ELFv2 ABI Compliance TH/TS Specification

Test Harness and Test Suite Definition

Workgroup Specification

Revision 1.5 (March 28, 2021)

www.openpowerfoundation.org

ELFv2 ABI Compliance TH/TS Specification: Test Harness and Test Suite Definition

Compliance Work Group <compliance-chair@openpowerfoundation.org>
OpenPower Foundation

Revision 1.5 (March 28, 2021)
Copyright © 2017, 2020, 2021 OpenPOWER Foundation

All capitalized terms in the following text have the meanings assigned to them in the OpenPOWER Intellectual Property Rights Policy (the "OpenPOWER IPR Policy"). The full Policy may be found at the OpenPOWER website or are available upon request.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OpenPOWER, except as needed for the purpose of developing any document or deliverable produced by an OpenPOWER Work Group (in which case the rules applicable to copyrights, as set forth in the OpenPOWER IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OpenPOWER or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THE OpenPOWER Foundation AS WELL AS THE AUTHORS AND DEVELOPERS OF THIS STANDARDS FINAL DELIVERABLE OR OTHER DOCUMENT HEREBY DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES, DUTIES OR CONDITIONS OF MERCHANTABILITY, OF FITNESS FOR A PARTICULAR PURPOSE, OF ACCURACY OR COMPLETENESS OF RESPONSES, OF RESULTS, OF WORKMANLIKE EFFORT, OF LACK OF VIRUSES, OF LACK OF NEGLIGENCE OR NON-INFRINGEMENT.

OpenPOWER, the OpenPOWER logo, and openpowerfoundation.org are trademarks or registered trademarks of OpenPOWER Foundation, Inc., registered in many jurisdictions worldwide. Other company, product, and service names may be trademarks or service marks of others.

Abstract

The purpose of the OpenPOWER ELFv2 Application Binary Interface (ABI) Compliance Test Harness and Test Suite (TH/TS) Specification is to provide the test suite requirements to be able to demonstrate OpenPOWER ELFv2 ABI compliance. It describes the tests required in the test suite and a test harness needed to execute the test suite. It also describes the successful execution of the test suite, including what it means for an optional feature to fail.

This document is a Standard Track, Work Group Specification work product owned by the Compliance Workgroup and handled in compliance with the requirements outlined in the *OpenPOWER Foundation Work Group (WG) Process* document. It was created using the *Master Template Guide* version 1.0.0. Comments, questions, etc. can be submitted to the public mailing list for this document at <openpower-elfv2abi-thts@mailinglist.openpowerfoundation.org>.

Table of Contents

Preface

1. Conventions

The OpenPOWER Foundation documentation uses several typesetting conventions.

Notices

Notices take these forms:

Note

A handy tip or reminder.

Important

Something you must be aware of before proceeding.

Warning

Critical information about the risk of data loss or security issues.

Changes

At certain points in the document lifecycle, knowing what changed in a document is important. In these situations, the following conventions will used.

- *New text will appear like this.* Text marked in this way is completely new.
- ~~Deleted text will appear like this.~~ Text marked in this way was removed from the previous version and will not appear in the final, published document.
- Changed text will appear like this. Text marked in this way appeared in previous versions but has been modified.

Command prompts

In general, examples use commands from the Linux operating system. Many of these are also common with Mac OS, but may differ greatly from the Windows operating system equivalents.

For the Linux-based commands referenced, the following conventions will be followed:

$ prompt Any user, including the `root` user, can run commands that are prefixed with the $ prompt.

 # prompt The `root` user must run commands that are prefixed with the # prompt. You can also prefix these commands with the **sudo** command, if available, to run them.

Document links

Document links frequently appear throughout the documents. Generally, these links include a text for the link, followed by a page number in parenthesis. For example, this link, Preface [iv], references the Preface chapter on page iv.

2. Document change history

This version of the guide replaces and obsoletes all earlier versions.

