The V8+ Technical Specification

Sun Microsystems Computer Company
A Sun Microsystems, Inc. Business
2550 Garcia Avenue
Mountain View, CA 94043 USA
415 960-1300 fax 415 969-9131

Part No.: 802-7447-10
Revision 50, August 1996

Contents

Tables

Preface

The V8+ Technical Specification is intended to give ISVs and other SPARC™ developers information about SPARC V8+'s restrictions and features.

We expect usage of V8+ objects to fall into one of three categories:

1. Platform-specific libraries, such as graphics libraries, and DDX drivers that need to use V8+ facilities in order to meet performance goals. This includes libraries that interpose on the standard multiply, divide, and storage copy and fill routines, so that vanilla V8 applications can transparently take advantage of performance- enhancing V8+ features.

2. End-user-written applications, where the end user compiles for a specific platform, wants every last bit of performance possible, and portability of binaries across different SPARC-based systems, are not a consideration.

3. Certain third-party applications where the ISV has seriously considered the costs and benefits of maintaining two binaries (V8 and V8+), and has decided that the extra performance outweighs the maintenance cost. Note that a V8+ application is ineligible for SPARCware branding.

Before You Read This Book

It is assumed that you are familiar with the contents of these books:

- *SPARC Architecture Manual*, Version 9, SPARC International, SAV09R1429309, ISDN 0-13-099227-5.
- *System V Application Binary Interface*, ISBN 0-13-877598-2

- *System V Application Binary Interface SPARC Processor Supplement*, ISBN 0-13-877630-X
- *Linker and Libraries Guide* (part number 802-6319-10)

How This Book Is Organized

The chapter titles and subject matter are as follows

Chapter 1, "V8+ Introduction", is an overview of the V8+ specification.

Chapter 2, "Differences", is comprised of definitions and discussions of the differences between SPARC V8+ and SPARC V8; also, the SPARC V9 and UltraSPARC™ features, which comprise the V8+ specification.

Appendix A, "UltraSPARC-I-Specific Changes", is a discussion of these changes made to V8 for V8+.

Typographic Conventions

The following table describes the uses of the type faces used in this book.

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% You have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`machine_name% su` `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type `rm` *filename.*
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in the *User's Guide.* These are called *class* options. You *must* be root to do this.

Shell Prompts

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

Shell	Prompt
C shell	`machine_name%`
C shell superuser	`machine_name#`
Bourne shell and Korn shell	`$`
Bourne shell and Korn shell superuser	`#`

Related Documents

The following documents contain topics that relate to the information in *The V8+ Technical Specification*.

- *System V Application Binary Interface*, ISBN 0-13-877598-2
- *System V Application Binary Interface SPARC Processor Supplement*, ISBN 0-13-877630-X

  Referred to as 'SysV SPARC ABI'.
- *SPARC Architecture Manual*, Version 8, SPARC International, SAV080SI9106

  Referred to as SPARC V8.
- *SPARC Architecture Manual*, Version 9, SPARC International, SAV09R1419309.

  Referred to as SPARC V9.
- *UltraSPARC-1 User's Manual*, Revision 1.4, January 1996, STP1030-UG

  Referred to as UltraSPARC-I.
- *Solaris 2.x Reference Manual*
- *Linker and Libraries Guide* (part number 802-6319-10).

Sun™ Welcomes Your Comments

Please use the *Reader Comment Card* that accompanies this document. We are interested in improving our documentation and welcome your comments and suggestions.

If a card is not available, you can email or fax your comments to us. Please include the part number of your document in the subject line of your email or fax message.

- Email: `smcc-docs@sun.com`
- Fax: SMCC Document Feedback
  1-415-786-6443

Ordering Sun Documents

SunDocs™ is a distribution program for Sun Microsystems™ technical documentation. Easy, convenient ordering and quick delivery is available from SunExpress™. You can find a full listing of available documentation on the World Wide Web: `http://www.sun.com/sunexpress/`

Country	Telephone	Fax
United States	1-800-873-7869	1-800-944-0661
United Kingdom	0-800-89-88-88	0-800-89-88-87
France	05-90-61-57	05-90-61-58
Belgium	02-720-09-09	02-725-88-50
Luxembourg	32-2-720-09-09	32-2-725-88-50
Germany	01-30-81-61-91	01-30-81-61-92
The Netherlands	06-022-34-45	06-022-34-46
Sweden	020-79-57-26	020-79-57-27
Switzerland	155-19-26	155-19-27
Japan	0120-33-9096	0120-33-9097

V8+ Introduction 1

V8+ is the name for the usage of SPARC Version 9 and UltraSPARC features in a SPARC Version 8 environment.

