Procedure Linkage: AMD64 ABI (1 of 4)

[Ref: System V Application Binary Interface AMD64 Architecture Processor Supplement Version 1.0, §3.2.1]

- Registers
  - GPRs `%rbp`, `%rbx`, and `%r12` through `%r15` "belong" to the calling function and the called function is required to preserve their values. A called function must preserve these registers' values for its caller. [Callee-saved registers]
  - The remaining registers "belong" to the called function. If a calling function wants to preserve such a register value across a function call, it must save the value in its local stack frame. [Caller-saved registers]

Procedure Linkage: AMD64 ABI (2 of 4)

Position	Contents	Frame
`8n+16(%rbp)`	memory argument eightbyte n	Previous
	...	
`16(%rbp)`	memory argument eightbyte 0	
`8(%rbp)`	return address	Current
`0(%rbp)`	previous `%rbp` value	
`-8(%rbp)`	unspecified	
	...	
`0(%rsp)`	variable size	
`-128(%rsp)`	red zone	

Procedure Linkage: AMD64 ABI (3 of 4)

[Ref: System V Application Binary Interface AMD64 Architecture Processor Supplement Version 1.0, §3.2.2]

- Stack Frame
  - Grows downwards from high addresses.
  - The end of the input argument area shall be aligned on a 16-byte boundary.
  - Once control has been passed to the function entry point, `%rsp` points to the return address, and the value of `(%rsp+8)` is a multiple of 16.
  - The 128-byte area beyond the location pointed to by `%rsp` is considered to be reserved and shall not be modified by signal or interrupt handlers. Therefore, functions may use this area for temporary data that is not needed across function calls. In particular, leaf functions may use this area for their entire stack frame, rather than adjusting the stack pointer in the prologue and epilogue. This area is known as the red zone.
  - The conventional use of `%rbp` as a frame pointer for the stack frame may be avoided by using `%rsp` (the stack pointer) to index into the stack frame. This technique saves two instructions in the prologue and epilogue and makes one additional GPR (`%rbp`) available.

Procedure Linkage: AMD64 ABI (4 of 4)

[Ref: System V Application Binary Interface AMD64 Architecture Processor Supplement Version 1.0, §3.2.3]

- Parameter Passing [Simplified]
  - Arguments are classified into multiple classes based on size.
    - INTEGER: Integral types that fit into one of the GPRs.
    - MEMORY: Types that will be passed and returned in memory via stack.
  - Once arguments are classified, the registers get assigned (in left-to-right order) for passing as follows:
    - If the class is MEMORY, pass the argument on the stack at an address respecting the argument's alignment (which might be more than its natural alignment).
    - If the class is INTEGER, the next available register of the sequence `%rdi`, `%rsi`, `%rdx`, `%rcx`, `%r8`, `%r9` is used. Once registers are assigned, the arguments passed in memory are pushed on the stack in reversed (right-to-left) order.
- Returning of Values [Simplified]
  - Classify the return type as above.
    - If the type has class MEMORY, then the caller provides space for the return value and passes the address of this storage in `%rdi` as if it were the first argument to the function. In effect, this address becomes a "hidden" first argument. On return `%rax` will contain the address that has been passed in by the caller in `%rdi`.
    - If the class is INTEGER, the next available register of the sequence `%rax`, `%rdx` is used.

Alternate Linkage: x64 Calling Convention

[Ref: https://docs.microsoft.com/en-us/cpp/build/x64-calling-convention?view=msvc-160&viewFallbackFrom=vs-2017]

- By default, the first four arguments to a function are passed in registers. Integer-valued arguments in the leftmost four positions are passed in left-to-right order in `RCX`, `RDX`, `R8`, and `R9`, respectively. The fifth and higher arguments get pushed on the stack in right-to-left order. All integer arguments in registers are right-justified.
- A scalar return value that can fit into 64 bits is returned through `RAX`.
- Registers `RAX`, `RCX`, `RDX`, `R8`, `R9`, `R10`, and `R11` are volatile. Consider volatile registers destroyed on function calls unless otherwise safety-provable otherwise by analysis.
- Registers `RBX`, `RBP`, `RDI`, `RSI`, `RSP`, `R12`, `R13`, `R14`, and `R15` are nonvolatile. They must be saved and restored by a function that uses them.