CallingSequences

(Note: This needs to be filled in.)

A calling sequence is a contract between two blocks of code, a call site (within a caller) and and entry point (within a callee).

The contract governs the transfer of control to the caller, and its eventual return to the caller (or a successor version of the caller). The contract is specific to some function type agreed on between the caller and callee. The type may require data values to be passed between the caller and callee, using certain registers or other variable locations.

Registers and Stack

Native

The CPU (x86, SPARC), word size (ILP32, LP64), and OS (Windows, Solaris, Linux) together determine how native (C-style) calls are made. On systems which support argument registers, leftward arguments are packed into registers until the registers run out, and then stack locations are used. On ILP32 systems, longs and doubles are passed as pairs of 32-bit arguments.

value	x86_32	x86_64	sparc (W=4 /8)
native sp	ESP	RSP	O6
return pc	ESP(0)	RSP(0)	O7
int result	EAX	RAX	O0
long result	<EDX: EAX>	RAX	<O0:O1> / O0
float result	FPR1	XMM0	F0
reg. int args	none	(see below)	O0..O5
reg. long args	none	same as ints	int pairs / ints
reg. float args	none	(see below)	none / F0..F15
stack arg #i	ESP(4+i*4)	RSP(8+i*8)	SP(92/176+i*W)
sp alignment	16 bytes	16 bytes	2*W bytes

On x86 LP64 systems, as many as the first 6 non-float and first 8 float arguments are allocated to registers.

reg. arg	int#0	int#1	int#2	int#3	int#4	int#5	float regs
Windows	RCX	RDX	R8	R9	none	none	XMM0..XMM3
Lin/Sol	RDI	RSI	RDX	RCX	R8	R9	XMM0..XMM7

Interpreted

The top entries of the interpreter stack are used to marshal arguments. The leftmost argument (which is the method receiver if there is one) is the first one pushed, and therefore is deepest in the stack. The callee gets an argument pointer to the most recently pushed argument (which is the rightmost).

The native registers are used for the return PC and return values. On x86, the argument pointer is cleverly overloaded on the native stack pointer. On other systems, the argument pointer is passed in a separate register. In any case, the outgoing native stack pointer is passed and recorded separately during any interpreter call.

value	x86_32	x86_64	sparc (W=4 /8)
interp. method	EBX	RBX	G5
interp. arg ptr	ESP+4	RSP+8	G4
interp. saved sp	ESI	RSI	O5

Certain registers may be reserved, by both the interpreter and compiler, to refer to current thread and the base of the heap (if compressed oops are enabled).

value	x86_32	x86_64	sparc (W=4 /8)
JavaThread	none	R15	G2
HeapBase	none	R12	none/G6

The following interpreter register assignments do not participate in calling conventions, but are given here for reference. Note that the Java stack pointer is the native stack pointer on x86 systems.

value	x86_32	x86_64	sparc (W=4 /8)
interp. java sp	ESP	RSP	L0
interp. fp	EBP	RBP	none
(what else?)			

Compiled

Compiled method calls are designed, like native calls, to get the job done in the fewest possible machine cycles.

TO DO: Define inline caches, etc.

value	x86_32	x86_64	sparc (W=4 /8)
java int args	ECX, EDX	j_rarg0..5	O0..O5
java long args	none	same asints	G1,G4 / O0..O5
java float args	XMM0, XMM1	j_farg0..7	F0..F7
inline cache	EAX	RAX	G5

For compiled code, the integer register assignments are different between Java and C. They are shifted to allow JNI wrappers to insert an extra leading argument without moving arguments around.

reg. arg	int#0	int#1	int#2	int#3	int#4	int#5
Windows, Java	RDX	R8	R9	RDI	RSI	RCX
Lin/Sol, Java	RSI	RDX	RCX	R8	R9	RDI
C argument	C #1	C #2	C #3	C #4	C #5	C #0

Call Site Setup

Interpreted

Compiled

Statically Linked Calls

Interpreted

Compiled

Virtual Method Calls

Interpreted

Compiled

Interface Method Calls

Interpreted

Compiled

Call Reception

Interpreted Methods

Specialized Interpreted Methods

Compiled Methods

Native Methods

Verified Entry Point

Return Processing

Return from Interpreted

Return from Compiled

Return from Native

Return to Interpreted

Return to Compiled

Exception Processing

Exception from Interpreted

Exception from Compiled

Exception from Native

Exception to Interpreted

Exception to Compiled

Edge Cases

Initial Call Site Linkage

Unexpected Receiver Type

Call to Deoptimized Method

Return to Deoptimized Method

Exception to Deoptimized Method

Adapter Stubs

Call Site Setup

Interpreted

Compiled

Statically Linked Calls

Interpreted

Compiled

Virtual Method Calls

Interpreted

Compiled

Interface Method Calls

Interpreted

Compiled

Call Reception

Interpreted Methods

Specialized Interpreted Methods

Compiled Methods

Native Methods

Verified Entry Point

Return Processing

Return from Interpreted

Return from Compiled

Return from Native

Return to Interpreted

Return to Compiled

Exception Processing

Exception from Interpreted

Exception from Compiled

Exception from Native

Exception to Interpreted

Exception to Compiled

Edge Cases

Initial Call Site Linkage

Unexpected Receiver Type

Call to Deoptimized Method

Return to Deoptimized Method

Exception to Deoptimized Method

Adapter Stubs