Parameter Passing

Visual Studio 2013

The first four integer arguments are passed in registers. Integer values are passed (in order left to right) in RCX, RDX, R8, and R9. Arguments five and higher are passed on the stack. All arguments are right-justified in registers. This is done so the callee can ignore the upper bits of the register if need be and can access only the portion of the register necessary.

Floating-point and double-precision arguments are passed in XMM0 – XMM3 (up to 4) with the integer slot (RCX, RDX, R8, and R9) that would normally be used for that cardinal slot being ignored (see example) and vice versa.

__m128 types, arrays and strings are never passed by immediate value but rather a pointer is passed to memory allocated by the caller. Structs/unions of size 8, 16, 32, or 64 bits and __m64 are passed as if they were integers of the same size. Structs/unions other than these sizes are passed as a pointer to memory allocated by the caller. For these aggregate types passed as a pointer (including __m128), the caller-allocated temporary memory will be 16-byte aligned.

Intrinsic functions that do not allocate stack space and do not call other functions can use other volatile registers to pass additional register arguments because there is a tight binding between the compiler and the intrinsic function implementation. This is a further opportunity for improving performance.

The callee has the responsibility of dumping the register parameters into their shadow space if needed.

The following table summarizes how parameters are passed:

Parameter type	**How passed**
Floating point	First 4 parameters – XMM0 through XMM3. Others passed on stack.
Integer	First 4 parameters – RCX, RDX, R8, R9. Others passed on stack.
Aggregates (8, 16, 32, or 64 bits) and __m64	First 4 parameters – RCX, RDX, R8, R9. Others passed on stack.
Aggregates (other)	By pointer. First 4 parameters passed as pointers in RCX, RDX, R8, and R9
__m128	By pointer. First 4 parameters passed as pointers in RCX, RDX, R8, and R9

Example of argument passing 1 – all integers

```
func1(int a, int b, int c, int d, int e);
// a in RCX, b in RDX, c in R8, d in R9, e pushed on stack
```

Example of argument passing 2 – all floats

```
func2(float a, double b, float c, double d, float e);
// a in XMM0, b in XMM1, c in XMM2, d in XMM3, e pushed on stack
```

Example of argument passing 3 – mixed ints and floats

```
func3(int a, double b, int c, float d);
// a in RCX, b in XMM1, c in R8, d in XMM3
```

Example of argument passing 4 –__m64, __m128, and aggregates

```
func4(__m64 a, _m128 b, struct c, float d);
// a in RCX, ptr to b in RDX, ptr to c in R8, d in XMM3
```

See Also

Reference

Calling Convention