Porting Applications from Windows NT for Intel to Windows NT for Alpha

Product Version: Windows NT Version 4.0
Document version: 2.0
September 10, 1997

This guide describes how to port Windows NT Applications on the Intel Platform to Alpha NT. Issues of interest to Windows NT Version 4.0 developers, such as compiler descriptions, porting methodologies, and how to optimize for Alpha are discussed.

Digital Equipment Corporation
Maynard, Massachusetts

TABLE OF CONTENTS

About This Document

This porting guide provides information about porting applications from Windows NT Version 4.0 on Intel systems to Windows NT Version 4.0 on Alpha . The document gives information about software that is part of the Windows NT operating system and about the Alpha specific compilers and performance tools.

Revision History

Version 1.0 October 5, 1994, porting applications from Intel NT 3.51 to Alpha NT 3.51
Version 2.0 September 10, 1997, porting applications from Intel NT 4.0 to Alpha NT 4.0

Audience

This document is written for software developers who have developed Windows NT application(s) on Intel and would like to port their application(s) to Alpha .

Organization

This manual contains the following Chapters and Appendices

Chapter 1. Introduction
Provides an overview of the Alpha Windows NT platform and assesses requirements for porting Intel applications.

Chapter 2. Compiler Descriptions
Provides an overview of the C/C++ Compiler, DEC FORTRAN Compilers and the Assembler for Alpha.

Chapter 3. Porting Overview
Outlines the steps required to port your application to Windows NT Version 4.0 of Alpha.

Chapter 4. Architectural Issues
Itemizes the differences between the two machines to be considered in porting.

Chapter 5. Alpha Application Tuning and Optimization
Describes features for tuning applications on Alpha NT.

Chapter 6. Troubleshooting C/C++ Compiler and Link Problems
Provides techniques for resolving compiler and link problems.

Chapter 7. FX!32 Binary Translator
Describes the capabilities of Digital's Intel binary translation software.

Chapter 8. Spike
Describes Digital's optimizer for Alpha/NT executables.

1. Introduction

This chapter provides an overview of porting applications from Windows NT Version 4.0 on the Intel NT platform to Windows NT Version 4.0 on the Alpha platform. This chapter briefly describes the Alpha architecture and then helps you assess the effort involved in porting your application.

1.1 Windows NT 4.0 for Alpha

The Alpha processor is a high-performance 64-bit, operating system neutral, architecture developed by Digital Equipment Corporation. It has a 64-bit physical and virtual address space and processes 64-bit integers and floating-point numbers.

The Alpha architecture is scaleable and supports a variety of implementations in a range of workstation and server products. Alpha based platforms have consistently provided leading performance across industry standard as well as a large range of application benchmarks.

Alpha based platforms perform well due to the combination of high clock speeds, optimizing compilers, multiple instructions being issued every clock cycle and an optimized cache hierarchy.

Alpha based platforms running Windows NT are available from Digital and from other manufacturers exploiting the Alpha processor. The platforms are available as Servers, SMP Servers and Workstation configurations. Because Alpha based platforms running Windows/NT provide excellent performance and support little endian byte ordering (same as Intel based platforms), they interoperate very efficiently with Intel systems.

Windows NT is a Microsoft operating system which has been ported to and optimized for the Alpha architecture through collaborative engineering work between Digital and Microsoft. It is one of three operating systems (the others being Digital UNIX and OpenVMS) currently shipping on the Alpha platform.

Windows NT is a modern, 32-bit, multi-tasking operating system that exploits some advanced features of the Alpha architecture -- like the 64-bit registers. The Alpha version of Windows NT supports a 32-bit virtual address space, 32-bit pointer and long integer data types which is the same as on the Intel implementation.

All Windows NT platforms use identical IEEE floating point formats. On the Alpha platform, floating point operations can be carried out in either single precision (32-bits) or double precision (64-bits). On the Intel platform, temporary floating-point values have an 80-bit precision by default.

1.2 Execution of Intel Executables on Alpha

The Alpha version of Windows NT 4.0 provides transparent translation of 486 instructions. This translation is built into the operating system. It allows legacy Intel DOS, Intel WIN16 executables and Intel WIN16 enhanced mode applications to run on the Alpha platform. These applications run slower than native applications for many reasons, but for personal productivity tools, mail and many other less intensive applications, performance will be very satisfactory.

Also in Windows NT Version 4.0, OLE 2.01 allows enhanced interoperability between WIN16 and WIN32 applications. Further, each WIN16 application can run in its own address space, improving system robustness. Performance of these Intel executables on the Alpha platform is significantly improved.

The features described above are important to your clients. Your clients will want to interoperate between native, high performance WIN32 Alpha applications and legacy, Intel WIN16 executables, lower performance applications, and personal productivity tools. Developers may find some value in this feature, and your support staff needs to understand how WIN16 and WIN32 applications interoperate and share data on both Intel and Alpha platforms.

1.3 FX!32

In contrast to the built in emulation for Intel DOS and WIN16 executables on Windows NT 4.0 for Alpha, Intel WIN32 applications will not execute. An attempt to execute an Intel binary on the Alpha will cause the operating system to report an error similar to the one in **Error! Reference source not found.**.

Figure 1-1: Incorrect Platform Error Message

If you are unable to port your Intel WIN32 application to the AXP platform, you may still be able to run it using Digital FX!32. FX!32 is a means for providing fast and transparent execution of Intel WIN32 applications on the Alpha platform. It emulates and translates the Intel instruction set to the Alpha based RISC instructions. See Chapter 7 for further details on FX!32.

1.4 Porting Intel Applications to Windows NT for Alpha

In order to provide the fastest implementation of an application, the recompilation on the Alpha platform of an Intel WIN32 application is the proper means for providing this. A recompilation is usually very straightforward. By following the guidelines in the next section for writing portable code, one can recompile large amounts of code between the Intel and Alpha platforms without experiencing errors or platform penalties. Some

developers have been known to port significant Windows NT applications in less than one working day.

Many popular CASE tools and compilers, including Visual C++, are supported on both the Intel and Alpha Windows NT platforms. These common tools ease migration and allow Windows NT applications to be built and deployed in a very similar manner on both platforms. For instance, an application built using the Intel version of Visual C++ can be easily ported to the Alpha version of Visual C++ with very few changes.

1.5 Assessing the Porting Effort

As long as an application doesn't make assumptions about the underlying hardware or how arguments are located on the stack or in memory, then the number of Alpha related porting issues will be minimal. Here are some features specific to the Alpha which developers may need to address.

- Application data should be naturally aligned wherever possible. This avoids the additional processing required by the operating system to fix up unaligned data operations. Naturally aligned data will improve the performance and quality of an application. See Chapter 5.3 for details on handling data alignment issues.

- The Alpha handles finite floating-point values in the same way as the Intel platform but handles nonfinite floating-point values differently. Imprecise floating-point exceptions are raised when nonfinite floating-point values are used or when division by zero or overflow occurs. These exceptions should be addressed to avoid application performance degradation. See Chapters 4.10 and 5.1.6 for details on handling floating point exceptions.

- The code generation switches that are specific to Intel need to be either removed or replaced with those for the Alpha platform. See Chapter 3.1.2 and Appendix A for details.

- Applications that make assumptions about how arguments are stored in memory will cause incompatibility problems. The stdarg.h standards should be used for the application to be portable.

2. Compiler Descriptions

This chapter first describes the Microsoft Visual C++ Developer Studio for the Alpha platform. As of this writing, Visual C++ 5.0 for Alpha is the latest compiler available from Microsoft. This chapter also mentions additional compilers that are available for purchase on Alpha for Windows NT.

The Alpha compiler available in the Visual C++ Developer Studio contains the latest compiler technology from Digital. The compiler that used to be available through the older versions of the SDK, CLAXP.EXE, is now considered obsolete and should not be used. However, the same command line nmake development environment that was available through the SDK is available with Visual C++. Therefore, you are not required to use the Visual C++ Workbench environment to develop or port an application.

2.1 C/C++ Compiler

The compiler available in the Visual C++ Developer Studio is Digital and Microsoft's compiler for Alpha on Windows NT. It generates Alpha specific optimized machine code. The compiler front-end, CL1.EXE, accepts and parses the compiler switches. The back-end, CL2.EXE, is Digital's code generator and optimizer, or what is referred to as GEM. GEM is Digital's multi-language compiler technology that underlies several compilers which ship under Digital's NT, VMS and UNIX operating systems. It generates code optimized specifically for the Alpha processor.

The C/C++ Compiler produces Common Object File Format (COFF) object (.OBJ) files. The linker produces executable (.EXE) files or dynamic-link libraries (DLLs).

Note that the C/C++ Compiler only supports the development of 32-bit applications for Windows NT on Alpha platforms. However 16-bit application porting could be achieved using other available compilers and with Microsoft's "PortTool". PortTool is included in the Microsoft Resource Kit.

2.1.1 Microsoft Visual C++ Developer Studio

The Visual C++ Developer Studio is a universal development environment that operates equally well on both the Alpha and Intel platforms. The look and feel of the development environments for both platforms are virtually the same.

An Intel port to the Alpha requires that you first copy all project files to the system, open the project file to add new Alpha configurations to it, and then verify that the project options were carried over. This may not always be the case since not all project options are portable between systems. See Chapter 3.2 for more detailed information on porting using the Visual C++ Workbench for Alpha.

Most of the compiler switches are available via the C/C++ Tab of the Project Settings dialog box. Each of the options on the C/C++ Tab are described in detail in the online documentation under Setting Compiler Options. The description of each option includes the name of the equivalent command-line option.

2.1.2 Microsoft Foundation Class Library (MFC)

The Microsoft Foundation Library (MFC) enables C++ programmers to write applications for Microsoft windows. The class library gives a complete "application framework" which defines an architecture for integrating the user interface of an application for Windows with the rest of the application.