The following table describes the most recent changes:

Revision Date	Summary of Changes
January 10, 2017	• Rev 0.71 - Ported v 0.7 from word to docbook
January 27, 2017	• Rev 0.72 - Improved formatting
January 31, 2017	• Rev 0.90 - Public review draft
June 27, 2017	• Rev 1.00 - Approved Work Group Specification
November 10, 2020	• Rev 1.5_pre3 - Updated to ELFv2 ABI v1.5 - draft
November 17, 2020	• Rev 1.5-PRD : OpenPOWER ELFv2 Application Binary Interface (ABI) Compliance Test Harness and Test Suite - Workgroup Approved Public Review Draft
March 28, 2021	• Rev 1.5 : OpenPOWER ELFv2 Application Binary Interface (ABI) Compliance Test Harness and Test Suite Specification - Workgroup Approved Specification

1. Introduction

The purpose of the OpenPOWER ELFv2 Application Binary Interface (ABI) Compliance Test Harness and Test Suite (TH/TS) Specification is to provide the test suite requirements to be able to demonstrate OpenPOWER ELFv2 ABI compliance. It contains the following:

- Section describing the test harness needed to execute the test suite
- Section describing the tests required to be in the test suite
- Section describing the successful execution of the test suite, including what it means for an optional feature to fail

This version of this document is applicable to POWER8, POWER9, and POWER10 systems.

The input to this specification is the following specification which describes the application binary interface for OpenPOWER systems:

64-bit ELF V2 ABI Specification, Version 1.5

The ABI specification defines aspects of the platform that are required to enable interoperability of binary object files. The ABI specification document is organized into the following chapters:

1. Introduction

2. Low-level System Information

3. Object Files

4. Program Loading and Dynamic Linking

5. Libraries

6. Vector Programming Interfaces

Chapter 1 (Introduction) does not specify any interoperability requirements. Chapter 6 (Vector Programming Interfaces) is beyond the scope of this compliance specification. The rest of this compliance specification describes how to demonstrate compliance with chapters 2-5 of the ABI specification document.

While any binary application running on an OpenPOWER system must comply with the ABI aspects primarily documented in chapter 2 (Low-Level System Information) of the ELFv2 ABI specification in order to ensure interoperability, it is not possible to validate ABI compliance at the level of a single application. Instead, validation occurs at the level of the *compiler* used to build the application. This document therefore specifies conformance tests that a compiler used to build binary applications intended to run on OpenPOWER systems must adhere to.

Specifically, those conformance tests take the form of interoperability tests that validate that the binaries generated by the compiler under test are compatible with those generated by the OpenPOWER reference compiler. That reference compiler is the GNU Compiler Collection version 10. This document only specifies interoperability tests for compilers implementing the C or C++ programming languages.

Similarly, the validation of the ABI aspects documented primarily in chapter 3 (Object Files) occurs at the level of the binary *tool chain* (assembler, linker, and related tools) used to build application binary files and operate on such files. This documents specifies properties of those tools that need to be verified to demonstrate compliance with the ELFv2 ABI.

ABI aspects documented primarily in chapter 4 (Program Loading and Dynamic Linking) are implemented by the *operating environment* (kernel, dynamic loader, application start-up code). This document specifies properties of the operating environment that need to be verified to demonstrate compliance with the ELFv2 ABI.

Finally, chapter 5 (Libraries) documents ABI aspects primarily implemented by *system libraries* provided with the operating system. This document specifies properties of the system libraries that need to be verified to demonstrate compliance with the ELFv2 ABI.

1.1. Conformance to this Specification

The following lists a set of numbered conformance clauses to which any implementation of this specification must adhere in order to claim conformance to this specification (or any optional portion thereof):

1. For a compiler, the required tests in Chapter 2, “GCC Compatibility Test Harness and Test Suite” [3] must be successfully executed. In addition, the properties documents in Chapter 3, “Compiler Compatibility Requirements” [5] must be successfully verified.

2. For an assembler, the properties documented in Chapter 4, “Assembler Compatibility Requirements” [7] must be successfully verified.

3. For a tool that generates and/or operates on binary object files (which includes all assemblers and some compilers), the properties documented in Chapter 5, “Tool Chain Compatibility Requirements” [10] must be successfully verified.

4. For an operating environment, the properties documented in Chapter 6, “Operating Environment Compatibility Requirements” [12] must be successfully verified.

