

# Instructions for asmlib

A multi-platform library of highly optimized functions for C and C++.

By Agner Fog. © 2008. GNU General Public License.  
Version 2.00. 2008-07-22.

## Contents

1	Introduction .....	1
1.1	Support for multiple platforms .....	2
1.2	Calling from other programming languages .....	2
1.3	Position-independent code.....	2
1.4	Overriding standard function libraries.....	2
1.5	Comparison with other function libraries .....	3
2	Standard library functions.....	4
2.1	memcpy .....	4
2.2	memmove.....	4
2.3	memset.....	5
2.4	strcat.....	5
2.5	strcpy.....	5
2.6	strlen.....	6
3	Random number generator functions .....	6
3.1	Mersenne Twister .....	6
3.2	Mother-of-all generator .....	7
4	Other functions.....	8
4.1	round .....	8
4.2	InstructionSet.....	8
4.3	ProcessorName .....	9
4.4	ReadTSC.....	9
4.5	Serialize.....	10
5	Library versions .....	10
6	File list.....	11
7	License conditions.....	12
8	No support.....	12

## 1 Introduction

Asmlib is a function library to call from C or C++ for all x86 and x86-64 platforms. It includes:

- Faster versions of several standard C functions.
- Efficient random number generators.
- Functions that are often missing in standard libraries: `round`, etc.
- Functions for identifying the microprocessor and for testing speed, etc.

These functions are written in assembly language for the sake of optimizing speed. Many of the functions have multiple branches for different instruction sets, such as SSE2, SSE3, etc. These functions will automatically detect which instruction set is supported by the computer it is running on and select the optimal branch.

The latest version of asmlib is always available at [www.agner.org/optimize](http://www.agner.org/optimize).

## 1.1 Support for multiple platforms

Different operating systems and compilers use different object file formats and different calling conventions. Asmlib is available in many different versions, supporting 32-bit and 64-bit Windows, Linux, BSD and Mac running Intel or AMD x86 and x86-64 family processors. The following object file formats are supported: OMF, COFF, ELF, Mach-O. Almost all C and C++ compilers for these platforms support at least one of these object file formats. Processors running different instruction sets, such as Itanium or Power-PC are not supported.

See page 10 for a list of asmlib versions for different platforms.

## 1.2 Calling from other programming languages

Asmlib is designed for calling from C and C++. Calling the library functions from other programming languages can be quite difficult. It is necessary to use dynamic linking (DLL) if the compiler doesn't support static linking or if the static link library is incompatible.

A DLL uses the `stdcall` calling convention by default. Not all the functions in asmlib have a `stdcall/DLL` version. See the description of each function.

Strings and arrays are represented differently in other programming languages. It is not possible to use string and memory functions in other programming languages unless there is a feature for linking with C. See the compiler manual for how to link with C code.

Linking with Java is particularly difficult. It is necessary to use the Java Native Interface (JNI).

## 1.3 Position-independent code

Shared objects (\*.so) in 32-bit Linux, BSD and Mac require position-independent code. Special position-independent versions of asmlib are available for building shared objects. Not all functions in asmlib are available in the position-independent versions. See the description of each function.

## 1.4 Overriding standard function libraries

The standard libraries that are included with common compilers are not always fully optimized and may not use the latest instruction set extensions (SSE2, SSE3, etc.). It is sometimes possible to improve the speed of a program simply by using a faster function library.

You may use a profiler to measure how much time a program spends in each function. If a significant amount of time is spent executing library functions then it may be possible to improve performance by using faster versions of these functions.

There are two ways to replace a standard function with a faster version:

1. Use a different name for the faster version of the function. For example call `A_memcpy` instead of `memcpy`. Asmlib have functions with `A_` prefix as replacements for several standard functions.
2. Asmlib is available in an "override" version that uses the same function names as the standard libraries. If two function libraries contain the same function name then the linker will take the function from the library that comes first.

If you use the "override" version of the asmlib library then you don't have to modify the program source code. All you have to do is to link the appropriate version of asmlib into your project. See page 10 for available versions of asmlib. If standard libraries are included explicitly in your project then make sure asmlib comes before the standard libraries.

