5.2.1 Global R Registers (A1)

Registers R[0]–R[7] refer to a set of eight registers called the *global* registers (labeled g0 through g7). At any time, one of *MAXGL* +1 sets of eight registers is enabled and can be accessed as the current set of global registers. The currently enabled set of global registers is selected by the GL register. See *Global Level Register (glP) (PR 16)* on page 99.

Global register zero (G0) always reads as zero; writes to it have no software-visible effect.

5.2.2 Windowed R Registers (A1)

A set of 24 R registers that is visible as R[8]–R[31] at any given time is called a "register window". The registers that become R[8]–R[15] in a register window are called the *out* registers of the window. Note that the *in* registers of a register window become the *out* registers of an adjacent register window. See TABLE 5-1 and FIGURE 5-2.

The names *in*, *local*, and *out* originate from the fact that the *out* registers are typically used to pass parameters from (out of) a calling routine and that the called routine receives those parameters as its *in* registers.

TABLE 5-1 Window Addressing

Windowed Register Address	R Register Address
in[0] – *in*[7]	R[24] – R[31]
local[0] – *local*[7]	R[16] – R[23]
out[0] – *out*[7]	R[8] – R[15]
global[0] – *global*[7]	R[0] – R[7]

V9 Compatibility Note In the SPARC V9 architecture, the number of 16-register windowed register sets, *N_REG_WINDOWS*, ranges from 3 to 32 (impl. dep. #2-V8). The maximum global register set index in the UltraSPARC Architecture, *MAXGL*, ranges from 2 to 15. The number of implemented global register sets is *MAXGL* + 1. The total number of R registers in a given UltraSPARC Architecture implementation is:

$$(N_REG_WINDOWS \times 16) + ((MAXGL + 1) \times 8)$$

Therefore, an UltraSPARC Architecture processor may contain from 72 to 640 R registers.

The current window in the windowed portion of R registers is indicated by the current window pointer (CWP) register. The CWP is decremented by the RESTORE instruction and incremented by the SAVE instruction.

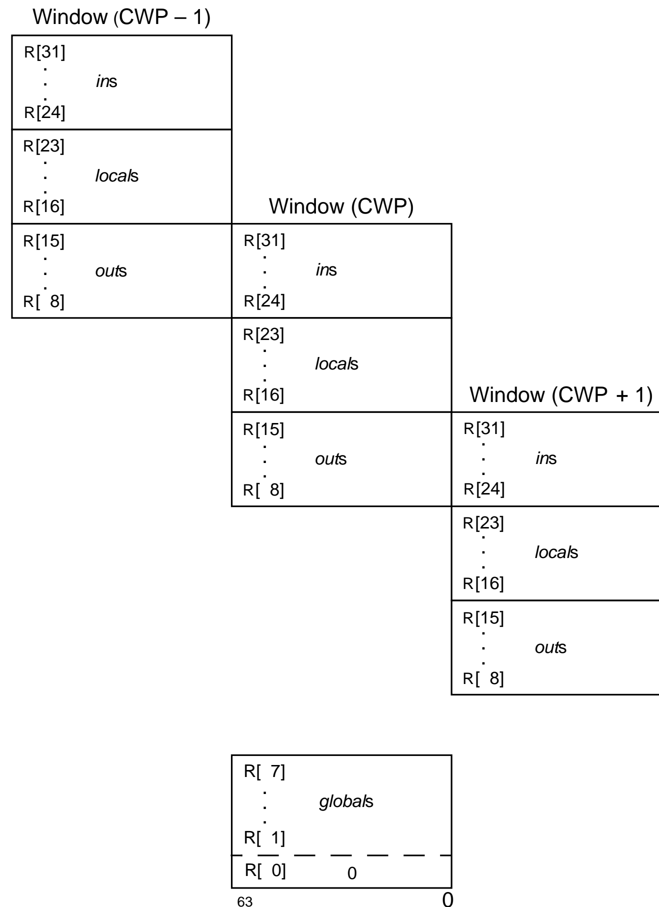


FIGURE 5-2 Three Overlapping Windows and Eight Global Registers

Overlapping Windows. Each window shares its *in*s with one adjacent window and its *out*s with another. The *out*s of the CWP – 1 (**modulo** *N_REG_WINDOWS*) window are addressable as the *in*s of the current window, and the *out*s in the current window are the *in*s of the CWP + 1 (**modulo** *N_REG_WINDOWS*) window. The *local*s are unique to each window.