5. For a system library, the properties documented in Chapter 7, “System Library Compatibility Requirements” [14] must be successfully verified.

2. GCC Compatibility Test Harness and Test Suite

The purpose of this chapter is to provide the test suite requirements to be able to demonstrate OpenPOWER ELFv2 ABI compliance of binary code generated by a compiler under test. The GCC Compatibility Test Harness and Test Suite are defined for testing interoperability of binary code generated by a compiler under test with binary code generated by the reference compiler.

2.1. Test Harness to Execute the GCC Compatibility Test Suite

The Test Harness for verifying GCC Compatibility is the DejaGNU-based test suite provided with the GNU Compiler Collection source code. To prepare the test harness, install the source code of the reference compiler (GCC version 10), configure and build the reference compiler.

To install the source code of GCC version 10:

```
wget ftp://gcc.gnu.org/pub/gcc/releases/gcc-10.1.0/gcc-10.1.0.tar.xz
tar xvf gcc-10.1.0.tar.xz
```

To configure and build the reference compiler:

```
mkdir gcc-10.1.0-build
cd gcc-10.1.0-build
../gcc-10.1.0/configure --prefix=/usr --enable-languages=c,c++
                        --disable-bootstrap
make
```

Note

You may need to adapt the details of the configure line to your host system. For example, when building GCC on an Ubuntu distribution, you need to add the configure options `--enable-multiarch` `--disable-multilib`.

2.2. GCC Compatibility Test Suite Required Tests

To test interoperability of binary code generated by a compiler under test with binary code generated by the reference compiler, create a configuration file that specifies the path names to invoke the compiler under test:

`ALT_CC_UNDER_TEST`	C compiler under test.
`ALT_CXX_UNDER_TEST`	C++ compiler under test.

The test suite will assume the compiler under test supports all GCC extensions. If this is not the case for your compiler, you can additionally specify in the configuration file a list of extensions to be skipped during test:

`COMPAT_SKIPS`	Tcl list of GCC extensions to be skipped.

A typical set of GCC extensions that may need to be excluded can be:

`ATTRIBUTE`	The GCC `__attribute__` extension.
`VLA_IN_STRUCT`	The GCC variable-length array in struct extension.
`DECIMAL_FLOAT`	The GCC decimal floating-point type extension.

Creating the configuration file can be achieved by executing the following commands:

```
make -C gcc site.exp
echo 'set ALT_CC_UNDER_TEST <C compiler under test>' >>gcc/site.exp
echo 'set ALT_CXX_UNDER_TEST <C++ compiler under test>' >>gcc/site.exp
echo 'set COMPAT_SKIPS "[list {ATTRIBUTE} {VLA_IN_STRUCT}
                           {DECIMAL_FLOAT}]" >>gcc/site.exp
```

in the directory where the reference compiler was built (see above).

Once the configuration file is set up, execute at least the following test cases provided by the GCC test suite:

`compat.exp`	Generic ABI compatibility tests.
`struct-layout-1.exp`	Structure layout compatibility tests.

This can be achieved by executing the following command:

```
make check-gcc RUNTESTFLAGS="compat.exp struct-layout-1.exp"
```

in the same directory.

2.3. Successful Execution of GCC Compatibility Required Tests

After executing the make check command as shown above, the DejaGNU test suite should report successful execution of tests (PASS), and no tests as unsuccessful (FAIL), with the exception of certain tests known to fail.

The following tests are currently known to fail:

`gcc.dg/compat/struct-complex-1`	Intel-specific testcase.
`gcc.dg/compat/struct-complex-2`	Intel-specific testcase.
`g++.dg/compat/decimal/*`	Fail if the compiler does not support decimal float.

3. Compiler Compatibility Requirements

The purpose of this chapter is to provide the requirements to be able to demonstrate OpenPOWER ELFv2 ABI compliance of a compiler. This chapter does not specify any particular test harness that needs to be used to demonstrate that those requirements are satisfied; it is expected that tests verifying the listed requirements are added to whatever test harness is already in use for development of the compiler.