The override method will replace not only the function calls you write in the source code, but also function calls generated implicitly by the compiler as well as calls from other libraries. For example, the compiler may call `memcpy` when copying a big object. The override version of asmlib accepts function names both with and without the `A_` prefix.

The override method sometimes fails to call the asmlib function because the compiler uses built-in inline codes for some common functions rather than calling a library. The built-in codes are not optimal on modern microprocessors. Use option `-fno-builtin` on the Gnu compiler or `/Oi-` on the Microsoft compiler to make sure the library functions are called.

The override method may fail if the standard library has multiple functions in the same module. If, for example, the standard library has the functions `strcpy` and `stricmp` in the same module, and your program uses both functions, then you cannot get `stricmp` from the standard library without also getting `strcpy`. The linker will then generate an error message saying that there are two definitions of the function `strcpy`.

If the override method fails or if you don't want to override the standard library then use the no-override version of asmlib and call the desired functions with the `A_` prefix.

## 1.5 Comparison with other function libraries

Test	Processor	Microsoft	CodeGear	Intel	Mac	Gnu 32-bit	Gnu 32-bit -fno-builtin	Gnu 64 bit -fno-builtin	Asmlib
<code>memcpy</code> 16kB aligned operands	Intel Core 2	0.12	0.18	0.12	0.11	0.18	0.18	0.18	0.11
<code>memcpy</code> 16kB unaligned op.	Intel Core 2	0.63	0.75	0.18	0.11	1.21	0.57	0.44	0.12
<code>memcpy</code> 16kB aligned operands	AMD Opteron K8	0.24	0.25	0.24	n.a.	1.00	0.25	0.28	0.22
<code>memcpy</code> 16kB unaligned op.	AMD Opteron K8	0.38	0.44	0.40	n.a.	1.00	0.35	0.29	0.28
<code>strlen</code> 128 bytes	Intel Core 2	0.77	0.89	0.40	0.30	4.5	0.82	0.59	0.27
<code>strlen</code> 128 bytes	AMD Opteron K8	1.09	1.25	1.61	n.a.	2.23	0.95	0.6	1.19

### Comparing performance of different function libraries.

Numbers in the table0 are core clock cycles per byte of data (low numbers mean good performance). Aligned operands means that source and destination both have addresses divisible by 16.

#### Library versions tested:

Microsoft Visual studio 2008, v. 9.0

CodeGear Borland bcc, v. 5.5

Mac: Darwin8 g++ v 4.0.1.

Gnu: Glibc v. 2.7, 2.8.

Asmlib: v. 2.00.

Intel C++ compiler, v. 10.1.020. Functions `_intel_fast_memcpy` and

`__intel_new_strlen` in library `libircmt.lib`. Function names are undocumented.

Asmlib uses 128-bit xmm registers for moving data. This works faster on processors with 128-bit internal data paths (e.g. Intel Core 2, AMD K10) than on older processors with 64-bit internal data paths (e.g. Intel Pentium 4, AMD Opteron K8). Tests on AMD K10 have not been made yet. See my manual [Optimizing software in C++](#) for a discussion of the different function libraries available.

## 2 Standard library functions

### 2.1 memcpy

#### C++ prototype

```
void * A_memcpy(void * dest, const void * src, size_t count);
```

#### Description

Fast implementation of the standard `memcpy` function. Copies `count` bytes from `src` to `dest`. It is the responsibility of the programmer to make sure `count` does not exceed the size in bytes of `dest`.

#### Uncached writes

This function can write either via the data cache or directly to memory. Writing to the cache is always faster, but it may be advantageous to write directly to memory when the size of the data block is very big, in order to avoid polluting the cache.

The integer `CacheBypassLimit` (defined as `extern size_t CacheBypassLimit;`) controls how this function writes data. If `count < CacheBypassLimit` then the write goes via the data cache. If `count > CacheBypassLimit+48` then the write does not use the cache. (Between these values it can use either method).

`CacheBypassLimit` is currently set to `0x8000 = 32 kbytes`. You may change this to a lower value if it is unlikely that the data written by this function will be used again soon. You may change `CacheBypassLimit` to a larger value if the data cache is big and it is likely that the data written will be used again soon.

`CacheBypassLimit` should not be smaller than half the size of the level-1 cache and not bigger than half the size of the level-2 cache of the microprocessor.