Register address o, where $8 \le o \le 15$, refers to exactly the same *out* register before the register window is advanced by a SAVE instruction (CWP is incremented by 1 (**modulo** *N_REG_WINDOWS*)) as does register address $o+16$ after the register window is advanced. Likewise, register address i, where $24 \le i \le 31$, refers to exactly the same

in register before the register window is restored by a RESTORE instruction (CWP is decremented by 1 (**modulo** *N_REG_WINDOWS*)) as does register address $i-16$ after the window is restored. See FIGURE 5-2 on page 51 and FIGURE 5-3 on page 53.

To application software, the virtual processor appears to provide an infinitely-deep stack of register windows.

Programming Note | Since the procedure call instructions (CALL and JMPL) do not change the CWP, a procedure can be called without changing the window. See the section "Leaf-Procedure Optimization" in *Software Considerations*, contained in the separate volume *UltraSPARC Architecture Application Notes*

Since CWP arithmetic is performed modulo *N_REG_WINDOWS*, the highest-numbered implemented window overlaps with window 0. The *out*s of window *N_REG_WINDOWS* − 1 are the *in*s of window 0. Implemented windows are numbered contiguously from 0 through *N_REG_WINDOWS* −1.

Because the windows overlap, the number of windows available to software is 1 less than the number of implemented windows; that is, *N_REG_WINDOWS* – 1. When the register file is full, the *outs* of the newest window are the *ins* of the oldest window, which still contains valid data.

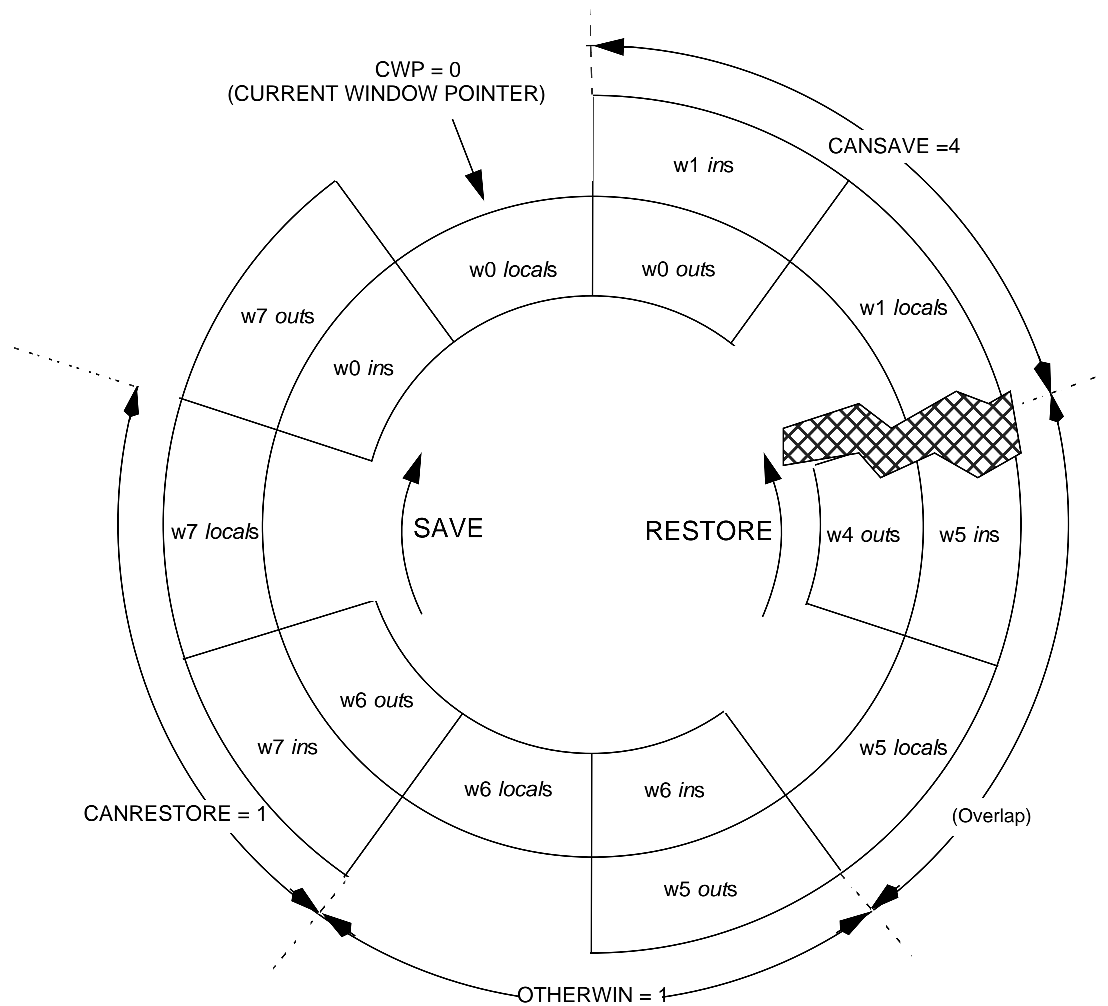
Window overflow is detected by the CANSAVE register, and window underflow is detected by the CANRESTORE register, both of which are controlled by privileged software. A window overflow (underflow) condition causes a window spill (fill) trap.