Note

If the compiler directly generates binary object files, the requirements in Chapter 5, “Tool Chain Compatibility Requirements” [10] apply in addition to those in this chapter.

3.1. Compiler Required Tests

To demonstrate compliance of a compiler under test with the ELFv2 ABI, verify that the compiler correctly implements the aspects of the ELFv2 ABI listed below. This can be done by writing test cases that verify these aspects, and successfully executing them.

3.1.1. Pre-defined macros

A C preprocessor that conforms to the ELFv2 ABI shall predefine the following macros to the specified values:

`__PPC__=1`	Indicate the POWER architecture.
`__powerpc__=1`	Indicate the POWER architecture.
`__powerpc64__=1`	Indicate the 64-bit POWER architecture.
`_ARCH_PWR8=1`	Indicate the POWER8 hardware architecture.
`_CALL_ELF=2`	Indicate the ELFv2 ABI.

Depending on the target processor, the following macros shall also be predefined if applicable:

`_ARCH_PWR9=1`	Indicate the POWER9 hardware architecture.
`_ARCH_PWR10=1`	Indicate the POWER10 hardware architecture.
`__MMA__=1`	Indicate support for the Matrix-Multiply Assist instructions.

If the target processor is little-endian, the following macros shall also be predefined:

```
__LITTLE_ENDIAN__=1
__BYTE_ORDER__=__ORDER_LITTLE_ENDIAN__
__FLOAT_BYTE_ORDER__=__ORDER_LITTLE_ENDIAN__
```

If the target processor is big-endian, the following macros shall also be predefined:

```
__BIG_ENDIAN__=1
__BYTE_ORDER__=__ORDER_BIG_ENDIAN__
__FLOAT_BYTE_ORDER__=__ORDER_BIG_ENDIAN__
```

3.1.2. Function call linkage

Code generated to implement direct function calls must adhere to the rules in section 2.2.1 "Function Call Linkage Protocols" of the ELFv2 ABI specification. In particular, direct calls must be implemented using a `bl` instruction directly followed by a `nop` instruction, where the `nop` may be omitted if either the `bl` is marked with a `R_PPC64_REL24_NOTOC` relocation or the callee is in the same compilation unit and is guaranteed to preserve `r2`.

3.1.3. TOC pointer usage

To enable linker-based optimizations when global data is accessed, in generated code the TOC pointer needs to be available for dereference at the point of all uses of values derived from the TOC pointer in conjunction with the `@l` modifier. In addition, all reaching definitions for a TOC-pointer-derived access must compute the same definition.

4. Assembler Compatibility Requirements

The purpose of this chapter is to provide the requirements to be able to demonstrate OpenPOWER ELFv2 ABI compliance of an assembler. This chapter does not specify any particular test harness that needs to be used to demonstrate that those requirements are satisfied; it is expected that tests verifying the listed requirements are added to whatever test harness is already in use for development of the assembler.

Note

Since an assembler will usually generate binary object files, the requirements in Chapter 5, “Tool Chain Compatibility Requirements” [10] will usually apply in addition to this chapter.

4.1. Assembler Required Tests

To demonstrate compliance of an assembler under test with the ELFv2 ABI, verify that the assembler correctly implements the aspects of the ELFv2 ABI listed below. This can be done by writing test cases that verify these aspects, and successfully executing them.

4.1.1. Pseudo operations

An assembler compliant with the ELFv2 ABI must support the following pseudo operations:

`.abiversion`	Specify the ABI version. Only ABI version 2 (to indicate the ELFv2 ABI) is required to be supported. ABI version information must be encoded into the ELF header's `e_flags` field.
`.localentry`	Specify the local entry point of a subroutine. Local entry point information must be encoded into the subroutine symbol's `st_other` field as defined in section 3.4.2 "Symbol Values" of the ELFv2 ABI specification.

4.1.2. Special symbols

An assembler compliant with the ELFv2 ABI must support the following special symbols:

`.TOC.`	The TOC base for the current object file.

4.1.3. Symbol modifiers

An assembler compliant with the ELFv2 ABI must support the following symbol modifiers:

- Modifiers to access segments of a 32-bit offset

`symbol``@l`	Low 16 bits of a 32-bit offset.
`symbol``@h`	High 16 bits of a 32-bit offset.
`symbol``@ha`	High adjusted 16 bits of a 32-bit offset.