#### Versions included

Standard library override version: Yes

Position-independent version: Yes

Stdcall version: No

### 2.2 memmove

#### C++ prototype

```
void * A_memmove(void * dest, const void * src, size_t count);
```

#### Description

Fast implementation of the standard `memmove` function. Copies `count` bytes from `src` to `dest`. Allows overlap between `src` and `dest` by copying the last bytes first if `dest` overlaps the last part of `src`. It is the responsibility of the programmer to make sure `count` does not exceed the size in bytes of `dest`.

#### Uncached writes

See `A_memcpy` page 4.

#### Versions included

Standard library override version: Yes

Position-independent version: Yes

Stdcall version: No

### **2.3 memset**

#### C++ prototype

```
void * A_memset(void * dest, int c, size_t count);
```

#### Description

Fast implementation of the standard `memset` function. Inserts `count` copies of the lower byte of `c` into `dest`. It is the responsibility of the programmer to make sure `count` does not exceed the size in bytes of `dest`.

#### Uncached writes

See `A_memcpy` page 4.

#### Versions included

Standard library override version: Yes

Position-independent version: Yes

Stdcall version: No

### **2.4 strcat**

#### C++ prototype

```
char * strcat(char * dest, const char * src);
```

#### Description

Fast implementation of the standard `strcat` function. Concatenates two zero-terminated strings by inserting a copy of `src` after `dest` followed by a terminating zero. It is the responsibility of the programmer to make sure that `strlen(dest)+strlen(src)+1` does not exceed the size in bytes of the array containing `dest`.

#### Uncached writes

Extremely long strings can bypass the cache, see page 4.

#### Versions included

Standard library override version: Yes

Position-independent version: Yes

Stdcall version: No

### **2.5 strcpy**

#### C++ prototype

```
char * A_strcpy(char * dest, const char * src);
```

#### Description

Fast implementation of the standard `strcpy` function. Copies a zero-terminated string `src` into an array `dest` followed by a terminating zero. It is the responsibility of the programmer to make sure that `strlen(src)+1` does not exceed the size in bytes of the array `dest`.

#### Uncached writes

Extremely long strings can bypass the cache, see page 4.

#### Versions included

Standard library override version: Yes

Position-independent version: Yes

Stdcall version: No

## 2.6 strlen

#### C++ prototype

```
size_t strlen(const char * str);
```

#### Description

Fast implementation of the standard `strlen` function. Returns the length of a zero-terminated string `str`, not counting the terminating zero.

If `str` is an ASCII string then the return value is the number of characters. If `str` is UTF-8 encoded then the return value is the number of code bytes, not the number of Unicode characters.

#### Versions included

Standard library override version: Yes

Position-independent version: Yes

Stdcall version: No

## 3 Random number generator functions

### 3.1 Mersenne Twister

#### C++ prototypes

See the file `asmlibran.h`

#### Description

This is an excellent random number generator which is widely used for demanding applications. The random number generator must be initialized with an integer seed. It will generate the same sequence of random numbers when initialized again with the same seed. It will generate a different sequence when initialized with a different seed. The function `ReadTSC` may be used for generating the seed. Several versions of the Mersenne Twister are available in this library.

#### Thread-safe C++ version

Use this version for multi-threaded C++ code. Make one instance of the class `CRandomMersenneA` for each thread. This is the only version that can be used in 32-bit position-independent code. Member functions of class `CRandomMersenneA`:

`void RandomInit(uint32 seed)`: Initialize with seed.

`void RandomInitByArray(uint32 seeds[], int length)`: Initialize with multiple seeds. Sequences will be different if at least one of the integers in `seeds[length]` differ.

`int IRandom (int min, int max)`: Generate a random integer number with uniform distribution in the interval `[min,max]`. Slightly inaccurate.

`int IRandomX(int min, int max)`: Same. Distribution is exact.

`double Random()`: Generate a random floating point number with uniform distribution in the interval [0,1). The resolution is  $2^{-32}$ .  
`uint32 BRandom()`: Generate 32 random bits.

