5.
Calling conventions
for different C++ compilers and operating systems

By Agner Fog. Technical University of Denmark.

Contents

1 Introduction

This is the fifth in a series of five manuals:

1. Optimizing software in C++: An optimization guide for Windows, Linux and Mac platforms.

2. Optimizing subroutines in assembly language: An optimization guide for x86 platforms.

3. The microarchitecture of Intel, AMD and VIA CPUs: An optimization guide for assembly programmers and compiler makers.

4. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs.

5. Calling conventions for different C++ compilers and operating systems.

The latest versions of these manuals are always available from www.agner.org/optimize. Copyright conditions are listed on page 57 below.

The present manual describes differences between various C++ compilers that affect binary compatibility, such as data storage, function calling conventions, and name mangling. The function calling methods, name mangling schemes, etc. are described in detail for each compiler.

The purposes of publishing this information are:

- Point out incompatibilities between compilers.

- Make new compilers compatible with old ones.

- Solve compatibility problems between function libraries produced by different compilers.

- Facilitate linking different programming languages together.

- Facilitate the making of assembly subroutines that are compatible with multiple compilers and multiple operating systems.

- Solve compatibility problems for data stored in binary files.

- Facilitate the construction of debugging, profiling and disassembly tools.

- Facilitate the construction of object file conversion utilities.

- Provoke compiler vendors to use open standards.

- Inspire future standardization.

Hardware platforms covered:

- x86 microprocessors with 16 bit, 32 bit and 64 bit architectures from Intel, AMD, VIA and possibly other vendors.

The IA64 architecture, which is implemented in Intel's Itanium processor, is not compatible with the x86 architecture, and is not covered in this report.

Operating systems covered:

- DOS, 16 bit.
- Windows, 16 bit, 32 bit and 64 bit.
- Linux, 32 bit and 64 bit.
- FreeBSD etc. 32 bit and 64 bit.
- Mac OS X, Intel based, 32 bit and 64 bit.

C++ compilers tested:

- Borland, 16 bit v. 3.0 and 5.0
- Microsoft, 16 bit, v. 8.0
- Watcom, 16 bit v. 1.2
- Borland 32 bit v. 5.0
- Microsoft, 32 bit, v. 9.0 and 13.10
- Gnu, 32 bit, v. 2.95, 3.3.3, 4.1.0 and several other versions under Linux, FreeBSD and Windows.
- Watcom, 32 bit, v. 1.2
- Symantec, 32 bit, v. 7.5
- Digital Mars, 32 bit, v. 8.3.8
- Codeplay VectorC, 32bit, v. 2.1.7
- Intel, 32 bit for Windows and Linux, v. 8.1 and 9.1
- Microsoft, 64 bit, v. 14.00
- Gnu, 64 bit, v. 3.3.3 and 4.1.0 (Linux and FreeBSD)
- Intel, 64 bit for Windows and Linux, v. 8.1 and 9.1

This document provides information that is typically difficult to find. The documentation of calling conventions and binary interfaces of compilers and operating systems is often shamefully poor and sometimes completely absent. Name mangling schemes are rarely documented. For example, it is stated in Microsoft Knowledge Base that "Microsoft does not publish the algorithm its compilers use for name decoration because it may change in the future." (article number Q126845). However, the name mangling scheme has not changed much from the old 16-bit compiler to the newest 32-bit and 64-bit compilers, and it is unlikely to be changed in the future because of compatibility requirements.

As most of the information given here is based on my own experiments, it is obviously not authoritative, and it is not guaranteed to be accurate or complete. This document tells how things are, not how they are supposed to be. Some details appear to be the haphazard consequences of how compilers happen to be implemented rather than results of careful planning. Calling "conventions" may not be the most appropriate term in this case, but it may be necessary to copy the quirks of existing compilers when full compatibility is desired.

I have no knowledge about whether any information provided here is protected by patents or other legal restrictions, but I have found no specific patent markings on the compilers.

I have gathered this information mainly by converting C++ code to assembly. All the compilers I have tested are capable of converting C++ to assembly, either directly or via object files. The reader is encouraged to do your own research, if necessary, to get additional information needed or to clarify any questions you may have. The easiest way of doing this research is to make the compiler convert a C++ test file to assembly. Other possible methods are to use object file dump utilities, disassembly utilities, or provoke error messages from a linker. If you find any errors in this document then please let me know.

Please note that I don't have the time and resources to help people with their programming problems. If you Email me with such questions, you will not get any answer. You may send your questions to appropriate internet forums instead.

2 The need for standardization

In the days of the old DOS operating system, it was often possible to combine development tools from different vendors with few compatibility problems. With 32-bit Windows, the situation has gone completely out of hand. Different compilers use different data representations, different function calling conventions, and different object file formats. While static link libraries have traditionally been considered compiler-specific, the widespread use of dynamic link libraries (DLL's) has made the distribution of function libraries in binary form more common. Unfortunately, the standardization of data representation and calling conventions that would make DLL's compatible is still lacking.

In the Linux, BSD and Mac operating systems, there are fewer compatibility problems because a more or less official standard is defined. Most of this standard is followed by Gnu C++ version 3.x and later. Earlier versions of the Gnu compiler are not compatible with this.

Fortunately, there is a growing recognition of the need for standardization of application binary interfaces (ABI's). The ABI's for the new 64-bit operating systems are specified in much more detail than we have seen in older operating systems. However, some of these ABI's still lack specification of name mangling schemes and other details. Traditionally, compiler vendors have not published or standardized their name mangling schemes. A common excuse was that the object files would not be compatible anyway because of differences in data formats and calling conventions. Now that data formats and calling conventions are specified in the ABI's, there is no excuse any more for not publishing and standardizing name mangling schemes as well. It is my hope that this document will be a contribution towards this end.

Compilers and other development tools is an area where *de facto* standards play an important role. Almost all compilers for UNIX-like x86 platforms are designed to be compatible with the Gnu compiler. And the calling "conventions" of the Microsoft compiler has almost become a *de facto* standard for the Windows operating system. The C++ compilers from Intel, Symantec, Digital Mars and Codeplay are all designed to be binary compatible with Microsoft's C++ compiler, despite the fact that Microsoft has refused to publish important details. At least some of these compiler makers have relied on reverse engineering for obtaining the necessary information. There is a pressing need for publishing the relevant standards, and the present document is my contribution towards this end.

It is highly recommended that designers of development tools follow all available standards. Where no official standard exists, use an existing compiler for reference. Use the Microsoft compiler as a reference for Windows systems and the Gnu compiler as a reference for UNIX-like systems. For features that are not supported by these compilers, use the Intel compiler for reference. The calling conventions of these compilers may be considered *de facto* standards for Windows and UNIX platforms.