- Modifiers to access segments of a 64-bit offset

symbol@l	Low 16 bits of a 64-bit offset.
symbol@high	High (next significant) 16 bits of a 64-bit offset.
symbol@higha	High (next significant) adjusted 16 bits of a 64-bit offset.
symbol@higher	Higher (next significant) 16 bits of a 64-bit offset.
symbol@highera	Higher (next significant) adjusted 16 bits of a 64-bit offset.
symbol@highest	Highest (most significant) 16 bits of a 64-bit offset.
symbol@highesta	Highest (most significant) adjusted 16 bits of a 64-bit offset.
symbol@higher34	Bits 34-49 of a 64-bit offset.
symbol@highera34	Adjusted bits 34-49 of a 64-bit offset.
symbol@highest34	Bits 50-63 of a 64-bit offset.
symbol@highesta34	Adjusted bits 50-63 of a 64-bit offset.

- Modifiers related to GOT and TOC

symbol@got	Offset from .TOC. of the GOT slot storing the value of *symbol*.
symbol@got@l *symbol*@got@h *symbol*@got@ha	Segments of *symbol*@got.
symbol@toc	Offset from .TOC. of *symbol*.
symbol@toc@l *symbol*@toc@h *symbol*@toc@ha	Segments of *symbol*@toc.

- Modifiers related to thread-local storage

symbol@tprel	Offset of *symbol* relative to the thread pointer.
symbol@tprel@l *symbol*@tprel@h *symbol*@tprel@ha *symbol*@tprel@high *symbol*@tprel@higha *symbol*@tprel@higher *symbol*@tprel@highera *symbol*@tprel@highest *symbol*@tprel@highesta	Segments of *symbol*@tprel.
symbol@got@tprel	Offset from .TOC. of the GOT slot storing the value *symbol*@tprel.
symbol@got@tprel@l *symbol*@got@tprel@h *symbol*@got@tprel@ha	Segments of *symbol*@got@tprel.
symbol@dtprel	Offset of *symbol* relative to the dynamic thread vector value of its defining module.
symbol@dtprel@l *symbol*@dtprel@h *symbol*@dtprel@ha *symbol*@dtprel@high *symbol*@dtprel@higha *symbol*@dtprel@higher *symbol*@dtprel@highera *symbol*@dtprel@highest *symbol*@dtprel@highesta	Segments of *symbol*@dtprel.
symbol@got@dtprel	Offset from .TOC. of the GOT slot storing the value *symbol*@dtprel.
symbol@got@dtprel@l *symbol*@got@dtprel@h *symbol*@got@dtprel@ha	Segments of *symbol*@got@dtprel.
symbol@got@tlsgd	Offset from .TOC. of the GOT slot storing the general-dynamic tls_index for *symbol*.

symbol@got@tlsgd@l *symbol*@got@tlsgd@h *symbol*@got@tlsgd@ha	Segments of *symbol*@got@tlsgd.
symbol@got@tlsld	Offset from .TOC. of the GOT slot storing the local-dynamic tls_index for *symbol*.
symbol@got@tlsld@l *symbol*@got@tlsld@h *symbol*@got@tlsld@ha	Segments of *symbol*@got@tlsld.
symbol@dtpmod	Dynamic thread vector index of the module defining *symbol*.
symbol@tls	Replaced by the thread pointer register; used in instructions computing the address of a thread-local symbol.

- Modifiers related to PC-relative addressing

symbol@pcrel	34-bit PC-relative offset of *symbol*.
symbol@got@pcrel	34-bit PC-relative offset of the GOT slot storing the value of *symbol*.
symbol@got@tprel@pcrel	34-bit PC-relative offset of the GOT slot storing the value of *symbol*@tprel.
symbol@got@dtprel@pcrel	34-bit PC-relative offset of the GOT slot storing the value of *symbol*@dtprel.
symbol@got@tlsgd@pcrel	34-bit PC-relative offset of the GOT slot storing the general-dynamic tls_index for *symbol*.
symbol@got@tlsld@pcrel	34-bit PC-relative offset of the GOT slot storing the local-dynamic tls_index for *symbol*.
symbol@tls@pcrel	Replaced by the thread pointer register; used in instruction sequences involving PC-relative addressing to compute the address of a thread-local symbol.