#### Single-threaded static version

No class object is needed. Can be used in C language. Cannot be used in 32-bit position-independent code. Functions:

```
void MersenneRandomInit(int seed);
void MersenneRandomInitByArray(uint32 seeds[], int length);
int MersenneIRandom (int min, int max);
int MersenneIRandomX(int min, int max);
double MersenneRandom();
uint32 MersenneBRandom();
```

#### Single-threaded DLL version

No class object is needed. Cannot be used in 32-bit position-independent code. Functions:

```
void __stdcall MersenneRandomInitD(int seed);
void __stdcall MersenneRandomInitByArrayD(uint32 seeds[],
int length);
int __stdcall MersenneIRandomD (int min, int max);
int __stdcall MersenneIRandomXD(int min, int max);
double __stdcall MersenneRandomD();
uint32 __stdcall MersenneBRandomD();
```

#### Alternatives

A full C++ implementation is available in [www.agner.org/random/randommc.zip](http://www.agner.org/random/randommc.zip). This can be used without any external function library.

## **3.2 Mother-of-all generator**

#### C++ prototypes

See the file `asmlibran.h`

#### Description

This random number generator is simpler than the Mersenne Twister, but still very good.

#### Thread-safe C++ version

Use this version for multi-threaded C++ code. Make one instance of the class `CRandomMotherA` for each thread. This is the only version that can be used in position-independent code. Member functions of class `CRandomMotherA`:

```
void RandomInit(uint32 seed): Initialize with seed.
int IRandom (int min, int max): Generate a random integer number with uniform
distribution in the interval [min,max]. Slightly inaccurate.
double Random(): Generate a random floating point number with uniform distribution in
the interval [0,1). The resolution is  $2^{-32}$ .
uint32 BRandom(): Generate 32 random bits.
```

#### Single-threaded static version

No class object is needed. Can be used in C language. Cannot be used in 32-bit position-independent code. Functions:

```
void MotherRandomInit(int seed);
int MotherIRandom (int min, int max); Output random integer
double MotherRandom();
uint32 MotherBRandom();
```

#### Single-threaded DLL version

No class object is needed. Cannot be used in 32-bit position-independent code. Functions:

```
void    DLL_STDCALL MotherRandomInitD(int seed);
int     DLL_STDCALL MotherIRandomD (int min, int max);
double  DLL_STDCALL MotherRandomD();
uint32  DLL_STDCALL MotherBRandomD();
```

#### Alternatives

A full C++ implementation is available in [www.agner.org/random/randommc.zip](http://www.agner.org/random/randommc.zip). This can be used without any external function library.

## 4 Other functions

### 4.1 round

#### C++ prototype

```
int Round(float x);
int Round(double x);
extern "C" int RoundF(float x);
extern "C" int RoundD(double x);
```

#### Description

Converts a floating point number to the nearest integer. When two integers are equally near, then the even integer is chosen (provided that the current rounding mode is set to default). This function does not check for overflow. The default way of converting floating point numbers to integers in C++ is truncation. Rounding is much faster than truncation in 32 bit mode when the SSE2 instruction set is not enabled.

#### Versions included

Position-independent versions: Yes  
Stdcall versions: No

#### Alternatives

Compilers with C99 support have the identical functions `lrint` and `lrintf`.  
Compilers with intrinsics support have `_mm_cvtsd_si32` and `_mm_cvt_ss2si` when SSE2 is enabled.

### 4.2 InstructionSet

#### C++ prototype

```
extern "C" int InstructionSet (void);
```

#### Description

This function detects which instructions are supported by the microprocessor and the operating system. The return value is also stored in a global variable named `IInstrSet`. If `IInstrSet` is not negative then `InstructionSet` has already been called and you don't need to call it again.

Return values:

Return value	Meaning
0	Use 80386 instruction set only
1 or above	MMX instructions supported
2 or above	conditional move and FCOMI supported
3 or above	SSE (XMM) supported by processor and enabled by Operating system
4 or above	SSE2 supported by processor and Operating system
5 or above	SSE3 supported by processor and Operating system
6 or above	Supplementary-SSE3 supported by processor and Operating system
8 or above	SSE4.1 supported by processor and Operating system

10 or above	SSE4.2 supported by processor and Operating system
-------------	--

The return value will always be 4 or above in 64-bit systems.

