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

The MC68020 architecture supports a simple yet effective call and return mechanism. A subroutine is invoked via the branch to subroutine (**bsr**) or the jump to subroutine (**jsr**) instructions. These instructions push the return address on the current stack. The return from subroutine (**rts**) instruction pops the return address off the current stack and transfers control to that instruction. It is is important to note that the MC68020 call and return mechanism does not

automatically save or restore any registers. It is the responsibility of the high-level language compiler to define the register preservation and usage convention.

1.3 Calling Mechanism

All **RTEMS** directives are invoked using either a **bsr** or **jsr** instruction and return to the user application via the **rts** instruction.

1.4 Register Usage

As discussed above, the **bsr** and **jsr** instructions do not automatically save any registers. **RTEMS** uses the registers **D0**, **D1**, **A0**, and **A1** as scratch registers. These registers are not preserved by **RTEMS** directives therefore, the contents of these registers should not be assumed upon return from any **RTEMS** directive.

1.5 Parameter Passing

RTEMS assumes that arguments are placed on the current stack before the directive is invoked via the **bsr** or **jsr** instruction. The first argument is assumed to be closest to the return address on the stack. This means that the first argument of the C calling sequence is pushed last. The following pseudo-code illustrates the typical sequence used to call a **RTEMS** directive with three (3) arguments:

```
push third argument
push second argument
push first argument
invoke directive
remove arguments from the stack
```

The arguments to **RTEMS** are typically pushed onto the stack using a move instruction with a pre-decremented stack pointer as the destination. These arguments must be removed from the stack after control is returned to the caller. This removal is typically accomplished by adding the size of the argument list in bytes to the current stack pointer.

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.