*Chapter 1*

# N32 ABI Overview

Welcome to the N32 ABI Handbook. This book describes the N32 High Performance 32-bit Application Binary Interface (ABI) for the MIPS architecture.

## Contents of This Guide

As you continue reading this guide, you'll learn about N32. Topics include:

- Chapter 1 (this chapter), "N32 ABI Overview"
  - "What is N32?," which describes the n32 ABI and compares it with the other MIPS ABIs.
  - "Why We Need a New ABI," which lists the reasons why we need a new ABI.
  - "N32 Migration Requirements," which describes what is required of both SGI and its customers to use the n32 ABI.
- Chapter 2, "Calling Convention Implementations"
- Chapter 3, "N32 Compatibility, Porting, and Assembly Language Programming Issues"
- Chapter 4, "N32 Examples and Case Studies"

This document uses the following terminology:

| | |
|---|---|
| o32 | The current 32-bit ABI generated by the ucode compiler, that is, 32-bit compilers prior to IRIX 6.1 operating system. |
| n32 | The new 32-bit ABI generated by the MIPSpro 64-bit compiler. |
| n64 | The new 64-bit ABI generated by the MIPSpro 64-bit compiler. |

## What is N32?

N32 is a minor variation on the high performance 64-bit ABI. All the performance of the hardware is available to the program and existing 32-bit ABI programs are easily ported. Table 1-1 compares the various ABIs.

**Table 1-1** ABI Comparison Summary

| | o32 | n32 | n64 |
|---|---|---|---|
| Compiler Used | ucode | MIPSpro | MIPSpro |
| Integer Model | ILP32 | ILP32 | LP64 |
| Calling Convention | mips | new | new |
| Number of FP Registers | 16 (FR=0) | 32 (FR=1) | 32 (FR=1) |
| Number of Argument Registers | 4 | 8 | 8 |
| Debug Format | mdbug | dwarf | dwarf |
| ISAs Supported | mips1/2 | mips3/4 | mips3/4 |
| 32/64 Mode | 32 (UX=0) | 64 (UX=1) * | 64 (UX=1) |

* UX=1 implies 64-bit registers and also indicates that MIPS3 and MIPS4 instructions are legal. N32 uses 64-bit registers but restricts addresses to 32 bits.

## Why We Need a New ABI

The Application Binary Interface, or ABI, is the set of rules that all binaries must follow in order to run on an SGI system. This includes, for example, object file format, instruction set, data layout, subroutine calling convention, and system call numbers. The ABI is one part of the mechanism that maintains binary compatibility across all SGI platforms.

Until IRIX operating system version 6.1, Silicon Graphics supported two ABIs: a 32-bit ABI and a 64-bit ABI. IRIX 6.1 supports a new ABI, n32.

The following sections outline limitations of the old 32-bit ABI the 64-bit ABI. These issues form the motivation for the new n32 ABI. Specifically, topics covered include:

- "Limitations of the 32-bit ABI"
- "Limitations of the 64-bit ABI"
- "Motivation for the N32 ABI"

## Limitations of the 32-bit ABI

The 32-bit ABI was designed essentially for the R3000. We can't extend the 32-bit ABI to use new performance-related features and instructions of the R4400 and beyond. For example:

- We can't use 16 of the 32 floating point registers.
- We can't use any 64-bit arithmetic instructions.
- We can't use any 64-bit data movement instructions.
- We can't use any MIPS4/R8000 instructions.

Because of this, we lose the performance available from the chip. Floating point intensive programs are especially hurt by these limitations; indeed some are 50%-100% slower!

## Limitations of the 64-bit ABI

Although the 64-bit ABI exploits many performance-related features of the MIPS architecture, it also has issues that make using it problematical. These issues include the following:

- Many ISVs have been unable to port their code from the 32-bit ABI to the 64-bit ABI. This is not a simple port; in fact, it usually requires redesign and recoding.
- When ported from the 32-bit ABI to the 64-bit ABI, some C programs get significantly larger.

### Motivation for the N32 ABI

Many ISVs and customers are finding it difficult to port to the 64-bit ABI. Clearly we need an ABI with all of the performance advantages of the 64-bit ABI, but with the same data type sizes as the 32-bit ABI to allow ease of porting.

## N32 Migration Requirements

In order to implement n32, Silicon Graphics provides the following to our customers:

- A compiler that supports n32. (The 6.1 version of MIPSpro compilers supports n32).
- A kernel that supports n32. (IRIX 6.1 supports n32).
- N32 versions of each library.

To take advantage of the n32 ABI, our customers must:

- Install n32 OS, compiler, and all n32 libraries.
- Rewrite Assembly code to conform to n32 guidelines.
- Prototype C functions that use *varargs.*
- Recompile all the code with the compiler that supports n32 (use **–n32** on the command line). *Makefile* changes are needed only if explicit library paths are used.