An MFC application is easily portable on all NT platforms, because these classes are available across all the available NT platforms

2.2 DEC FORTRAN

DEC FORTRAN for Alpha is an implementation of full language FORTRAN-77 conforming to American National Standard FORTRAN, ANSI X3.9-1978. As such, DEC FORTRAN will compile any source code pool that strictly adheres to ANSI FORTRAN-77.

The compiler also supports extensions to the ANSI standard, including a number of extensions defined by the DEC FORTRAN compiler that run on OpenVMS, Digital UNIX, and ULTRIX systems. The following is some of the more significant extensions:

- Additional statements such as DO WHILE, ENDDO, ACCEPT, and TYPE
- Support for dynamic memory allocation and the POINTER data type.
- Support for reading and writing binary data files in big-endian VAX, IBM[R], and CRAY[R] floating point formats.
- Support for 64-bit signed integers using INTEGER*8 and LOGICAL*8.

Other vendor FORTRAN compilers' extensions may differ. However, since Digital's FORTRAN extensions through the years have often become de facto standards, compatibility is likely even when syntax extensions are used.

Certain FORTRAN extensions specific to Microsoft FORTRAN that are not yet supported by DEC FORTRAN are dynamically allocated arrays, SELECT CASE, INTERFACE TO, CYCLE, and EXIT statements. (All of these except INTERFACE TO will be supported in the next major release of DEC FORTRAN.)

In addition to language extensions, the DEC FORTRAN runtime library provides a number of built-in utility routines in addition to the ANSI-defined intrinsic functions. Other compilers are likely to differ in what utility routines are available.

The development kits provided with DEC FORTRAN supports a command line interface and the *nmake* utility. Source code debugging with a graphical user interface is provided via *windbg*.

DEC FORTRAN uses the GEM compiler as its backend on all Alpha platforms. The DEC FORTRAN compiler provides a multi-phased optimizer that is capable of performing optimizations across entire programs. Although builds may take more time and memory compared to compilers that optimize less thoroughly, the improved performance of highly optimized code at run-time is worth the added time.

DEC FORTRAN is run-time compatible with the C/C++ Compiler. This capability allows you to mix-and-match FORTRAN and C/C++ modules to meet your application needs.

Run-time compatibility with other compilers have not been tested, though it may work if the proper calling standard is followed.

The DEC FORTRAN kit includes dynamic-link (DLL) versions of its run-time Libraries RTLs). These RTLs can be reproduced and distributed royalty-free worldwide.

2.3 Digital Visual FORTRAN

DIGITAL Visual FORTRAN for Windows® 95 and Windows NT ® (currently on Intel®-based systems) combines robust FORTRAN compiler technology from DIGITAL with the leading Developer Studio IDE from Microsoft®. Both the compiler and the development environment have been updated.

The compiler supports the latest ISO/ANSI draft FORTRAN 95 language features in addition to most popular VAX and Microsoft FORTRAN PowerStation extensions.

DIGITAL Visual FORTRAN 5.0 is compatible with the Microsoft Visual C++® development system version 5.0 so that mixed language programming can be done from a common development environment which includes an editor, debugger, resource editor, incremental linker, profiler, browser, and project manager as well as extensive on-line documentation.

An Alpha version of Digital Visual FORTRAN will be available during the late summer of 1997.

2.4 Windows NT Assembler (AS)

The Alpha version of Visual C++ contains an assembler (ASAXP.EXE) that converts assembly language statements into Alpha machine code. Although not portable, experienced Alpha assembly language programmers can use this where required. Refer to the Alpha Architecture Reference Manual, 2nd Edition, published by Digital Press and available through Digital Equipment Corporation, ISBN 1-55558-145-5 for information regarding the Alpha instruction set.

3. Porting Overview

In most cases, porting your application simply means re-compiling and re-linking the application on Alpha. For example, if you use the Visual C++ Developer Studio to develop your application, then only minor changes to the environment are required. If you used another build environment, be sure an Alpha version of it is available. If there is not a version for the Alpha then the changes required may be more extensive.

This chapter describes how to port your application to the Alpha platform using both the nmake and Visual C++ development environments.

Section 3.1 deals with those changes required when porting a makefile. Section 3.2 describes those changes necessary when using the Visual C++ development environment.

3.1 Porting the nmake Environment

If you are using nmake, then your build environment will require very few changes to function correctly on Alpha. These changes require either removing or replacing unsupported command line options with the Alpha equivalents as described here.

3.1.1 Testing Processor Type in Makefiles

Windows NT defines the environment variable PROCESSOR_ARCHITECTURE on all platforms. You should use PROCESSOR_ARCHITECTURE in your makefile to conditionally compile code. On Intel platforms, this string is defined as “x86”. On Alpha platforms, this string is defined as “ALPHA”.

You can use the !IF and !ELSEIF conditionals in your makefiles to execute different actions depending on the value of the PROCESSOR_ARCHITECTURE string. For example:

```
!IF “$(PROCESSOR_ARCHITECTURE)”  == “ALPHA”
# Execute Alpha specific action
!ELSEIF “(PROCESSOR_ARCHITECTURE)” == “x86”
# Execute Intel specific action
!ELSE
# Die with error message
!ENDIF
```

3.1.2 Modify Compiler Switches

Most of the compiler switches are the same on both platforms. However, the Intel compiler supports some switches that the Alpha version does not. The majority of Intel switches are ignored by the Alpha C/C++ compiler, however since some are not, all Intel related switches should be removed from you makefiles. Use the environment variable PROCESSOR_ARCHITECTURE where necessary to distinguish between Alpha and Intel builds.

Some of the more widely used switches are described in detail in Chapter 5 and in Appendix A.

3.1.3 Other Build Environments

If you use other build environments and your build software is not ported to Alpha, you have to rebuild your dependency files. The `nmake` compiler, linker and resource macros are predefined in a file named `\MSTOOLS\INCLUDE\NTWIN32.MAK` in the Microsoft Win32 SDK. Use this file as a start for rebuilding your application.

The section, building Win32-Based Applications in the Getting Started Guide of the on-line Win32 SDK documentation provides further information about the build process.

3.1.4 Re-Compiling

Once all the source files have been copied to the Alpha platform and the necessary changes have been made to your makefile, executing `nmake` will compile your application. It is also possible to compile the application directly from the command line by using the C/C++ Compiler.

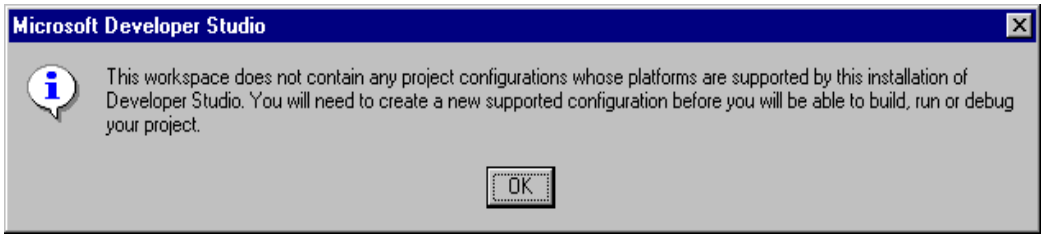
3.1.5 Re-Linking

The `link` command on Alpha is the same as Intel. The command understands the same flags.

3.2 Porting to the Alpha Visual C++ Environment

One can easily port projects from the Intel platform to the Alpha platform. You can use the same project and source files, but not the intermediate files. These have to be rebuilt.

In addition to copying the project files to the Alpha platform, you will need to create new Alpha project configurations. You will see that you are required to do this when opening an Intel based project. The window error message that will be reported is:

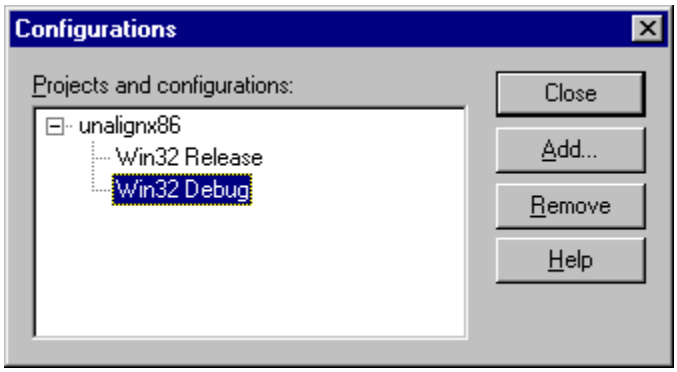


In addition to creating project configurations, you will also need to check the project options to make sure they were carried over. Because the Alpha edition of the Visual C++ does not support all of the exact same set of switches, the defaults are assumed.

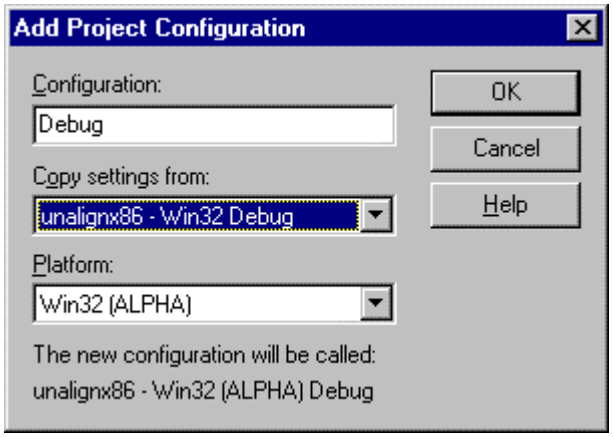
To convert an Intel project to an Alpha project perform the following steps:

1. Copy all source and configuration files directly related to the project to the new Alpha platform.
2. On the Alpha platform, click Open Workspace from the File menu. The Open Project Workspace dialog box opens.
3. Select your Intel project and click Open.