When a new register window is made visible through use of a SAVE instruction, the *local* and *out* registers are guaranteed to contain either zeroes or valid data from the current context. If software executes a RESTORE and later executes a SAVE, then the contents of the resulting window's *local* and *out* registers are not guaranteed to be preserved between the RESTORE and the SAVE[1]. Those registers may even have been written with "dirty" data, that is, data created by software running in a different context. However, if the clean_window protocol is being used, system software must guarantee that registers in the current window after a SAVE always contains only zeroes or valid data from that context. See *Clean Windows (cleanwinP) Register (PR 12)* on page 84, *Savable Windows (cansaveP) Register (PR 10)* on page 83, and *Restorable Windows (canrestoreP) Register (PR 11)* on page 84.

Implementation Note | An UltraSPARC Architecture virtual processor supports the guarantee in the preceding paragraph of "either zeroes or valid data from the current context"; it may do so either in hardware or in a combination of hardware and system software.

1. For example, any of those 16 registers might be altered due to the occurrence of a trap between the RESTORE and the SAVE, or might be altered during the RESTORE operation due to the way that register windows are implemented. After a RESTORE instruction executes, software must assume that the values of the affected 16 registers from before the RESTORE are unrecoverable.

Register Window Management Instructions on page 127 describes how the windowed integer registers are managed.



CANSAVE + CANRESTORE + OTHERWIN = *N_REG_WINDOWS* – 2

The current window (window 0) and the overlap window (window 5) account for the two windows in the right side of the equation. The "overlap window" is the window that must remain unused because its *ins* and *outs* overlap two other valid windows.

FIGURE 5-3 Windowed R Registers for *N_REG_WINDOWS* = 8

In FIGURE 5-3, *N_REG_WINDOWS* = 8. The eight *global* registers are not illustrated. CWP = 0, CANSAVE = 4, OTHERWIN = 1, and CANRESTORE = 1. If the procedure using window w0 executes a RESTORE, then window w7 becomes the current window. If the procedure using window w0 executes a SAVE, then window w1 becomes the current window.