Note – V8+ is not being promoted as a new ABI, particularly as it is not a system-independent set of interfaces.

The V8+ environment offers SPARC V9 and UltraSPARC features and performance enhancements. A vanilla V8 ABI user-level binary benefits from most of these features and performance enhancements if dynamically linked. This is due to a new mechanism called Auxiliary Filters which interposes platform-specific library routines.

For more information about Auxiliary Filters, refer to *Linker and Libraries Guide*.

Due to the nature of these UltraSPARC platform-specific features, management of extra SPARC V9 and UltraSPARC-specific register state is required. This management of the extra register set requires interface changes in a few areas. These are discussed in Chapter 2, “Differences”.

Note – Unless otherwise specified, interfaces established by this document have a classification of Unstable.

A V8+ object is an ELF-executable or shared object that violates the SPARC V8 ABI. It does this by issuing instructions specific to SPARC V9 or UltraSPARC that are not available in SPARC V8, and/or by using registers specific to SPARC V9 or UltraSPARC that are not available in SPARC V8.

Note – General usage of V8+ objects is strongly discouraged.

We expect usage of V8+ objects to fall into one of three categories:

1. Platform-specific libraries, such as graphics libraries, and DDX drivers, that need to use V8+ facilities in order to meet performance goals. This includes libraries that interpose on the standard multiply, divide, and storage copy and fill routines, so that vanilla V8 applications can transparently take advantage of performance- enhancing V8+ features.

2. End-user-written applications, where the end user compiles for a specific platform, wants every last bit of performance possible, and portability of binaries across different SPARC-based systems, are not a consideration.

3. Certain third-party applications where the ISV has seriously considered the costs and benefits of maintaining two binaries (V8 and V8+), and has decided that the extra performance outweighs the maintenance cost. Note that a V8+ application is ineligible for SPARCware branding.

Differences 2

This chapter includes descriptions of the V8+ specification features and their differences from the V8 specification.

2.1 Low-Level System Information

This section should be read in conjunction with SysV SPARC ABI Chapter 3.

2.1.1 Permitted Instructions

You may use all non-privileged V8 instructions and all non-privileged V9 instructions, except as listed below. The exceptions to these rules are based on the fact that the upper 32 bits of the *in* and *local* registers are not preserved in user stack frames. Thus, their contents are unpredictable, even after execution of an instruction intended to set them. A violation of these restrictions produces unpredictable results. The kernel preserves the upper 32 bits of the *out* and *global* registers on a trap or interrupt.

Note – *Out* registers are volatile across function calls. Even if the function being called is known not to use them, it could perform a SAVE, causing the *out*s to become *in*s, and putting their upper 32 bits at risk.

Note – Other restrictions limit the *global* registers which may be used; see below.

The section numbers in this table refer to *SPARC Architecture Manual*, Version 9.

Table 2-1 V9 Instructions Allowed in V8+

Instructions	Section Number	Restriction
BPr	A.3	rsl must be a *global* (%g) or *out* (%o) register.
FMOVr	A.33	
MOVr	A.35	
BPcc	A.7	Use of %xcc is unpredictable unless the operation setting the condition codes uses *global* or *out* registers as sources.
FMOVcc	A.32	
MOVcc	A.34	
TCC	A.60	
CASXA	A.9	rs2 and rd must be *global* or *out* registers.
LDX	A.27	rd must be a *global* or *out* register.
LDXA	A.28	
STX	A.53	
STXA	A.54	
MULX	A.36	rs1, rs2 (if used), and rd must be *global* or *out* registers.
SDIVX		
UDIVX		
POPC	A.40	rs2 (if used) must be a *global* or *out* register.
RDTICK	A.43	rd must be a *global* or *out* register.
RDASR	A.43	rd must be a *global* or *out* register if the ASR is wider than 32 bits.
SLLX	A.48	With shift counts less than 32: rsl and rd must be *global* or *out* registers.
SRLX		
SRAX		
SLLX	A.48	With shift counts greater than or equal to 32: rd must be a *global* or *out* register.

Table 2-1 V9 Instructions Allowed in V8+ *(Continued)*

Instructions	Section Number	Restriction
SRLX	A.48	With shift counts greater than or equal to 32: `rs1` must be a *global* or *out* register.
SRAX		
WRASR	A.62	`rs1` and `rs2` (if used) must be *global* or *out* registers if the ASR is wider than 32 bits.

2.1.2 Processor-Specific Opcodes in V8+ Objects

A V8+ object may contain processor-specific opcodes. It is strongly recommended that such opcodes not be compiler-generated, but used in assembler routines and in-line templates. Processor-specific opcodes should only be used by processor-specific shared libraries. They should not appear in application executables and shared objects that are intended to be portable to other systems.