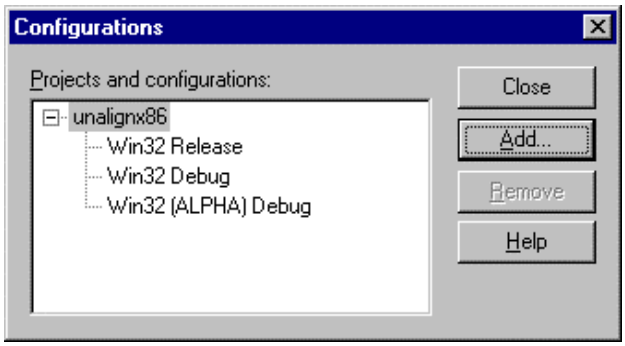
4. From the Build menu, click Configurations. The Configurations dialog box appears. Notice that the Intel-based configuration projects are listed. Here is an example.



5. Then click Add. The Add Project Configuration dialog box appears.
6. In the Platforms drop-down list, select the Alpha platform. Only those platforms for which you have a cross-development package appear in the Platforms list. For example:



7. Note the name given to this new configuration after clicking OK.



An Alpha-based project with the same build settings as your Intel-based project is automatically created. At this point you can set the newly created configuration to be the active project and start building on the Alpha.

3.3 Common Porting Issues

The issues described in this section are common to both the nmake and Visual C++ development environments.

3.3.1 Conditional Compiling

For the conditional compiling of source code, any one of the following macros can be used to identify Alpha as the processor for which the compiler is generating code. Use the `#ifdef` directive to accomplish this.

```
_M_ALPHA
_ALPHA_
__alpha
__ALPHA
```

The preferred macro is `_M_ALPHA` because it is consistent with the Intel version of Visual C++.

The `_ALPHA_` symbol defines the Alpha target environment. The make file should pass `-D_ALPHA_= 1` to the C/C++ Compiler. If the make file includes <ntwin32.mak>, this is automatically ensured.

3.4 Calling Conventions

The Alpha edition of the compiler uses a different calling convention than does the Intel edition. The Alpha calling convention is a register-based convention that frequently uses the Alpha argument registers (six integer registers and six floating-point registers) in cases where the Intel edition of the compiler would place all arguments on a stack frame. Consequently, code that assumes that an argument is on the stack is likely to cause problems.

For more information on the Alpha calling convention, see the Windows NT for Alpha Calling Standard. It is included in Books Online with Visual C++, the Alpha Architecture Reference Manual, 2nd Edition, published by Digital Press, available through Digital Equipment Corporation, ISBN 1-55558-145-5 and also available through Digital's ASAP web site at http://www.partner.digital.com/www-swdev/pages/Home/TECH/Ntfocus.html.

3.5 Debugging

There are several debuggers currently available for the Alpha to help you debug code at the application and kernel levels. They exist in the Win32 SDK and Visual C++ Developer Studio. The ones most commonly used are WinDbg, NTSD, KD and the Visual C++ debugger. A brief introduction to each one is described in the following sections.

For productive use of these debuggers, you should disable code optimization while compiling. This will produce a symbolic executable so that line numbers in the source files match line numbers being executed. Because of the code scheduling on the Alpha that takes place when code is optimized, the instructions can have a very scrambled

order. Use the /FAcs compiler switch to make an annotated .COD file, which can help you debug optimized code.

All of these debuggers work essentially the same on both platforms. The areas that a user will see differences are the viewing of the register set and the disassembly of the instructions. The Alpha has 32 general integer registers, each of which is 64 bits wide, and 32 floating-point registers, each of which is 64 bits wide. The RISC based Alpha instructions are 32 bits and of course differ greatly from the CISC architecture on the Intel platform.

3.5.1 WinDbg

The WinDbg debugger is a powerful, graphical tool that allows you to debug programs and edit source code. You can use WinDbg to debug most applications, user-mode drivers and kernel-mode drivers. It allows you to set breakpoints, and examine values of local variables, registers, assembly-language instructions and much more. WinDbg resides in the Win32 SDK and gets loaded into the \mstools\bin directory.

3.5.2 NT Symbolic Debugger (NTSD)

Using NTSD, you can display and execute program code, set breakpoints, and examine and change values in memory. Because NTSD can access memory locations through addresses, global symbols, or line numbers, you can refer to data and instructions by name rather than by address, making it easy to locate and debug specific sections of code. You can debug C programs at source-file level as well as at machine level. You can display source statements, disassembled machine code, or a combination of both. NTSD supports debugging multiple threads and processes. It is extensible, and can read and write both paged and nonpaged memory.

There are two commands (rL, rF) available on Windows NT for that are not available on Intel. These commands enable you to examine large integers and floating point registers.

3.5.3 Kernel Debugger (KD)

KD allows you to debug kernel-mode executables and device drivers. It can also be used to perform remote driver debugging between different architectures. For example, you can debug an Alpha driver from an Intel platform and vice versa. It is intended for use by system developers and those developing device drivers.

KD also supports multiprocessor debugging. Because it is deficient in setting breakpoints in user-mode code and cannot be used to examine or deposit paged-out memory, KD is not well suited to debugging user-mode programs. KD does not provide support for threads. KD runs on a host computer and communicates through a null-modem serial cable with the target computer running Windows NT. It is a requirement that the host computer be running Windows NT.

3.5.4 Visual C++ Debugger

The Visual C++ Developer Studio provides an integrated debugger to help locate bugs in an executable program, dynamic-link library (DLL), thread, or OLE and Active X components. The debugger interface provides special menus, windows, dialog boxes,

and spreadsheet fields. Drag-and-drop functionality is available for moving debug information between components.

4. Architectural Issues

This section discusses the architectural issues that you need to consider when porting applications from Windows NT for Intel to Windows NT for Alpha. The following section is a brief description of these differences. For more detailed information see the RISC Programmer's Guide which is a document that describes in detail the porting considerations that need to be addressed. The document is located in the MSDN library CD-ROM. The file /MSTOOLS/AS.HLR is also a good reference for compiler specific features.

4.1 Variable Argument Lists

For defining variable argument lists, the C/C++ Compiler supports two header files:

- ANSI standards <stdarg.h>
- traditional <varargs.h>

Ideally, one should use stdarg.h since it is part of the ANSI C standard. The ANSI-compatible macros defined in stdarg.h are:

- va_arg
- va_end
- va_list
- va_start

However, you can port varargs.h in your existing code to the Alpha. All programs that properly use the varargs macros for variable argument list processing will port unchanged to Windows NT on Alpha.

Below is an example that illustrates the proper use of the stdarg.h macros.

Figure 4-1: Example VARARG.C

```
#include <stdio.h>
#include <stdarg.h>
int average( int first, ... );

void main()
{
    /* Call with 3 integers (-1 is used as terminator). */
    printf( "Average is: %d\n", average( 2, 3, 4, -1 ) );
    /* Call with 4 integers. */
    printf( "Average is: %d\n", average( 5, 7, 9, 11, -1 ) );
    /* Call with just -1 terminator. */
    printf( "Average is: %d\n", average( -1 ) );
}

/* Returns the average of a variable list of integers. */
int average( int first, ... )
{
    int count = 0, sum = 0, i = first;
    va_list marker;
    va_start( marker, first );     /* Initialize variable arguments  */
    while( i != -1 )
    {
```

```
        sum += i;
        count++;
        i = va_arg( marker, int);
    }
    va_end( marker );              /* Reset variable arguments      */
    return( sum ? (sum / count) : 0 );
}
```

4.2 Uninitialized Variables

Alpha platforms are less tolerant of uninitialized data elements than Intel platforms. You are likely to get "Memory Access Errors" when accessing uninitialized variables because garbage data may exist on the high order bits of those variables. You can disassemble in a debugger to help you find the improperly initialized variables. When using a disassembler, a listing from the compiler can be very helpful. Create a .COD file using the /FAcs compiler option. In optimized only cases use -O*n*, -Zi and use the listing file.

4.3 Data Alignment and Fixups

The Alpha memory architecture naturally aligns and references data in 2, 4, or 8 byte quantities. Therefore, your data should be aligned on these same sized boundaries. For example, 16-bit values should be aligned on 2-byte boundaries and 32-bit values on 4-byte boundaries. Applications that perform operations on data that is not naturally aligned to one of these boundaries may exhibit poor performance by faulting or terminating abnormally. For these reasons, applications should avoid unaligned data operations whenever possible.

By default, the Alpha version of Windows NT automatically fixes up unaligned data accesses. This is done by the operating system. Due to the time required to fix a fault, and the number of faults in question, an application can exhibit poor performance.

Automatic fix ups of alignment faults can be turned off on a system and on an application basis. With this capability, you are able to locate where data alignment faults are occurring. Chapter 5.3 provides instructions on how to disable alignment fixups and also how to determine where in your code the alignment problem exists.

For maximum performance, you should make sure to the greatest extent possible, that your data structure components are all aligned on natural boundaries. Where possible avoid byte and word length integers in favor of long words or quadwords. For example, avoid using chars and shorts and use ints or int64s instead. Performance gains upwards of 20% have been observed for applications that were changed to avoid alignment faults.

Refer to Chapter 5.3 on how to use the Performance Monitor to determine if and how many alignments faults are occurring while your application is running.

4.4 Structure Packing

Natural alignment rules can sometimes be difficult to follow. For example, in the case where a data structure from a pre-existing file needs to be read or in applications that use large structures and want to reduce memory requirements. To work around this, the `#pragma pack` directive can be used to control how specific data structures are packed into memory. For example:

Figure 4-2 : Correct Use of #pragma pack

```
#pragma pack(1)
typedef struct
{
struct s {
      char c;       // 1 byte
      int i; // 4 bytes
      short s;      // 2 bytes
} goodexample;
#pragma pack()  //resume default
```

In this example, the `#pragma pack` has caused the compiler to generate a structure with none of the normal padding that it would generally add in order to create naturally aligned data elements. As a consequence, access to these data elements will cause alignment faults. The `UNALIGNED` qualifier, which is discussed in the next section, can be used to avoid the operating system from having to take alignment faults when accessing this type of structure.

Note that `#pragma pack` must be enclosed around an entire structure, not just sections of it. The following example shows the incorrect use of #pragma pack.