5.2.3 Special R Registers

The use of two of the R registers is fixed, in whole or in part, by the architecture:

- The value of R[0] is always zero; writes to it have no program-visible effect.
- The CALL instruction writes its own address into register R[15] (*out* register 7).

Register-Pair Operands. LDTW, LDTWA, STTW, and STTWA instructions access a pair of words ("twin words") in adjacent R registers and require even-odd register alignment. The least significant bit of an R register number in these instructions is unused and must always be supplied as 0 by software.

When the R[0]–R[1] register pair is used as a destination in LDTW or LDTWA, only R[1] is modified. When the R[0]–R[1] register pair is used as a source in STTW or STTWA, 0 is read from R[0], so 0 is written to the 32-bit word at the lowest address, and the least significant 32 bits of R[1] are written to the 32-bit word at the highest address.

An attempt to execute anLDTW, LDTWA, STTW, or STTWA instruction that refers to a misaligned (odd) destination register number causes an *illegal_instruction* trap.

5.3 Floating-Point Registers (A1)

The floating-point register set consists of sixty-four 32-bit registers, which may be accessed as follows:

- Sixteen 128-bit quad-precision registers, referenced as $F_Q[0]$, $F_Q[4]$, …, $F_Q[60]$
- Thirty-two 64-bit double-precision registers, referenced as $F_D[0]$, $F_D[2]$, …, $F_D[62]$
- Thirty-two 32-bit single-precision registers, referenced as $F_S[0]$, $F_S[1]$, …, $F_S[31]$ (only the lower half of the floating-point register file can be accessed as single-precision registers)

The floating-point registers are arranged so that some of them overlap, that is, are aliased. The layout and numbering of the floating-point registers are shown in TABLE 5-2. Unlike the windowed R registers, all of the floating-point registers are accessible at any time. The floating-point registers can be read and written by

floating-point operate (FPop1/FPop2 format) instructions, by load/store single/double/quad floating-point instructions, by VIS™ instructions, and by block load and block store instructions.

TABLE 5-2 Floating-Point Registers, with Aliasing *(1 of 3)*

Single Precision (32-bit)		Double Precision (64-bit)			Quad Precision (128-bit)		
Register	Assembly Language	Bits	Register	Assembly Language	Bits	Register	Assembly Language
$F_S[0]$	%f0	63:32	$F_D[0]$	%d0	127:64	$F_Q[0]$	%q0
$F_S[1]$	%f1	31:0					
$F_S[2]$	%f2	63:32	$F_D[2]$	%d2	63:0		
$F_S[3]$	%f3	31:0					
$F_S[4]$	%f4	63:32	$F_D[4]$	%d4	127:64	$F_Q[4]$	%q4
$F_S[5]$	%f5	31:0					
$F_S[6]$	%f6	63:32	$F_D[6]$	%d6	63:0		
$F_S[7]$	%f7	31:0					
$F_S[8]]$	%f8	63:32	$F_D[8]$	%d8	127:64	$F_Q[8]$	%q8
$F_S[9]$	%f9	31:0					
$F_S[10]$	%f10	63:32	$F_D[10]$	%d10	63:0		
$F_S[11]$	%f11	31:0					
$F_S[12]$	%f12	63:32	$F_D[12]$	%d12	127:64	$F_Q[12]$	%q12
$F_S[13]$	%f13	31:0					
$F_S[14]$	%f14	63:32	$F_D[14]$	%d14	63:0		
$F_S[15]$	%f15	31:0					
$F_S[16]$	%f16	63:32	$F_D[16]$	%d16	127:64	$F_Q[16]$	%q16
$F_S[17]$	%f17	31:0					
$F_S[18]$	%f18	63:32	$F_D[18]$	%d18	63:0		
$F_S[19]$	%f19	31:0					
$F_S[20]$	%f20	63:32	$F_D[20]$	%d20	127:64	$F_Q[20]$	%q20
$F_S[21]$	%f21	31:0					
$F_S[22]$	%f22	63:32	$F_D[22]$	%d22	63:0		
$F_S[23]$	%f23	31:0					