrm{NOT}(a)$
DWORD	**BigOr** (HBIG a, HBIG b);	$a \leftarrow a$ OR b
DWORD	**BigPow** (HBIG a, DWORD e);	$a \leftarrow a^e$
DWORD	**BigPowMod** (HBIG a, HBIG e, HBIG m);	$a \leftarrow a^e \bmod m$
DWORD	**BigRndInit** (DWORD s);	Init the Randomgenerator with seed *s*
DWORD	**BigRnd** (HBIG a);	$a \leftarrow$ value at random

DWORD **BigRndRange** (HBIG a, HBIG l, HBIG h); $a \leftarrow$ value at random, $l \leq a \leq h$

DWORD **BigShl** (HBIG a, DWORD c); $a \leftarrow a \cdot 2^c$

DWORD **BigShr** (HBIG a, DWORD c); $a \leftarrow \lfloor a/2^c \rfloor$

DWORD **BigSub** (HBIG a, HBIG b); $a \leftarrow a - b$

DWORD **BigTest** (HBIG a, HBIG b);

DWORD **BigXor** (HBIG a, HBIG b); $a \leftarrow a$ XOR b

DWORD **BigZero** (HBIG a); $a \leftarrow 0$

3 BigLib - Return Values

The majority of the instructions returns a simple error indicator `BIG_ERROR` if the operation failed, otherwise `0`. Instructions with return values of different interpretation will be discussed here.

$$\textbf{BigCmp}\ (\text{HBIG a, HBIG b}) = \begin{cases} -1 & \text{if } a < b \\ 0 & \text{if } a = b \\ 1 & \text{if } a > b \end{cases}$$

$$\textbf{BigDec}\ (\text{HBIG a}) = \begin{cases} 0 & \text{if } a = 0 \text{ after decrease} \\ 1 & \text{else} \end{cases}$$

$$\textbf{BigInc}\ (\text{HBIG a}) = \begin{cases} 0 & \text{if } a = 0 \text{ after increase} \\ 1 & \text{else} \end{cases}$$

$$\textbf{BigIsPrime}\ (\text{HBIG a}) = \begin{cases} 0 & \text{if } a \text{ is not prime} \\ 1 & \text{if } a \text{ is probably prime} \end{cases}$$

$$\textbf{BigTest}\ (\text{HBIG a, HBIG b}) = \begin{cases} 0 & \text{if } (a \text{ AND } b) = 0 \\ 1 & \text{if } (a \text{ AND } b) \neq 0 \end{cases}$$

4 BigLib - Masm32 Example

```
; ---------------------------------------------------------------------------
; Demonstrates how to multiply two BigNums
; ---------------------------------------------------------------------------

; include Lib and headers

  includelib BigLib.lib
  include BigLib.inc

[...]

.DATA
  szDecString1 DB "12345678901234567890123456789012345678901234567890", 0
  szDecString2 DB "99999999999999999999999999999", 0

.DATA?
  hBig1 HBIG ? ; to save the BigNum handles
  hBig2 HBIG ?
  buffer DB 1024 DUP(?) ; to store converted result

.CODE

  [...]

  ; somewhere in the CODE:

  ; Create two 512Bit Bignums from Strings
  ; BigCreate( <<size>> , BIG_STRING, <<pointer to String>> , <<NumberBase of String>> )

    invoke BigCreate, BIG_512, BIG_STRING, offset szDecString1, 10
    mov hBig1, eax

    invoke BigCreate, BIG_512, BIG_STRING, offset szDecString2, 10
    mov hBig2, eax

  ; Big1 = Big1 * Big2
    invoke BigMul, hBig1, hBig2

  ; Result is stored in hBig1. Convert hBig1 back to alphadecimal string
    invoke BigToString, hBig1, offset buffer, 10

  ; destroy the BigNums to free memory
    invoke BigDestroy, hBig1
    invoke BigDestroy, hBig2

  ; Show result using MessageBox
    invoke MessageBox, 0, offset buffer, 0, MB_OK

  ; go on with something

  [...]
```

5 BigLib - C/C++ Example

```
#include "windows.h" // Because BigLib uses Windows-Memory Management
#include "BigLib.h"  // include BigLib Header

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow) {

    char charbuf[1024];  // to store converted alphanumerical string from BigNum

    // create BigNum initialized by integer 100000
    HBIG hBig1 = BigCreate(BIG_256, BIG_INT, 100000, 0);

    // create BigNum initialized by decimalstring
    HBIG hBig2 = BigCreate(BIG_256, BIG_STRING, (DWORD)"9999999", 10);

    // create BigNum initialized to zero
    HBIG hBig3 = BigCreate(BIG_256, BIG_ZERO,0,0);

    // Big1 = Big1 + Big2
    BigAdd(hBig1, hBig2);

    // Big3 = Big1
    BigMov(hBig3, hBig1);

    // convert Big3 to decimalstring
    BigToString(hBig3, charbuf);

    // free memory
    BigDestroy(hBig1);
    BigDestroy(hBig2);
    BigDestroy(hBig3);

    // show result of addition
    MessageBox(0, charbuf1, "Test", 65);

    return 0;
}
```