Figure 4-3: Incorrect Use of #pragma pack

```
typedef struct
{
struct s {
#pragma pack(1)
      char c;       // 1 byte
      int i; // 4 bytes
#pragma pack()  //resume default
      short s;      // 2 bytes
} badexample;
```

In addition to `#pragma pack` which packs structures on an individual basis, the /Zp<n> C/C++ Compiler switch can be used to pack all structures within a module. The default is /Zp8.

Ideally, to avoid the cost of alignment issues, pre-existing packed structures should be unpacked before used in an application. This requires unpacking a structure element by element to a location that is correctly aligned.

4.5 The UNALIGNED Keyword

If you determine that unaligned access of data is required, use the UNALIGNED pointer qualifier which is defined in the Windows NT system include files. Using UNALIGNED will cause the compiler to insert special code to handle the unaligned data and avoid operating system faults resulting in improved performance. Note however that there is still a performance penalty for accessing data with this technique. The code generated with `#pragma pack` is bigger and slower than aligned access because the compiler will load the 16-bit, 32-bit, or 64-bit object byte-by-byte, or check the alignment dynamically. This is however still much faster than the operating system having to perform fixups on the fly.

Here is an example using UNALIGNED and the packed structure defined above in Figure 4-4.

Figure 4-4: Example using UNALIGNED Qualifier

```
void example_func()
{
      UNALIGNED struct s *ptr = &goodexample;
      ptr->i = 1;
}
```

The keyword UNALIGNED should be used as opposed to __aligned to enable the same code to be compiled across both Intel and Alpha platforms. To ensure portable source code across both platforms, create a macro that defines UNALIGNED as an empty string for the Intel platform. For example:

```
#ifdef (_M_ALPHA)
      #define UNALIGNED __unaligned
#else
      #define UNALIGNED  //NULL
#endif
```

UNALIGNED controls what a pointer points to and has meaning only when used in a pointer declaration. To demonstrate the use of UNALIGNED further, the following is a list of correct and incorrect uses of UNALIGNED.

```
#define UNALIGNED __unaligned

UNALIGNED int              *ip;   // pointer to unaligned int
int UNALIGNED              *ip;   // pointer to unaligned int
int * UNALIGNED             ip;   // incorrect use of UNALIGNED
typedef struct _FOO UNALIGNED    *PFOO; // pointer to unaligned struct
typedef SYMBOLS UNALIGNED  *psymbols;   // pointer to unaligned symbol
UNALIGNED PLONG            NextEntry;   // incorrect
PLONG UNALIGNED            NextEntry;   // incorrect
LONG UNALIGNED             *NextEntry;  // correct
```

See Appendix C for details on datatype natural alignments on the Alpha.

4.6 Alignment Errors and Microsoft Foundation Classes (MFC)

MFC files should be compiled without packing structures. Compile only those modules that do not subclass an MFC class with structure packing. Otherwise, include an MFC class as a member or use the `#pragma pack` directive to limit structure packing to those portions of source code that do not refer to MFC classes.

Because the Classes in the MFC Library do not use the `__unaligned` keyword internally and assume that structure packing is off, when you subclass a class declared in the MFC Library, or if you include an instance of one of these classes as a member, then compiling with packed structures is liable to cause alignment errors.

You need to follow these rules even if you use `__unaligned` to solve alignment problems in your own classes. Similar considerations apply any time you use a class library that assumes structure packing is off.

4.7 64-Bit Integers

The 64-bit integer type is supported by Alpha processors. Visual C++ includes a __int64 keyword to enable use of signed and unsigned 64-bit integers. The following example shows how to declare two 64-bit integers. The first is signed and the other is unsigned.

```
__int64            signed_big_int;
unsigned __int64   unsigned_big_int;
```

Use the format prefix I64 to print out a 64-bit integer.

4.8 Intrinsic Functions

Intrinsic functions are function calls that the compiler resolves directly, by generating code, rather than by calling external function. The Alpha version of the compiler supports number of new intrinsics. With all intrinsics, you must explicitly enable the intrinsics before calling it, by using it with the /Oi compiler option or the "intrinsics" pragma. The general syntax for using this pragma is:

```
#pragma intrinsic(intrinsic-name)
```

Although not portable, refer to the RISC Programmer's Guide for detailed information on Alpha specific instrinsics. They are available for providing synchronization to shared memory, program execution concurrency, semaphores, floating point operations, program control and PAL code instructions.

4.9 Integer Division by Zero

On Alpha, a program that tries to divide an integer value by an integer divisor of zero will by default generate an exception. The Visual C++ debugger can be used to locate and debug integer-division-by-zero errors.

4.10 Floating Point Behavior

All Windows NT platforms use identical IEEE floating point formats. For finite floating point values, Alpha floating point behavior is identical to that of the Intel. However, for non-finite floating point values (e.g., infinity, denormals, and NaNs), the Alpha will raise a floating point exception. The floating point exception codes are:

Table 4-1: Floating Point Exception Codes

STATUS_FLOAT_DIVIDE_BY_ZERO	0xC000008E
STATUS_FLOAT_INVALID_OPERATION	0xC0000090
STATUS_FLOAT_OVERFLOW	0xC0000091
STATUS_FLOAT_UNDERFLOW	0xC0000093
STATUS_FLOAT_INEXACT_RESULT	0xC000008F

If an application requires support of nonfinite floating point values, then use one of the switches described below.

Table 4-2: Floating Point Related Switches

/QAieee0	IEEE floating point NaNs, Infinities, and denormals are not supported in the compiled code. Underflows are quickly forced to zero, and the use of a NaN or Infinity raises an exception. This is the default value, and should be used for all applications except those that require IEEE-floating point exception behavior, since it produces the fastest execution speed. Runtime library routines may still produce NaNs and denormals, however, so the use of _matherr to handle those situations is recommended. If an application does require support for IEEE NaNs and denormals, /QAieee should be used.
/QAieee and /QAieee1	These identical switches specify that IEEE floating point NaNs, Infinities, and denormals are supported. This value should be used for applications that require typical IEEE floating point exceptions enabled.
/QAieee2	Same as /QAieee and /QAieee1, but IEEE Inexact Operation exceptions are also enabled. This value should only be used for applications requiring the IEEE inexact operation exception to be raised (note that this is almost never needed).

4.10.1 Imprecise Location of Floating Point Exceptions

For normal compile modes, if one of the floating point exceptions listed in **Error! Reference source not found.** above does occur, it is likely that the exception PC does not point to the instructions that actually caused the exception. If using a debugger, look for the offending floating point instruction a few instructions prior to the continuation address. Or, if you are looking near the beginning of a function, look at the last few instructions in the calling frame.

If you want the exceptions to be precise, you must compile your code using the /QAieee1 option. This is occasionally useful during debugging and is required if you implement explicit floating point exception handlers with your application.

4.11 Data Granularity Switches

These switches control the level of granularity at which the C/C++ Compiler accesses data.

Table 4-3: Data Granularity Switches

/QAgl (Long Granularity)	For multi-threaded applications, the /QAgl option causes the compiler to access data in units of longwords (4 bytes of data on 4-byte address boundaries) as required by Windows NT. On Alpha systems, this produces slightly less efficient code than quadword granularity.
/QAgq (Quadword Granularity)	For single-threaded applications, the /QAgq option causes the compiler to access data in units of quadwords (8 bytes of data on 8-byte boundaries). If you know that a single-threaded application will have no data accessible to multiple threads of execution, use this option to produce the most efficient code on Alpha systems. Multi-threaded applications may not operate correctly with this option.

4.12 Miscellaneous Alpha Specific Switches

/QAOun	The /QAOu*n* option sets loop unrolling, where *n* specifies the number of iterations of the loop to unroll. If you set *n* to 0, the compiler chooses the number of iterations
/QAltls	The /QAltls option enables the compiler to generate extra code to reference data in thread-local storage when its size exceeds 32K. Enabling large thread-local storage exacts a moderate performance penalty on compiled code. Because of the extra code generated by the use of this switch, use it only when necessary. Linker error LNK2004 indicates that the /QAltls switch might be necessary.

4.13 Context Structure Definition

The CONTEXT structure is an architecture-dependent data structure. The CONTEXT structure contains register data. If you have code that accesses fields in this structure, the application will need to be modified for Alpha.

4.14 Page Size Assumptions

The page size is architecture dependent (4 KB on Intel and 8 KB on Alpha *)* and should not be hard-coded into applications. If the application assumes the page size to be 4KB, it will not work correctly on Windows NT on Alpha. If you use the /base:nn linker option, be sure to align to at least the default page size for your platform.

4.15 LARGE_INTEGER & Quadword Types

The LARGE_INTEGER structure is used to define a 64-bit signed integer value created from an array of two longwords. Although the LARGE_INTEGER data type is a 64-bit integer, it is not the same as a quadword. The quadword is a nonportable 64-bit integer data type and is not supported on Windows NT for Alpha. The LARGE_INTEGER data type can only be used in conjunction with a set of runtime library functions (e.g., LargeIntegerAdd, etc.).

4.16 Multi-Threaded Granularity of Access

Ideally, you would like to have atomic load/store of shared data. An atomic load/store requires a single instruction. On the Intel platform, the size of shared data is 1,2 or 4 bytes, but on Alpha it is 4 or 8 bytes. To ensure portability across platforms, you should protect multi-thread access to shared data structure with locks (e.g. EnterCriticalSection, LeavCriticalSection).

4.17 setjmp/longjmp Jump Buffer

The jump buffer is compiler and version specific. If you use setjmp/longjmp, do not link objects produced by different compilers or versions.

When used in conjunction with longjmp, setjmp provides a way to execute a nonlocal goto. To obtain the correct intrinsic definition of __setjmp for Alpha, you must use #include <setjmp.h> or <setjmpex.h>. Correct interaction with C structured exception handling and C++ exception handling extensions requires the use of #include <setjmpex.h>.

Refer to the RISC Programmer's Guide for additional details on the use of setjmp and longjmp on Alpha.

