1

Calling Conventions

1.1 Introduction

Each high-level language compiler generates subroutine entry and exit code based upon a set of rules known as the compiler's **calling convention**. These rules address the following issues:

- *register preservation and usage*
- *parameter passing*
- *call and return mechanism*

A compiler's calling convention is of importance when interfacing to subroutines written in another language either assembly or high-level. Even when the high-level language and target processor are the same, different compilers may use different calling conventions. As a result, calling conventions are both processor and compiler dependent.

1.2 Processor Background

All members of the i960 architecture family support two methods for performing procedure calls: a RISC-style branch-and-link and an integrated call and return mechanism.

On a branch-and-link, the processor branches to the invoked procedure and saves the return address in a register, **G14**. Typically, the invoked procedure will not invoke another procedure and is referred to as a **leaf procedure**. Many high-level language compilers for the i960 family recognize leaf procedures and

automatically optimize them to utilize the branch-and-link mechanism. Branch-and-link procedures are invoked using the **bal** and **balx** instructions and return control via the **bx** instruction. By convention, **G14** is zero when not in a leaf procedure. It is the responsibility of the leaf procedure to clear **G14** before returning.

The integrated call and return mechanism also branches to the invoked procedure and saves the return address as did the branch and link mechanism. However, the important difference is that the **call**, **callx**, and **calls** instructions save the local register set (**R0** through **R15**) before transferring control to the invoked procedure. The **ret** instruction automatically restores the previous local register set. The i960CA provides a register cache which can be configured to retain the last five to sixteen recent register caches. When the register cache is full, the oldest cached register set is written to the stack.

1.3 Calling Mechanism

All **RTEMS** directives are invoked using either a **call** or **callx** instruction and return to the user via the **ret** instruction.

1.4 Register Usage

As discussed above, the **call** and **callx** instructions automatically save the current contents of the local register set (**R0** through **R15**). The contents of the local registers will be restored as part of returning to the application. The contents of global registers **G0** through **G7** are not preserved by **RTEMS** directives.

1.5 Parameter Passing

RTEMS uses the standard i960 family **C** parameter passing mechanism in which **G0** contains the first parameter, **G1** the second, and so on for the remaining parameters. No **RTEMS** directive requires more than six parameters.

1.6 User-Provided Routines

All user-provided routines invoked by **RTEMS**, such as user extensions, device drivers, and MPCI routines, must also adhere to these calling conventions.

1.7 Leaf Procedures

RTEMS utilizes leaf procedures internally to improve performance. This improves execution speed as well as reducing stack usage and the number of register sets which must be cached.