2.1.3 C "long long"

Compare with SysV SPARC ABI, Figure 3-1, p.3-2.

A C long long is an 8-byte quantity, aligned on an 8-byte boundary. It usually occupies the low-order halves of a register pair: `LDD` and `STD` (or two `LD`s and `ST`s) are usually used to move a long long; not `LDX`/`STX`. If you use `LDX`/`STX`, you must use a *global* or *out* register.

2.1.4 Function Register Volatility

Compare with SysV SPARC ABI, Figure 3-15, p.3-11.

- `%g5`, *global* 5, is volatile across function calls and may be used by an application.
- `%d32` (`%q32`) through `%d62` (`%q60`) and `%ccr` are volatile across function calls.
- `%asi` by default holds ASI_PRIMARY_NOFAULT. If modified, it must be restored to the default value before calling another function or returning.
- `%fprs` is reserved for use by the system.
- `%tick` is reserved.

2.1.5 Function Registers with Unassigned Roles

Compare with SysV SPARC ABI, p.3-16.

The following have no specified role in the calling sequence:

- "`%f32`" (`%d32`) through "`%f63`" (`%d62`)
- `%g5`; no longer reserved for system software
- `%ccr` register

The `%asi` register will contain ASI_PRIMARY_NOFAULT on function entry and must contain that value on function call or return. `%tick` is reserved.

2.1.6 State Register Initialization

Register	Initial State
CCR(icc,xcc)	Unspecified
`ASI`	`ASI_PRIMARY_NOFAULT`

Compare with SysV SPARC ABI, p.3-31.

2.1.7 Auxiliary Vector

Compare with SysV SPARC ABI, p.3-5.

V8+ uses the `AT_FLAGS` entry in the auxiliary vector to communicate platform support between the kernel and the dynamic linker. See Section 2.2.4, "Run-time Checking," on page 2-7, for more information.

2.2 Object Files

This section should be read in conjunction with SysV SPARC ABI, Chapter 4.

2.2.1 ELF Header

Compare with SysV SPARC ABI, p.4-1.

A binary that uses SPARC V9 and UltraSPARC features, referred to as a V8+ binary, must contain an indication of this in its ELF header since it will not execute in a vanilla V8 environment.

Processor identification resides in the `e_machine` field of the ELF header. A V8+ binary is identified by an `e_machine` field value of `EM_SPARC32PLUS`, which has the value 18. Note that the `e_ident[EI_CLASS]`field of the ELF header for a V8+ binary is `ELFCLASS32`.

SPARC V9 allows implementations to provide implementation-specific extensions. Utilization of such extensions, as well as utilization of generic V9 features, is indicated in the `e_flags` field. These `e_flags` bit definitions have been added to `<sys/elf_SPARC.h>`. `EF_SPARC_32PLUS_MASK` bits are reserved for indicating V8+ typeness.

```
EF_SPARC_32PLUS_MASK 0xffff00  /* bits indicating V8+ type */
EF_SPARC_32PLUS      0x000100  /* generic V8+ features */
EF_SPARC_SUN_US1     0x000200  /* Sun UltraSPARC1 extensions */
EF_SPARC_HAL_R1      0x000400  /* HAL R1 extensions */
```

Based on the preceding, the settings for these processor-specific fields of the ELF header for a V8+ binary are as follows.

Table 2-2 Processor-Specific ELF Header Field Settings for a V8+ Object

Field	Setting
`e_ident[EI_CLASS]`	`ELFCLASS32`
`e_ident[EI_DATA]`	`ELFDATA32MSB`
`e_machine`	`EM_SPARC32PLUS`
`e_flags`	`EF_SPARC_32PLUS` OR'd with zero or more of the other bits within `EF_SPARC_32PLUS_MASK`

2.2.2 Compile-time Setting

This is an example of the various flags a compilation system might support to allow for building V8 and V8+ binaries. The example names of the flags are only for this description and may not correspond to any compiler, existing or planned.

Table 2-3 Examples of Supported Flags

Flag	Purpose
default mode	To build a V8 binary
`-v8plus` flag[1]	Specifies building a generic V8+ binary
`-v8plus_sun_us1` flag[2]	Specifies building a V8+ binary with Sun UltraSPARC 1 extensions
`-v8plus_hal_r1` flag	Specifies building a V8+ binary with HAL R1 extensions

1. This corresponds to the `-xarch=v8plus` flag of the Sun C 4.0 compiler.
2. This corresponds to the `-xarch=v8plusa` flag of the Sun C 4.0 compiler.

This table describes the setting of the `e_machine` and `e_flags` fields of the resulting binary based on the compilation flags.