5. Alpha Application Tuning and Optimization

This chapter is a summary of tips and hints for optimizing your application on Windows NT for Alpha. It is not a how to of performance analysis and optimization techniques for Windows NT. An excellent book on the subject, “Optimizing Window NT" by Russ Blake, is available from Microsoft Press. It is included in the Windows NT Resource Kit. It is highly recommended for any serious optimization work you are planning on Windows NT. In addition, there are several articles on the subject in Microsoft’s MSDN.

5.1 C/C++ Compiler Switches

An important tool in optimizing your application is the compiler. You should become familiar with all the available compiler optimization flags. This will enable you to build applications based on your particular size and speed requirements.

Once your application has been debugged, you should recompile with optimization turned on. This will enable you to measure the performance gain achieved through use of the compiler switches. Performance gains will vary from application to application but typically you can expect a 20-30% or more gain in performance for CPU intensive applications.

Since an application is initially developed in debug mode to facilitate debugging and testing, keep in mind that the compiler has optimization turned OFF. You can easily disable debugging in both nmake and Visual C++ which will enable optimization. For nmake, pass it then nodebug flag. For example, >nmake nodebug=1. For Visual C++ use the toolbar and select Project/Settings. Under the Optimizations field, change Disable(debug) to Maximum Speed.

The execution speed of generated Alpha machine instructions varies depending on the compiler switches specified. The following sections deal with those switches that are available to you based on your specific optimization needs.

5.1.1 Levels of Optimization

The Alpha Version of the C/C++ Compiler provides four basic levels of optimization as documented in Table 5-1.

Table 5-1: Basic Optimization Switches

/Od	Performs no optimization
/O1 or Os	Perform all optimizations that are consistent with keeping the resulting code small
/Ox	Perform optimization that favors speed over size
/O2 or O	Perform all speed optimizations. This option usually produces faster code than Ox, but some times results in much larger code size. Compare the size and speed of an application compiled with /O2 to its size and speed when compiled with Ox before deciding which option to use

5.1.2 Incremental Linking

Incremental linking is enabled by default within Visual C++ for no optimization and debug builds and also for the command line switch /DEBUG. This equates to using the /INCREMENTAL:YES option in the linker. Incremental linking is recommended during application development and debugging. For the release of a product, disable incremental linking because the final executable will run faster.

Incremental linking is turned on indirectly by the linker command option /DEBUG. It can be turned off indirectly by the linker command option /MAP and /PDB:NONE.

5.1.3 Inlining

A limited inline assembler is built into the Alpha version of the Visual C++ compiler. Although not portable, inlining allows you to pass references to C/C++ objects and expressions into the assembly code you write. The compiler switches used to control the optimization of inlining are documented in Table 5-2.

Table 5-2: Inlining Optimizations

/Ob[0,1,2,3,4]	Produces inline optimization /Ob0 disables inlining optimization /Ob1 only inlines small procedures marked with _inline /Ob2 enables automatic inlining, where the compiler balances code size and speed /Ob3 favors speed over code size /Ob4, also called benchmark inlining, tries to inline all calls, resulting in fastest possible performance, but large code size

5.1.4 Quadword Granularity

Use the quadword granularity switch /QAgq with single-threaded applications. This switch produces faster and smaller code. It specifies that all data may be accessed in units of quadwords (8-byte units beginning on 8-byte address boundaries). The default is /QAgl for granularity longword (4-byte granularity). Note that /QAgq may not operate correctly with multi-threaded applications.

5.1.5 Processor Optimizations

The following four command line switches are specific to the Intel platform.

Table 5-3: Intel Processor Specific Switches

/G3	Optimize for 80386
/G4	Optimize for 80486
/G5	Optimize for Pentium
/GB	Optimize for blended model

These switches are accepted but ignored by the Alpha C/C++ compiler. The PROCESSOR_ARCHITECTURE environment variable should be used to distinguish between Intel and Alpha builds.

Processor-specific switches specific to the Alpha processor family are listed below. If one is not used the compiler will default to the blended model which results in code that executes across all platforms.

Table 5-4: Alpha Processor Specific Switches

/QA21064
/QA21064A
/QA21066
/QA21068
/QA21164
/QA21164A

The /QA21164A switch is the only one that produces code that will not run on previous processors. If you do not want to optimize your code for a specific processor than do not use any of these switches.

5.1.6 Floating Point Optimization

The Alpha processor does not handle denormals, or what are considered extremely small numbers. Demormals are rounded to zero by default when they occur to avoid the time it takes to fix them up. For applications that require denormal values, use the /QAieee switch and change the default behavior of flushing denormals to zero by having your application call __controlfp(DN_SAVE, MCW_DN).

The Alpha processor also doesn't support infinity and NaN values. These values slow down application performance since they must be calculated by the operating system. Ideally, these values should be trapped using a debugger and fixed to avoid the cost of them generating exceptions. Use the /QApe switch as described next for further information on how to do this.

Compiling with the /QApe switch will enable precise floating point exceptions. It causes the insertion of trap barriers after every floating-point instruction and thereby allows floating point exceptions to occur. These exceptions can then be trapped by a debugger and fixed.

Compiling with the /QAieee switch enabled, causes floating point exceptions never to be exposed thereby slowing down the performance of your application due to the operation system fixing up exceptions.

The /Op- is the default switch used for overriding floating point precision in favor of faster code. This switch disables specific floating-point optimizations that generally improve performance but may slightly impair accuracy. For example, using inverse multiplication instead of division within a loop, which can reduce floating point accuracy. When developing programs where the highest degree of floating point accuracy is important, use the /Op switch.

5.1.7 Set Data Value

The /Gt switch is used on Alpha to enable the optimal use of the Global Pointer (gp) register for most applications. It controls how data is placed into the area accessed through gp. It is not supported on the Intel for 32-bit applications.

This switch sets the size threshold for small chunks of data. For example, with /Gt32 used, which is the maximum value allowed, the Alpha architecture will access a memory area of 32K. This small data area concept enables gp to access data with one machine instruction rather than two.

The compiler defaults to /Gt0 when the switch is not used.

A note of importance is that all items linked into the same program (i.e., object files and DLLs) must be linked with the same /Gt value. Since system DLLs are usually linked with /Gt0, if your application links them in, then the /Gt switch should not be used. In addition, if you link separately using the LINK.EXE, then the value used with /Gt must also be passed to the linker by using the /gpsize:n option.

5.2 F77 Compiler Switches

If you're using the DEC F77 compiler for your FORTRAN application, the optimization switches which are not enabled by default are listed in the table below.

Table 5-5: DEC F77 Compiler Switches

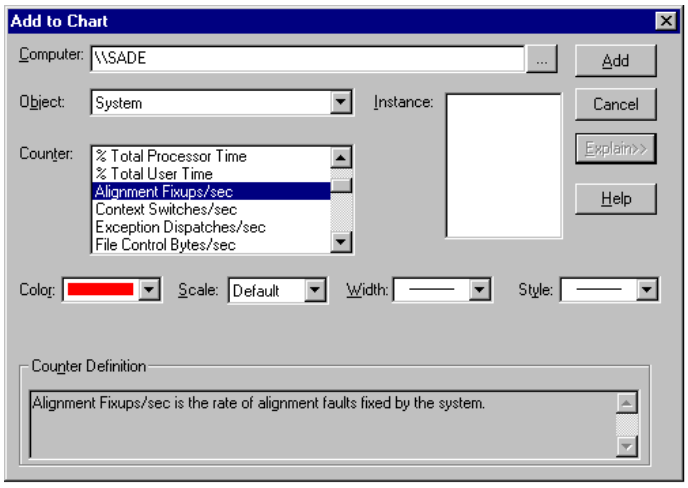
Optimization Switch	Description	Notes
/align:dcommons	Aligns COMMON data blocks on natural boundaries up to eight bytes	These three switches may be enabled en masse via/fast
/assume:noaccuracy_sensitive	Allows floating point operations to be re-ordered	
/math_library:fast	Uses versions of some intrinsics that trade a small amount of accuracy for improved performance	
/inline:all	Inline every possible routine	These switches should be used carefully. By default, the compiler automatically inlines and unrolls according to its heuristics)
/unroll:<count>	Specifies how many times loops are unrolled.	

5.3 Alignment Faults

The Windows NT operating system for Alpha, by default, automatically fixes up alignment faults that may occur during runtime. In most cases this is a desirable feature because the alternative would be for an application to unexpectedly terminate with a data misalignment exception whenever a fault occurred.

Allowing the operating system to resolve alignment faults can significantly degrade the performance of your application. Especially if there are hundreds or thousands of alignment fix ups occurring per second. A developer can use the Performance Monitor to determine the number of alignment fix ups that are occurring while their application is running. In order to do this, bring up the Performance Monitor which is located under the Administrative Tools in the Start Menu. Then select the System Object and Alignment Fixups/Sec Counter as shown in **Error! Reference source not found.**.

Figure 5-1: Alignment Fixups Per/Sec Counter



If the Performance Monitor is reporting a significant number per second, then your application may be taking unnecessary performance hits. The fix ups should be eliminated since addressing alignment faults is essential for obtaining good performance with the Alpha architecture.

In order to eliminate alignment faults, you must change the default operating system control for alignment exceptions so that alignment faults become visible to your application. The debugger can then be used to locate the source of the alignment faults. Alignment faults can be typically avoided by not turning on structure packing and not accessing a small-aligned address by using a recast pointer of larger alignment.

Instructions for making alignment faults visible and tracking down their source are described in the sections that follow.

5.3.1 Changing Automatic Fixing of Alignment Errors

The operating system default to automatically fix up alignment errors can be changed using the `axpalign` command. This command comes bundled with the C/C++ compiler. It is run from a DOS command prompt and you must have Administrator privileges to execute it.

- `axpalign /enable` - this command causes alignment fault exceptions to occur. It will result in data misalignment exceptions. This is used for debugging the source of the fault.