This function is intended to indicate only instructions that are supported by both Intel and AMD and instructions that might be supported by both vendors in the future. It is unknown at present (2008) whether future AMD processors will support the full instruction sets listed above. The missing values in the sequence are reserved for future AMD processors possibly supporting only part of the SSE4.1 and SSE4.2 instruction sets. It is certain, however, that Intel will not support the AMD SSE5 instruction set. SSE5 is therefore not included. The future AVX and FMA instruction sets will probably be added in a later version. This function may be modified in the future when more is known about whether AMD and Intel will go separate ways or there will be a meaningful common instruction set.

#### Versions included

Position-independent version: Yes

Stdcall version: Same version can be used.

### 4.3 ProcessorName

#### C++ prototype

```
extern "C" char * ProcessorName (void);
```

#### Description

Returns a pointer to a static zero-terminated ASCII string with a description of the microprocessor as returned by the CPUID instruction.

#### Versions included

Position-independent versions: Yes

Stdcall version: Same version can be used.

### 4.4 ReadTSC

#### C++ prototype

```
extern "C" uint64 ReadTSC (void);
```

#### Description

This function returns the value of the internal clock counter in the microprocessor. Execution is serialized before and after reading the time stamp counter in order to prevent out-of-order execution. Does not work on 80386 and 80486. A 32-bit value is returned if the compiler doesn't support 64-bit integers.

To count how many clock cycles a piece of code takes, call `ReadTSC` before and after the code to measure and calculate the difference.

You may see that the count varies a lot because you may not be able to prevent interrupts during the execution of your code. If the measurement is repeated then you will see that the code takes longer time the first time it is executed than the subsequent times because code and data are not cached at the first execution.

Time measurements with `ReadTSC()` may not be fully reproducible on Intel processors with SpeedStep technology (i.e. Core and later) because the clock frequency is variable.

`ReadTSC()` is also useful for generating a seed for a random number generator.

#### Versions included

Position-independent version: Yes

Stdcall version: Same version can be used.

## 4.5 Serialize

### C++ prototype

```
void Serialize(void);
```

### Description

Serializes execution. Useful before and after reading performance monitor counters with `__readpmc()` in order to prevent out-of-order execution.

### Versions included

Position-independent version: Yes

Stdcall version: Same version can be used.

## 5 Library versions

The asmlib library has many versions for compatibility with different platforms and compilers. Use the tables below to select the right version for a particular application.

Library version selection guide: Windows					
Compiler/language	File format	Override standard library	32 bit	64 bit	
MS C++ unmanaged, Intel, Gnu	COFF	yes	alibcof32o.lib	alibcof64o.lib	
		no	alibcof32.lib	alibcof64.lib	
Borland C++, Digital Mars, Watcom	OMF	yes	alibomf32o.lib		
		no	alibomf32.lib		
MS C++ .net, C#, VB	DLL	no	alibd32.dll	alibd64.dll	
Borland Delphi	DLL	no	alibd32.dll		
Other languages	DLL	no	alibd32.dll	alibd64.dll	

Library version selection guide: Linux and BSD (x86 and x86-64)					
Compiler/language	File format	Override standard library	32 bit executable	32 bit shared object	64 bit
Gnu, Intel C++	ELF	yes	alibelf32o.a	alibelf32op.a	alibelf64o.a
		no	alibelf32.a	alibelf32p.a	alibelf64.a

Library version selection guide: Mac (Intel based)					
Compiler/language	File format	Override standard library	32 bit executable	32 bit shared object	64 bit
Gnu, Intel C++	MachO	yes	alibmac32o.a	alibmac32op.a	alibmac64o.a
		no	alibmac32.a	alibmac32p.a	alibmac64.a

### Explanation of the column headings:

Compiler/language: The compiler and programming language used. Different compilers may use different object file formats.

File format: It is necessary to select a library in the right object file format, or a dynamic link library if static linking is not possible.

Override standard library: Libraries with suffix o use the same names for standard functions as the standard libraries. If this library is linked before the standard library then it will replace

the standard functions. Libraries without suffix o use different names for the standard functions.

32 bit / 64 bit: Use the appropriate version when compiling for 32-bit mode or 64-bit mode.

32 bit executable: Use this version when making an executable binary file.