Table 2-4 `e_machine` and `e_flags` Settings Based on Compilation Flags

	`e_machine`	`e_flags`
default	`EM_SPARC`	`0`
`-v8plus`[1]	`EM_SPARC32PLUS`	`EF_SPARC_32PLUS`
`-v8plus_sun_us1`[2]	`EM_SPARC32PLUS`	`EF_SPARC_32PLUS|EF_SPARC_SUN_US1`
`-v8plus_hal_r1`	`EM_SPARC32PLUS`	`EF_SPARC_32PLUS|EF_SPARC_HAL_R1`

1. This corresponds to the `-xarch=v8plus` flag of the Sun C 4.0 compiler.
2. This corresponds to the `-xarch=v8plusa` flag of the Sun C 4.0 compiler.

2.2.3 Static Linking

The link editor, `ld`(1), links any combination of V8 and V8+ objects:

- If all linked objects contain `e_machine` fields of `EM_SPARC`, the resulting linked object's `e_machine` field will be `EM_SPARC`.
- If any of the linked objects contain an `e_machine` field of `EM_SPARC32PLUS`, then the resulting linked object's `e_machine` field will be `EM_SPARC32PLUS`.

- If e_machine is EM_SPARC32PLUS then the e_flags EF_SPARC_32PLUS bit must also be set.
- Any other e_machine field value is an error.

When linking objects, ld(1) ORs all the e_flags fields of the linked objects to produce the resulting linked object's e_flags value.

2.2.4 Run-time Checking

When executing a binary, the kernel and the runtime linker perform validity checking of the processor-specific ELF header fields. The kernel provides the runtime linker with information on this platform's V8+ binary support by way of the AT_FLAGS auxiliary vector. The kernel will construct the AT_FLAGS information based on the capabilities of the platform's cpu module(s), using the same bits as in e_flags.

The kernel checks the processor-specific fields of the ELF header. If e_machine is EM_SPARC32PLUS then EF_SPARC_32PLUS must be set in e_flags. If e_machine is EM_SPARC32PLUS then the only acceptable set e_flags bits within EF_SPARC_32PLUS_MASK are those allowed based on this platform's current cpu module support.

The runtime linker performs checking similar to the kernel, using the AT_FLAGS information to validate any dynamically linked objects.

2.2.5 Relocation Types

Compiler enhancements to take advantage of SPARC V9 and UltraSPARC via supporting additional instructions requires the usage of new relocation types. These new relocation types are generated by the assembler and supported by the linkers. The relocation types are described in *Linker and Libraries Guide.*

2.3 Changes to core(4)

A V8+ core file has the same structure as a V8 core file, with the addition of new NOTE types 4 and 5, described in the Solaris 2.5 (or later) core(4) man page.

2.4 *Changes to* `ucontext`*(5)*

V8+ adds a new mechanism for associating extra register state with the `ucontext` structure. This is accomplished by using entries in the `uc_mcontext filler` array. The following structure is put at the beginning of the `uc_mcontext` filler array, beginning at what used to be `uc_mcontext.filler[0]` (and is now `uc_mcontext.xrs`).

```
typedef struct {
    unsigned int xrs_id;
    caddr_t xrs_ptr;
} xrs_t;
```

If the `xrs_id` field is equal to the string “xrs”, in other words 0x78727300, then `xrs_ptr` contains a valid pointer to a `prxregset_t` (see `/usr/include/sys/procfs.h`).

2.5 *Changes to* `proc`*(4)*

An executing V8+ object has additional register state consisting of additional floating-point registers, all 64 bits of the global and out registers, and the address space identifier registers. This state may be examined with `proc`(4) commands `PIOCGXREGSIZE`, `PIOCGXREG`, `PIOCSXREG`, described in the Solaris 2.5 (or later) `proc`(4) `man` page.

UltraSPARC-I-Specific Changes A

This appendix discusses UltraSPARC-I-specific changes made to V8 for V8+.

UltraSPARC defines user-accessible registers GSR, PCR, and PIC. GSR is used by certain of the UltraSPARC Graphics Instructions. PCR and PIC are performance-monitoring registers. They are all accessed by way of `RDASR` and `WRASR` instructions.

`%gsr` is volatile across function calls, plays no role in the calling sequence, and may be used by a called function without saving its value before changing it and without restoring its value before returning. Its initial state is unspecified.

`%pcr` and `%pic` are reserved.

The `pr_filler` array inside the `prxregset_t` structure is used to hold platform-dependent extra register state. For UltraSPARC-based platforms, `pr_filler[0]`contains the graphics state register.

`sysinfo`(2) can be invoked with the command `SI_PLATFORM` to identify the current platform, so that a program can determine how to interpret the contents of the `pr_filler` array.

A

Index

V

W

X