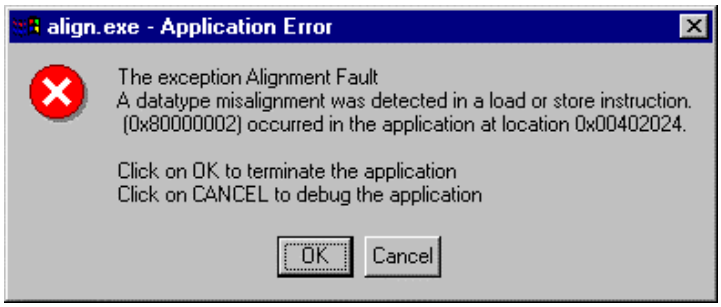
- `axpalign /disable` - this command disables alignment fault exceptions to occur. Here the operating system fixes up any misaligned data accesses. Therefore, faults cannot be seen by applications or debuggers.

- `axpalign /show` - this command allows you to view the current alignment exceptions setting.

Executing this command requires that the system be rebooted for the change made to be implemented. The change applies system-wide and affects all applications. Therefore, alignment faults should be enabled only while locating alignment faults in your application. Otherwise older applications that still contain alignment errors will terminate with data misalignment exceptions.

Once alignment fix ups are disabled, any alignment errors in a program will be terminate with a windows similar to that in Figure 5-2.

Figure 5-2: Example Alignment Fault



If your application requires that the operating system handle alignment faults regardless of the operating system alignment exception control, you should insert the following statement early in your program:

```
SetErrorMode(SEM_NOALIGNMENTFAULTEXCEPT);
```

This statement must be executed before any alignment error can occur. The effect of the statement is to set a flag that causes the operating system to handle alignment faults for your program.

5.3.2 Debugging Alignment Faults

This section provides an example on how to track down alignment faults using the Visual C++ integrated debugger. Here is an example piece of code that was written to purposely generate an alignment error. It also demonstrates use of the UNALIGNED keyword.

```
void get_data(int *ptr)
{
  (*ptr)++;
}
void aligned_data(UNALIGNED int *ptr)
{
  (*ptr)++;
}


int main()
{
  char data[16] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};

  get_data((int *) &data[0]);     /* Aligned data */
  get_data((int *) &data[1]);     /* Unaligned data */
  aligned_data((int *) &data[1]); /* OK-Unaligned data expected */

  return 0;
}
```

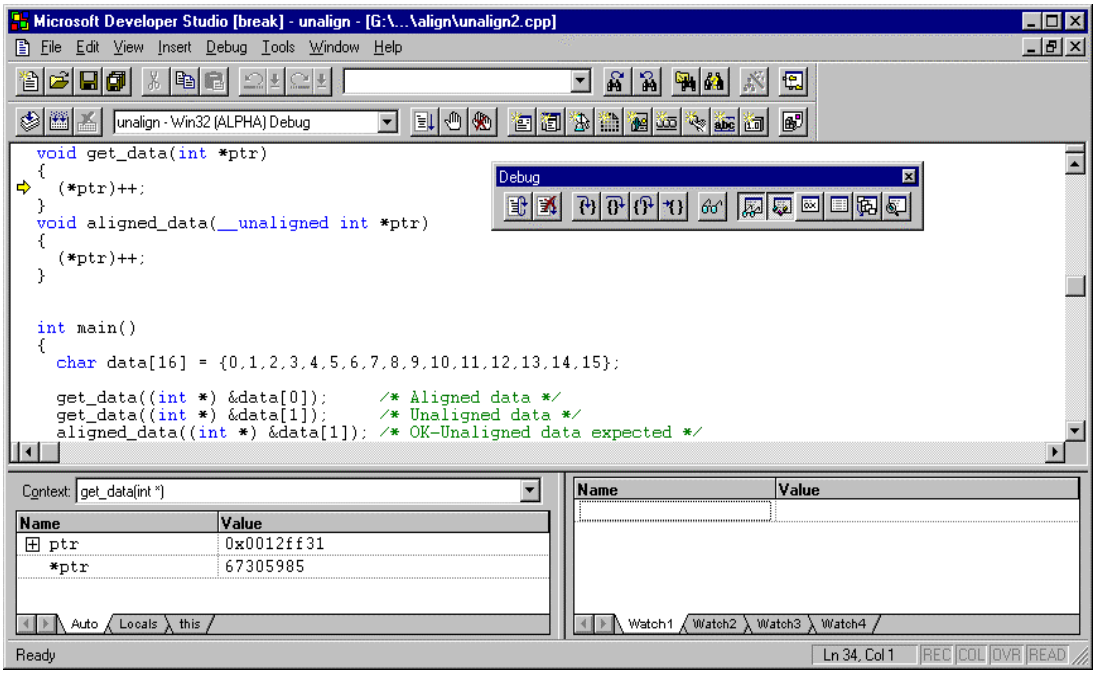
Before executing this application, issue the command `axpalign /enable` to ensure that the program will terminate when an alignment fault occurs.

Once this program has been built successfully, either execute it from the command line or explore and then use the Visual C++ Developer's Studio to debug. Otherwise, execute the program directly from another debugger. Either path will generate an alignment fault from which you will then be able to determine the source of and fix the problem.

When running your application from the command line or explorer, click on CANCEL to debug the application. This will bring up Visual C++ Developer Studio and the misalignment error will be reported as such:

Clicking on OK will cause the debugger to point out via an arrow where in the code the exception occurred. The figure below shows the debugger attempting to access unaligned data. Notice that the value of `*ptr` is equal to 67305985 which does not fall on a naturally boundary.

Figure 5-3: Debugger Broken At Unaligned Data Access



5.3.3 Techniques for Addressing Alignment Faults

There are several techniques for addressing alignment faults.

- Use the UNALIGNED compiler keyword if your application must access unaligned data. It is faster than letting the kernel fixup unaligned data. See Chapter 4.5 for a description of the UNALIGNED type qualifier.

- Do not use the `#pragma pack` directive unless absolutely necessary since it will cause alignment faults to occur. See Chapter 4.4 for a description of the `#pragma pack` directive.

- Avoid misleading casts. Pointer casts are used to cast one pointer type to another. One subtle and undetectable problem with this can occur when casting to a pointer with stronger alignment requirements. For example,

```
long *ip;
...
percent = (double *)ip / 100.0;
```

will cause an alignment fault if the value of pointer `ip` is not a multiple of 8. Since `ip` is merely a pointer to a 4-byte long there is in general no reason to believe it will be a multiple of 8. The compiler cannot detect this at compile time.

Another example is when the address for some data is obtained from a memory allocation function that makes no guarantee about the alignment of the address it returns. For example, if `MyAlloc()` does not round up addresses to 8-byte multiples, then:

```
TimePtr = MyAlloc(8);
```

```
    ...
NtQuerySystemTime((PLARGE_INTEGER)TimePtr);
```

may result in an alignment trap since NtQuerySystemTime is expecting the address of a properly aligned LARGE_INTEGER type.

On Alpha platforms the code may result in an alignment trap since LARGE_INTEGER types are treated as one naturally aligned quadword.

- Do not pass unaligned pointers to another function. If you take the address of a member of an unaligned or packed structure and pass it as an argument to a function, with or without a cast, it is likely that an alignment trap will occur.

```
struct _FOO {
..
long count;
};
struct _FOO UNALIGNED *pFoo;
void SetCounter(long * ip);
..
pFoo->count++;              // Ok, the compiler knows the long
                            // may be unaligned.

SetCounter(&pFoo->count);   // WRONG! Function SetCounter is
expecting a point to a normal, aligned long.
```

One work-around in this case is to type SetCounter to receive an UNALIGNED pointer rather than a normal pointer. Another work-around is to declare a local count variable and pass its address to SetCounter.

Note that NT system services do not explicitly enforce quadword alignment of quadword pointer parameters.

5.4 Performance Tools

An introduction to the key tools for optimizing applications are briefly described here. They are partitioned into several sections. The first being those tools that are integrated in Windows NT 4.0, those that are available in the Window NT Resource Kit, and those that are bundled with Visual C++.

5.4.1 Windows NT 4.0 Tools

- **Windows Performance Monitor** - this is a windows-based tool for analyzing the performance of your system or application. It can be used to analyze system data that keeps a count of CPU usage, disk I/O activity, memory statistics and network traffic. These are all the key areas for potential bottlenecks. In addition, you can log this data for future capacity planning. It is located under the Start→Programs→Administrative Tools menu.

- **Task Manager** - this is a new tool integrated as part of Windows NT 4.0. It allows you to monitor active applications and processes on your system. It uses a lot of the same functions used by the Windows Performance Monitor except it calls them directly, bypassing the Performance Library and Registry. It is a useful tool for quick checks of basic counters. To start it either press CTRL+SHIFT+.ESC, click the right mouse button on the Task bar, or press CTRL+ALT+DELETE.

5.4.2 Windows NT Resource Kit 4.0 Tools

- **Process Monitor** - this tool (PMON.EXE) is used to display process statistics in text format in a command prompt window. The data is automatically updated every five seconds.

- **Process Explode** - this tool (PVIEW.EXE) provides a wealth of accurate and detailed information on many aspects of your system, processes, threads, and memory. A lot of the same information is available from the Windows Performance Monitor. The difference being there is no setup required and all the information is provided in a single dialog box.

- **Process Viewer** - this tool (PVIEWER.EXE) displays information about processes on local and remote computers and is especially useful for investigating process memory use. It also lets you stop a running process and change the base priority class of the process.

- **SC Utility** - this is a command line interface for the service controller. It displays the configuration of the computer on which it is run. At the command prompt, type SC to see a brief description of the available commands.

- **Microsoft Windows NT Call Profiler (CAP)** this is a general purpose profiling tool that can be used to measure the function call performance of .EXE and .DLL files in a variety of ways. It records how long a function takes to execute and how it spends that time. CAP creates a call tree for each thread for tracking the flow of control within a particular thread.

  CAP requires that the application be recompiled with the /Gh and /Zd switches and linked with the CAP.LIB library.