- Modifiers related to the function calling convention

symbol@localentry	Address of the local entry point for a function symbol.
symbol@notoc	Used with a direct call instruction to indicate that the TOC pointer register is not initialized.

5. Tool Chain Compatibility Requirements

The purpose of this chapter is to provide the requirements to be able to demonstrate OpenPOWER ELFv2 ABI compliance of a tool under test that either generates or operates on binary object files. This could be an assembler, disassembler, linker, or similar tool, or it could be a compiler that directly generates binary object files. This chapter does not specify any particular test harness that needs to be used to demonstrate that those requirements are satisfied; it is expected that tests verifying the listed requirements are added to whatever test harness is already in use for development of the tool.

5.1. Tool Chain Required Tests

To demonstrate compliance of an object file tool under test with the ELFv2 ABI, verify that the tool correctly implements the aspects of the ELFv2 ABI listed below. This can be done by writing test cases that verify these aspects, and successfully executing them.

5.1.1. ELF Header

The ELF header of generated object files must be marked as follows to indicate compliance with the ELFv2 ABI:

1. The file class member of the ELF header identification array, `e_ident[EI_CLASS]`, identifies the ELF file as 64-bit encoded by holding the value `ELFCLASS64`.

2. Processor identification resides in the ELF header's `e_machine` member, and must have the value `EM_PPC64`, defined as the value `21`.

3. The ELF header's `e_flags` member holds the bit flag `2` to indicate use of the ELFv2 ABI.

5.1.2. Symbol table

The ELFv2 ABI uses the three most-significant bits in the symbol st_other field to specify the number of bytes between a function's global entry point and local entry point. The global entry point is used when it is necessary to set up the TOC pointer (`r2`) for the function. The local entry point is used when `r2` is known to already be valid for the function. A value of zero in these bits asserts that the function does not use `r2` as TOC pointer but preserves its value. A value of one in these bits asserts that the function does not use `r2` as TOC pointer and does not guarantee to preserve its value. Other values in these bits have the semantics as defined in section 3.4 of the ELFv2 ABI specification document.

5.1.3. Relocation records

The 64-bit OpenPOWER Architecture uses Elf64_Rela relocation entries exclusively. Tools compliant with the ELFv2 ABI that generate object files must only generate relocation types documented in section 3.5 of the ELFv2 ABI specification document. Tools compliant with the ELFv2 ABI that operate on object files must support all relocation types documented in section 3.5 of the ELFv2 ABI specification document.

5.1.4. Save and restore routines

The linker must make available all of the save and restore routines described in Section 2.3.3 "Register Save and Restore Functions" of the ELFv2 ABI specification. These functions can either be provided in a utility library that is linked by the linker to each module, or the functions can be synthesized by the linker as necessary to resolve symbols.

5.1.5. Local function calls

The linker must resolve any local function call between functions that differ in their usage of the TOC register to point to a stub in accordance with section 2.2.1 "Function Call Linkage Protocol" of the ELFv2 ABI specification.

In particular, when a function using `r2` as the TOC pointer register, as indicated by the `R_PPC64_REL24` relocation on the call site, calls a function that does not preserve `r2`, as indicated by the `st_other` field in the symbol table entry for the target symbol, the linker must create a stub that ensures the TOC pointer is saved to the TOC save slot on the stack, unless the calling routine indicates that it has already performed this operation by using the `R_PPC64_TOCSAVE` relocation on the `nop` instruction following the call. In addition, the linker must modify the `nop` instruction following the call to an instruction that restores the TOC pointer from the TOC save slot on the stack.

In the opposite case, when a function that does not use `r2` as the TOC pointer register, as indicated by the `R_PPC64_REL24_NOTOC` relocation on the call site, calls a function that uses the TOC pointer register, as indicated by the `st_other` field in the symbol table entry for the target symbol, the linker must create a stub that calls the target function's global entry point while that entry point address is present in the `r12` register.