32 bit shared object: Use this version when position-independent code is needed. Position-independent code is needed when building a shared object for 32-bit mode, but it is slightly slower.

## 6 File list

### Files in asmlib.zip

asmlib-instructions.pdf	This file
asmlib.h	C/C++ Header file for asmlib functions
asmlibran.h	C/C++ Header file for asmlib random number generators
alibelf32.a	Library 32-bit ELF format
alibelf32o.a	Library 32-bit ELF format, override standard library
alibelf32op.a	Library 32-bit ELF format, override, position-independent
alibelf32p.a	Library 32-bit ELF format, position-independent
alibelf64.a	Library 64-bit ELF format
alibelf64o.a	Library 64-bit ELF format, override standard library
alibmac32.a	Library 32-bit Mach-O format
alibmac32o.a	Library 32-bit Mach-O format, override standard library
alibmac32op.a	Library 32-bit Mach-O format, override, position-independent
alibmac32p.a	Library 32-bit Mach-O format, position-independent
alibmac64.a	Library 64-bit Mach-O format
alibmac64o.a	Library 64-bit Mach-O format, override standard library
alibd32.dll	Library 32-bit Windows DLL
alibd32.lib	Import library for alibd32.dll
alibd64.dll	Library 64-bit Windows DLL
alibd64.lib	Import library for alibd64.dll
alibcof32.lib	Library 32-bit COFF format
alibcof32o.lib	Library 32-bit COFF format, override standard library
alibcof64.lib	Library 64-bit COFF format
alibcof64o.lib	Library 64-bit COFF format, override standard library
alibomf32.lib	Library, 32-bit OMF format
alibomf32o.lib	Library, 32-bit OMF format, override standard library
asmlibSrc.zip	Source code

### Files in asmlibSrc.zip

alibd32.asm	Source code for DLL entry
alibd64.asm	Source code for DLL entry
instrset32.asm	Source code for InstructionSet function
instrset64.asm	Source code for InstructionSet function
memcpy32.asm	Source code for memcpy function
memcpy64.asm	Source code for memcpy function
memmove32.asm	Source code for memmove function
memmove64.asm	Source code for memmove function
memset32.asm	Source code for memset function
memset64.asm	Source code for memset function
mersenne32.asm	Source code for Mersenne Twister random number generator

mersenne64.asm	Source code for Mersenne Twister random number generator
mother32.asm	Source code for Mother-of-all random number generator
mother64.asm	Source code for Mother-of-all random number generator
procname32.asm	Source code for ProcessorName function
procname64.asm	Source code for ProcessorName function
rdtsc32.asm	Source code for ReadTSC function
rdtsc64.asm	Source code for ReadTSC function
round32.asm	Source code for Round functions
round64.asm	Source code for Round functions
serialize32.asm	Source code for Serialize function
serialize64.asm	Source code for Serialize function
strcat32.asm	Source code for strcat function
strcat64.asm	Source code for strcat function
strcpy32.asm	Source code for strcpy function
strcpy64.asm	Source code for strcpy function
strlen32.asm	Source code for strlen function
strlen64.asm	Source code for strlen function
randomah.asi	Include file for mersenne32.asm and mersenne64.asm
testalib.cpp	Test example
alibd32.def	Exports definition function for alibd32.dll
alibd64.def	Exports definition function for alibd64.dll
alibd32.exp	Exports definition function for alibd32.dll
alibd64.exp	Exports definition function for alibd64.dll
MakeAsmlib.bat	Batch file for making asmlib
asmlib.make	Makefile for making asmlib

## 7 License conditions

All versions of this function library and its source code are copyrighted © 2008 by Agner Fog. The software may be copied and used freely under the conditions of the [GNU General Public License](#) with the additional ethical restriction that the random number generators shall not be used for any gambling application. A gambling application is understood in this context as any game that allows players to win and loose money or convertible valuables and where randomness plays a significant role, e.g. a poker machine.

Commercial licenses are available on request.

## 8 No support

Note that asmlib is a free library provided without warranty or support. This library is for experts only, and it may not be compatible with all compilers and linkers. If you have problems using it, then don't.

I am sorry that I don't have the time and resources to provide support for this library. If you ask me to help with your programming problems then you will not get any answer.