5.4.3 Visual C++ Tools

- **Source Profiler** is a powerful analysis tool for examining the run-time behavior of an application. Use it to generate information to examine sections of code. It is comprised of modules called PREP, PROFILE and PLIST. Here is a brief description of each. See the Visual C++ User's Guide for further details.

  - PREP.EXE - This is used to setup an application for profiling. The PREP program runs twice during a normal profiling operation. In Phase I, it reads an executable file and then creates .PBI and .PBT files. In Phase II, it reads the .PBT and .PBO files and then writes a new .BPT file for the module PLIST. You can profile by functions counts, function times, function coverage and line numbers.

  - PROFILE.EXE - This module profiles an application and creates .PBO file of the results. If you do not specify a .PBO filename, then PROFILE uses the base name of the program name with the .PBI and .PBO extensions.

  - PLIST.EXE - This module converts results from a .PBT file into a formatted text file. It must be run from the directory in which the profiled program was compiled.

- **Spy++** (SPYXX.EXE) is a Win32-based utility that gives you a graphical view of the system's processes, threads, windows, and window messages. You can display a graphical tree of relationships among system objects, including processes, threads, and windows or search for specified windows, threads, processes, or messages.

6. Troubleshooting C/C++ Compiler and Link Problems

There are some hidden switches described here which can be used to troubleshoot problems you may have while compiling and linking.

6.1 Compiler Problems

For compiler problems, such as internal compiler errors (for example, C1001), hangs, or crashes, use the /Bd and /P switches to obtain additional compiler information. They will display the different passes of the compiler as it progresses through the phases of compilation and optional linking.

The /P compiler switch directs the compiler to send preprocessor output to a file. The preprocessor output file (with an extension of .i) has the same name as the file being compiled and is created in the same directory. Note that a build will not continue past the preprocessor phase when using this switch.

The undocumented /Bd switch directs the compiler to display all of the switches it is using. In addition, it displays the names of temporary files used in the build process.

You can compile the preprocessor output file outside of the context of a project. The file contains all of the header file code, macro replacement, and preprocessed compiler directive information needed for the compilation of that particular .C or .CPP source file. The resulting file will often be long and contain a large amount of white space.

Alternatively, you can use the /EP and /E compiler switches, which direct CL.EXE to send preprocessor output to the standard output device.

6.2 Linker Problems

For linker problems (LNKxxxx type errors), you can use the LINK_REPRO environment variable to reproduce the problem. Follow the steps below for using LINK_REPRO.

1. Open up a command prompt session.

2. Set LINK_REPRO to point to an existing and empty directory, for example:
   ```
   SET LINK_REPRO=C:\TESTPROB
   ```

3. Run the linker in the same command prompt session where the variable was set.

4. Run the VCVARS32.BAT file from the Visual C++ BIN directory to set the compiler environment variables.

5. Change to your project directory, and run NMAKE to rebuild your project, using the following command. Note that the /a option directs NMAKE to perform a complete rebuild of the project.

   ```
   NMAKE /a /f yourproj.mak
   ```

When LINK.EXE is invoked, it will copy everything it needs to link your project into the directory specified by the LINK_REPRO environment variable. Among the files copied will be your object files (*.OBJ), required library files (*.LIB), including Microsoft libraries, and a linker response file (LINK.RSP), so that LINK is no longer dependent on your project makefile.

This feature is also available from Visual C++ if it has been run from an environment where the LINK_REPRO variable has been set.

To confirm that you have all the necessary files to reproduce the link problem, you can run LINK in the directory specified by the LINK_REPRO environment variable, using the linker response file:

```
LINK @link.rsp
```

Use the following command to switch off this feature:

```
SET LINK_REPRO=
```

You can also use the LINK_REPRO environment variable to verify the files involved in creating a library, when using LIB.EXE or LINK /LIB.

7. FX!32 Binary Translator

FX!32 is Digital's Alpha to x86 binary translator. It provides fast and transparent execution of x86 32-bit applications on the Windows NT 4.0 Alpha platform. It provides a utility called Install x86 Application which enables you to install a 32-bit Intel application on Alpha, double-click its icon, and it runs just as if it were running on a Pentium system. In most cases, the performance on Alpha equals or betters Pentium performance. This is because FX!32 executes an application faster each time it runs until the application has been fully optimized. Overall, FX!32 performance depends on your specific Alpha system.

FX!32 supports end-user x86 32-bit applications that run on Windows NT 4.0 and use the standard Windows NT Application Programming Interfaces (APIs). Applications such as device drivers, services, screen savers, and debuggers are not end user applications since they include unsupported system-level code. See Digital's web site at http://www.digital.com/semiconductor/amt/fx32/fx-testapps.html for a list of those applications which have been tested to run under FX!32.

FX!32 requires a Windows NT 4.0 Alpha system with at least 32MB of memory, although 64MB is recommended. A README file within the latest FX!32 kit can be retrieved from http://www.digital.com/semiconductor/amt/fx32/index.html. It contains useful instructions for running DIGITAL FX!32 in lower memory systems.

7.1 How FX!32 Works

When you run a Win32 application that was compiled for an x86 system, FX!32 knows that the application is not native. It therefore loads the Win32 x86 application and any previously created optimized code and starts the application. As you run the application, FX!32 collects information about the application and saves this information as a series of profiles.

After you exit from the application, FX!32 uses the profile information to create more optimized code so that the next time you run that application, it runs faster. By default, FX!32 begins to generate optimized code as soon as you exit from the application. However, depending on your particular system, you might want to schedule optimization. FX!32 provides a Scheduler you can use to do that.

Usually, the greatest improvement occurs after the first or second time you run the application. Because FX!32 works transparently, you can just let it work quietly in the background, making your Win32 x86 applications run faster and faster.

7.2 Obtaining FX!32

DIGITAL FX!32 is available for downloading from the following web site.

`http://www.digital.com/semiconductor/amt/fx32/fx-download.html`

It is also bundled with all new Windows NT Alpha systems. Or for a small handling fee, you can order the DIGITAL FX!32 CD-ROM from 1-800-DIGITAL, order no. QR-21BFX-70.

8. Spike

Spike is an optimizer for Alpha/NT executables that can be used to optimize shipping products. It allows a user to easily instrument and optimize large applications composed of many images. Spike solves a lot of the problems associated with collecting, managing, and applying profile information. The optimizations performed by Spike have been know to provide significant performance improvement by reducing execution time by 5-20% across a wide range of applications.

Spike consists of the Spike Optimization Environment (SOE) and the Spike Optimizer. SOE provides a simple means to instrument and optimize applications. Both are described in more detail here.

8.1 Spike Optimization Environment (SOE)

SOE provides the means for instrumenting and optimizing applications. The user selects the application(s) that are to be instrumented and optimized. The user only needs to specify the application's main image since Spike is able to find all the implicitly linked dlls. SOE maintains a database of these applications and images (exes and dlls) as well as any profile information collected for those images.

In addition, SOE simplifies the collection of profile information with Transparent Application Substitution (TAS). TAS is a mechanism for transparently executing a modified version of an application, without replacing the original images on disk. The user simply executes the original application and the instrumented or optimized application is run in its place.

SOE can be accessed through a gui or a command-line interface. The gui is referred to as the Spike Manager and the command-line interface allows SOE to used as part of a batch build system such as `make`.

8.2 The Spike Optimizer

The Spike optimizer is used to instrument and optimize images. The optimizer is invoked by SOE and can also be directly invoked by the user. It performs code layout to improve instruction cache behavior and register allocation. It also performs what is referred to as hot-cold optimization where code is split into frequently executed (hot) and infrequently executed (cold) parts. The hot sections are placed so that frequently called routines are near the caller and cold routines are collected at then end of an image. This reduces the chance that routines that call each other will have addresses that clash in the instruction cache.

8.3 Unsupported/Special-Case Applications

Spike cannot currently optimize the following types of images:

- Images with multiple text sections
- Images that load in the system area (> 0x80000000)
- Non-Alpha images

Note that there have been some problems using the Spike Optimization Environment with

applications that contain a license manager, particularly FLEXlm. If the license manager is included in the application (e.g. as a dll) then it may not be possible to use SOE with that application. If it runs as a separate application, then it should be registered with Spike.

8.4 Using Spike

Why are there references to it being compiled in a special way? I thought an app didn't need to be recompiled.

Here is an example of how to use Spike to instrument and optimize the program `PROG.EXE`.

1. Create an instrumented executable.

   ```
   spike -pixie PROG.EXE PROG.PIX
   ```

   This will produce two files: `PROG.PIX` and `PROG.fbi`.

2. Run `PROG.PIX` with representative input. This will generate a feedback file `PROG.fbc`. `pix.dll` must be on the search path. The `.fbc` file is not written until the program exits, so be sure the process has stopped before using a profile.

3. Optimize the program:

   ```
   spike -Ox -fb PROG PROG.EXE PROG.OPT
   ```

   The argument `-fb PROG` will make Spike look for `PROG.fbc` and `PROG.fbi`. You can use `-fbi <filename>` and `-fbc <filename>` to specify the full patch name for each file.

8.5 Obtaining Spike

Version 1.0 of Spike is being distributed on the ASAP Partners CD and the Microsoft Professional Developers Conference CD. It will be soon be available from Digital's web pages.

9. Public Domain Tools for Alpha

The following are common utilities that are publicly available over the Internet for the Alpha platform. You can access the public domain tools from the tools menu item of Digital's Alpha Developer Support home page in the World Wide Web at http://www.digital.com/info.html.

Most of the utilities also contain the executables for Intel based Windows NT machines. Please note that all of these tools were collected from the Internet and are copy righted as per the agreements in the individual source code. Digital makes no warranties, either written or implied with regards to this software.