5.1.6. Dynamic linking

When building dynamically linked applications or shared libraries, the linker must generate a dynamic section, global offset table, and procedure linkage table in accordance with the requirements in section 4.2.2 "Dynamic Section", 4.2.3 "Global Offset Table", and 4.2.5 "Procedure Linkage Table" of the ELFv2 ABI specification, respectively.

5.1.7. External function calls

When building dynamically linked applications or shared libraries, the linker must resolve any function call targeting a function that may be defined in another module to point to a PLT stub in accordance with the requirements in section 4.2.5.3 "Procedure Linkage Table" of the ELFv2 ABI specification. Specifically, this stub must ensure the TOC pointer is saved to the TOC save slot on the stack, unless the calling routine indicates that it has already performed this operation by using the `R_PPC64_TOCSAVE` relocation on the `nop` instruction following the call. In addition, the linker must modify the `nop` instruction following the call to an instruction that restores the TOC pointer from the TOC save slot on the stack, unless the call is marked using the `R_PPC64_REL24_NOTOC` relocation.

5.1.8. Function addresses

When building dynamically linked applications or shared libraries, the linker must must follow the rules specified in section 4.2.4 "Function Addresses" of the ELFv2 ABI specification when resolving function addresses.

6. Operating Environment Compatibility Requirements

The purpose of this chapter is to provide the requirements to be able to demonstrate OpenPOWER ELFv2 ABI compliance of an OpenPOWER operating environment. The operating environment may include an operating system kernel, a dynamic loader, and application start-up code provided by the operating environment. This chapter does not specify any particular test harness that needs to be used to demonstrate that those requirements are satisfied; it is expected that tests verifying the listed requirements are added to whatever test harness is already in use for development of the operating environment component.

6.1. Operating Environment Required Tests

To demonstrate compliance of an operating environment with the ELFv2 ABI, verify that the operating environment correctly implement the aspects of the ELFv2 ABI listed below. This can be done by writing test cases that verify these aspects, and successfully executing them.

6.1.1. Process initialization

When the kernel or dynamic loader initially transfers control to a newly loaded application, the following registers must be set to defined values:

`r1`	Initial stack pointer.
`r3`	Number of arguments (`argc`).
`r4`	Pointer to the array of argument pointers (`argv`).
`r5`	Pointer to the array of environment pointers (`envp`).
`r6`	Pointer to the auxillary vector (`auxv`); see below.
`r7`	Termination function pointer.
`r12`	Global entry point address of the application start routine.
`fpscr`	Zero.
`vscr`	Zero.

6.1.2. Auxiliary vector

The auxiliary vector conveys information from the operating system to the program. This vector is an array of structures as defined in Section 4.1.2.3 "Auxiliary Vector" of the ELFv2 ABI. To comply with the ABI, at least the following auxiliary vector elements must be present:

```
AT_ENTRY
AT_PLATFORM
AT_BASE_PLATFORM
AT_HWCAP
AT_HWCAP2
```

6.1.3. Application start-up code

The run-time that gets control at application start-up is responsible for:

- Creating the first stack frame.
- Initializing the first stack frame's back chain pointer to `NULL`.
- Allocating and initializing TLS storage.
- Initializing the thread control block (TCB) and dynamic thread vector (DTV).
- Initializing any `__thread` variables.
- Setting `r13` for the initial process thread.

This initialization must be completed before any library initialization codes are run and before control is transferred to the main program.

The main program must be called with the following arguments: `argc`, `argv`, `envp`, and `auxv`.

6.1.4. Dynamic loading

When loading dynamically linked applications, the dynamic loader must recursively load all dependent shared libraries and resolve dynamic relocations in accordance with the requirements in section 3.5 Relocation Types of the ELFv2 ABI specification. Specifically, the dynamic loader must support at least the following dynamic relocation types:

```
R_PPC64_RELATIVE
R_PPC64_JMP_SLOT
R_PPC64_IRELATIVE
R_PPC64_JMP_IREL
R_PPC64_GLOB_DAT
R_PPC64_COPY
R_PPC64_DTPMOD64
R_PPC64_DTPREL64
R_PPC64_TPREL64
```

In addition, the dynamic linker must initialize the contents of the Procedure Linkage Table in accordance with the requirements in section 4.2.5 "Procedure Linkage Table" of the ELFv2 ABI specification.

