Figure 3-16: Standard Stack Frame

Base	Offset	Contents	Frame
		unspecified . . . variable size	*High Address*
%fp+BIAS	>+176		
%fp+BIAS	+176	(if present) additional incoming argument slots	Previous
%fp+BIAS	+126	6 extended word argument slots	
%fp+BIAS	0	16 extended word save area (see below)	
%fp+BIAS	-1	unspecified . . . variable size	Current
%sp+BIAS	>+176		
%sp+BIAS	+176	(if needed) additional outgoing argument slots	
%sp+BIAS	+128	6 extended word argument slots	
	+120	save area for %i7	
	+112	save area for %i6	
	+104	save area for %i5	
	+96	save area for %i4	
	+88	save area for %i3	
	+80	save area for %i2	
	+72	save area fir %i1	
	+64	save area for %i0	
	+56	save area for %l7	
	+48	save area for %l6	
	+40	save area for %l5	
	+32	save area for %l4	
	+24	save area for %l3	
	+16	save area for %l2	
	+8	save area for %l1	
%sp+BIAS	0	save area for %l0	
%sp+BIAS	-1	volatile memory	
%sp	0	(do not use)	*Low Address*

BIAS = 2047

Several key points about the stack frame deserve mention.

- Every stack frame must be 16-byte aligned.
- Every stack frame must have a 16-extended-word save area for the *in* and *local* registers, in case of window overflow or underflow. This save area always must exist at %sp plus a BIAS of 2047 (0x7ff).
- Arguments that do not fit in the argument registers are passed on the stack.
- Other areas depend on the compiler and the code being compiled. The standard calling sequence does not restrict how a language system uses the “unspecified” areas of the standard stack frame.

NOTE

The stack pointer is offset from the stack frame by a BIAS of 2047 (ox7ff). This BIAS permits stack frame references in the range of %fp+BIAS-6143 to %fp+BIAS+2047 and %sp+BIAS to %sp+BIAS+2047 to be made with only immediate offset addressing. By making the BIAS an odd number, the least significant bit of the stack pointer will be set and the register overflow and underflow handlers can easily distinguish a 64-bit register window from a 32-bit register window.

Across function boundaries, the standard function prologue shifts the register window, making the calling function’s *out* registers the called function’s *in* registers. It also allocates stack space, including the required areas of figure 3-16 and any private space it needs. The lowest 16 extended-words in the stack must—at all times—be reserved as the register save area. The example below illustrates this and allocates 176 bytes for the stack frame.

Figure 3-17: Function Prologue

```
second:
          save %sp, -176, %sp
```

For demonstration, assume a function named `first` calls `second`. The register windows for the two functions appear below.