TOOL	**DESCRIPTION**
bsdcmpat.lib	A library of routines to help in porting to NT. Routines include: bcmp, bcopy, bstring, zero,getopt, index, isctype. Headers include: ctype, getopt, paths, string, strings, unistd.
cal.exe	The cal command prints a calendar. If month (number between 1 and 12) is specified, only that month is printed for that year. Year can be between 1 and 9999.
cat.exe	The cat command reads each file in sequence and displays it on the standard output.
cmp.exe	The cmp command compares two files. With no options, cmp makes no comment if the files are the same. If they differ, it reports the byte and line number at which the difference occurred to the standard output.
color.exe	The color command changes the color of the foreground and background. The available colors are: black, blue, green, cyan, red, magenta.
comm.exe	comm compares sorted data.
compress.exe	The compress command used modified Lempel-Ziv. This command is compatible with the compress/decompression used on the Unix systems compress programs. This is version 4 and supports up to 16 bit compression.
egrep.exe,grep,exe	The egrep command searches file for regular expressions. egrep patterns are full regular expressions. Grep searches file for regular expressions. The command patterns are limited regular expressions in the style of 'ex'.
flex.exe	"Fast Lexical Analyzer generator", Flex is a tool for generating programs which recognizes lexical patterns in text. Flex generates its output as a C code source file.
fold.exe	The fold command is a filter which folds the contents of the specified files, or the standard input if no files are specified, breaking the lines to have a maximum of 80 characters.
head.exe	The head command gives the first n lines of each of the specified files or the standard input.
ls.exe	This is a UNIX is work-alike, although some options are different or absent. Use the '-?' command line argument for details. The executable included is compiled for i386. It should be Unicode compatible, but hasn't been tested.
mawk.exe	The mawk command is an implementation of the AWK programming language.

mewinnt.exe	MicroEMACS windows editor.
par.exe	The par command is a filter which copies its input to its output, changing all white characters (except newlines) to spaces, and re-formatting each paragraph. Paragraphs are delimited by vacant lines, which are lines containing no more than a prefix, suffix, and intervening spaces (see the Details section for a precise definition).
perl.exe	Perl is a language that combines some of the features of C,
perlglob.exe	sed, awk and shell.
sed.exe	The GNU sed is a batch stream editor. This sed uses regular-expression routines from EMACS, which are not as fast as they could be. For speed, use perl.
soss.exe	SOSS is a file server conforming to SUN Microsystems' NFS protocol version 2.
tar.exe	A version of the "tape archive" command available on most UN*X machines based upon the GNU tar utility. It does not yet utilize the tape drive available.
uniq.exe	The unique command reports repeated lines in a file. The repeated lines must be adjacent in order to be found.
unshar.exe	The unshared command extracts files from the SHELL archive.
viewps.exe	PostScript text extractor.
win100.exe	Kermit and terminal emulator.
winvn.exe	Visual Usenet news reader for Microsoft Windows.
xstr.exe	The xstr command extracts and hashes strings in a C program.
xvi.exe	The xvi command is a portable multi-window version of UNIX editor 'vi'.
yacc.exe	Berkeley Yacc is an LALR parser generator. It has been made as compatible as possible to AT&T Yacc.
zip.exe,unzip.exe	The zip command packages and compresses (archive) files. The unzip command lists/test/extracts from a ZIP archive file.

Appendix A Compiler Options Comparison

Alpha Compiler	Intel Compiler	Description
/?,/help	/?,/help	print compiler options
/batch	/batch	batch compiler mode
/Bd	/Bd	verbose, shows all default macros and include files
/c	/c	compile only, no link
/C	/C	preserve comments during preprocessing
/D<name>{=|#}<text>	/D<name>{=|#}<text>	define macro or constant
/E	/E	preprocess to stdout
/EP	/EP	preprocess to stdout, but no #line
/F<num>	/F<num>	set stack size
/Fa[file]	/Fa[file]	name assembly listing file
/FA<s|c>	/FA<a|s|c>	configure assembly listing in compiler, for Alpha /FAa=/FAc
/Fd[file]	/Fd[file]	specify program database .PDB file
/Fe<file>	/Fe<file>	specify executable file
/FI[file],/Fc[file]	/FI[file]	specify forced include file, use /FAc, /FAcs
/Fm[file]	/Fm[file]	specify linker map file
/Fo<file>	/Fo<file>	specify object file
/Fp<file>	/Fp<file>	specify precompiled header file
/Fr[file]	/Fr[file]	specify source browser file
/FR[file]	/FR[file]	specify extended .SBR file
not available	**/G3**	**optimize for 80386**
not available	**/G4**	**optimize for 80486 (default)**
not available	**/G5**	**optimize for Pentium**
not available	**/Gd**	**__cdecl calling convention**
/Ge	/Ge	enable stack checking calls (default)
/Gf	/Gf	enable string pooling
/Gh	/Gh	enable hook __penter function call
not available	**Gr**	**__fastcall calling convention**
/Gs[num]	/Gs[num]	disable stack checking calls
/Gt<n>	/Gt<n>	threshold for gp-relative data
/Gy	/Gy	separate functions for linker
not available	/Gz	__stdcall calling convention
/H<num>	/H<num>	max external name length
/I<dir>	/I<dir>	add to include search path
/J	/J	default char type is unsigned
/link	/link	linker control options
not available	**/LD**	**create .DLL**
/MD	/MD	link with MSVCRT.LIB
/MD	/ML	link with LIBC.LIB
/nologo	/nologo	logo suppress copyright message
/O	**not available**	**maximum speed, /Oi /Ob2 or /O2**
/O1	/O1	minimize space
/O2	/O2	maximize speed (same as /O)
not available	**/Oa**	**assume no aliasing**

Alpha Compiler	Intel Compiler	Description
/Ob<0|1|2>	/Ob<0|1|2>	inline expansion (default n=0)
/Od	/Od	disable optimizations (default if /Zi and no /O*
not available	**/Og**	**enable global optimization**
/Oi[-]	/Oi[-]	enable intrinsic functions (default=Oi-)?
/Op[-]	/Op[-]	improve floating-pt consistency (decrease performance)
not available (see /O1)	**/Os**	**minimize space**
not available (see /O2)	**/Ot**	**maximize speed**
not available	**/Ow**	**assume cross-function aliasing**
/Ox	/Ox	maximum opts (/Ogitybl /Gs for x86, /Oi /Ob2 for ALpha
not available	**/Oy[-]**	**enable frame pointer omission**
/P	/P	preprocess to file
not available	/QmipsGx	generate MIPS-specific instructions
/QAgl	**not available**	**generate fetches and stores in units of longword**
/QAgq	**not available**	**generate fetches and stores in units of quadward**
/QAieee, /QAieee1	**not available**	**IEEE floating point NaNs, inifinities and denormals support**
/QAieee0	**not available**	**disable IEEE floating point support**
/QAieee2	**not available**	**/QAieee1 and IEEE Inexact Operation exception support**
/Tc<source file>	/Tc<source file>	compile file as .c
/Tp<source file>	/Tp<source file>	compile file as .cpp
/u	/u	remove all predefined macros
/U<name>	/U<name>	remove predefined macro
/V<string>	/V<string>	set version string
/vd<0|1>	/vd<0|1>	disable/enable vtordisp
/vmb	/vmb	best case for pointers to class members
/vmg	/vmg	full generality for pointers to class members
/vms	/vms	define single inheritance
/vmm	/vmm	define multiple inheritance
/vmv	/vmv	define virtual inheritance
/w,/W0	/w,/W0	disable all warnings
/W<n>	/W<n>	set warning level (default n=1)
/WX	/WX	treat warnings as errors
/X	/X	ignore standard include directories
/Yc[file]	/Yc[file]	create .PCH file
/Yd	/Yd	put debug info in every .OBJ
/Yu	/Yu	use .PCH file
/YX[file]	/YX[file]	automatic .PCH
/Z7	/Z7	C7 style CodeView information
/Za	/Za	ANSI compatibility (implies /Op)
/Zd	/Zd	debugging information
/Ze	/Ze	enable extensions (default)
/Zg	/Zg	generate function prototypes
/Zh	/Zh	home arguments (low-level debugging)

Alpha Compiler	Intel Compiler	Description
/Zi	/Zi	prepare for debugging (CodeView I information for windbg)
not available	**/Zl**	**omit default library name in .OBJ**
/Zn	/Zn	turn off SBRPACK for .SBR files
/Zp[n]	/Zp[n]	pack structs on n-byte boundary
/Zs	/Zs	syntax check only

Appendix B Datatypes Comparison

Standard C Datatype (byte)	Alpha	Intel
char	1	1
unsigned char	1	1
short	2	2
unsigned short	2	2
int	4	4
unsigned int	4	4
long	4	4
unsigned long	4	4
void *	4	4
char *	4	4
float	4	4
double	8	8
long double	8	10

All data types are identical except for long double

Appendix C Data Type Natural Alignment

Data Type	Alignment Starting Position
8-bit character string	Byte boundary
16-bit integer	Address that is a multiple of 2 (word alignment)
32-bit integer	Address that is a multiple of 4 (longword alignment)
64-bit integer	Address that is a multiple of 8 (quadword alignment)
IEEE floating single S	Address that is a multiple of 4 (longword alignment)
IEEE floating double T	Address that is a multiple of 8 (quadword alignment)
IEEE floating extended X	Address that is a multiple of 16 (octaword alignment)
IEEE floating single S complex	Address that is a multiple of 4 (longword alignment)
IEEE floating double T complex	Address that is a multiple of 8 (quadword alignment)
IEEE floating extended X complex	Address that is a multiple of 16 (octaword alignment)

Appendix D Bibliography

1. \mstools\bin\cap.txt, Microsoft's Win32 Software Development Kit for Alpha AXP.
2. \mstools\bin\wap.txt, Microsoft's Win32 Software Development Kit for Alpha AXP.
3. Windows NT for Alpha AXP Calling Standards, Digital Equipment Corporation, Rev 1.8
4. Optimizing Windows NT, *Russ Blake*, Microsoft Corporation, Summer, 1993
5. Tools User's Guide, Microsoft's Win32 SDK Version 3.5, Microsoft Corporation, 1993
6. Visual C++ Risc Programmer's Guide, Visual C++ Online Documentation