The dynamic loader may assume that a dynamic section as specified in section 4.2.2 "Dynamic Section" of the ELFv2 ABI is present in each dynamically linked application or shared library.

7. System Library Compatibility Requirements

The purpose of this chapter is to provide the requirements to be able to demonstrate OpenPOWER ELFv2 ABI compliance of an OpenPOWER system library. The system library is usually provided with an operating system or operating environment, but may in certain cases also provided as a separate stand-alone product. This chapter does not specify any particular test harness that needs to be used to demonstrate that those requirements are satisfied; it is expected that tests verifying the listed requirements are added to whatever test harness is already in use for development of the system library.

7.1. System Library Required Tests

To demonstrate compliance of a system library with the ELFv2 ABI, verify that the system library correctly implement the aspects of the ELFv2 ABI listed below. This can be done by writing test cases that verify these aspects, and successfully executing them.

7.1.1. Malloc routine return pointer alignment

The `malloc()` routine must always return a pointer with the alignment of the largest alignment needed for loads and stores of the built-in data types. This is currently 16 bytes.

7.1.2. Library handling of limited-access bits in registers

Standary library functions must comply with the requirements for the handling of limited-access bits in certain registers defined in Section 2.2.1.2 "Limited-Access Bits" of the ELFv2 ABI specification.

7.1.3. Standard header files

The standard library header files must provide all data types defined in Section 5.1.3 "Types Defined in Standard Header" of the ELFv2 ABI specification.

Appendix A. OpenPOWER Foundation overview

The OpenPOWER Foundation was founded in 2013 as an open technical membership organization that will enable data centers to rethink their approach to technology. Member companies are enabled to customize POWER CPU processors and system platforms for optimization and innovation for their business needs. These innovations include custom systems for large or warehouse scale data centers, workload acceleration through GPU, FPGA or advanced I/O, platform optimization for SW appliances, or advanced hardware technology exploitation. OpenPOWER members are actively pursing all of these innovations and more and welcome all parties to join in moving the state of the art of OpenPOWER systems design forward.

To learn more about the OpenPOWER Foundation, visit the organization website at openpowerfoundation.org.

A.1. Foundation documentation

Key foundation documents include:

- *Bylaws of OpenPOWER Foundation*
- *OpenPOWER Foundation Intellectual Property Rights (IPR) Policy*
- *OpenPOWER Foundation Membership Agreement*
- *OpenPOWER Anti-Trust Guidelines*

More information about the foundation governance can be found at openpowerfoundation.org/about-us/governance.

A.2. Technical resources

Development resouces fall into the following general categories:

- Technical Steering Committee
- Foundation work groups
- OpenPOWER Ready documentation, products, and certification criteria
- Resource Catalog

  To find all OpenPOWER resources of the following types, select the specificied combination of **Resource Type**/**Main Category**/**Sub-category** in the Resource Catalog:

Specifications	`Developer Resources` / `OpenPOWER Documents` / `Specifications`
Work Group Notes	`Developer Resources` / `OpenPOWER Documents` / `Work Group Notes`

Cloud development virtual machines	`Developer Resources` / `Software Developer Cloud Resources` / <empty>
Developer Tools	`Developer Resources` / `Developer Tools` / <empty>

Note

Use the **Search** field to focus your search using key words or phrases for specific resources.

A.3. Contact the foundation

To learn more about the OpenPOWER Foundation, please use the following contact points:

- General information -- `<info@openpowerfoundation.org>`
- Membership -- `<membership@openpowerfoundation.org>`
- Technical Work Groups and projects -- `<tsc-chair@openpowerfoundation.org>`
- Events and other activities -- `<admin@openpowerfoundation.org>`
- Press/Analysts -- `<press@openpowerfoundation.org>`

More contact information can be found at openpowerfoundation.org/get-involved/contact-us.