

RET—Return from Procedure

| Opcode | Instruction | 64-Bit Mode | Compat/ Leg Mode | Description |
|--------------|------------------|-------------|------------------|---|
| C3 | RET | Valid | Valid | Near return to calling procedure. |
| CB | RET | Valid | Valid | Far return to calling procedure. |
| C2 <i>iw</i> | RET <i>imm16</i> | Valid | Valid | Near return to calling procedure and pop <i>imm16</i> bytes from stack. |
| CA <i>iw</i> | RET <i>imm16</i> | Valid | Valid | Far return to calling procedure and pop <i>imm16</i> bytes from stack. |

Description

Transfers program control to a return address located on the top of the stack. The address is usually placed on the stack by a CALL instruction, and the return is made to the instruction that follows the CALL instruction.

The optional source operand specifies the number of stack bytes to be released after the return address is popped; the default is none. This operand can be used to release parameters from the stack that were passed to the called procedure and are no longer needed. It must be used when the CALL instruction used to switch to a new procedure uses a call gate with a non-zero word count to access the new procedure. Here, the source operand for the RET instruction must specify the same number of bytes as is specified in the word count field of the call gate.

The RET instruction can be used to execute three different types of returns:

- **Near return** — A return to a calling procedure within the current code segment (the segment currently pointed to by the CS register), sometimes referred to as an intrasegment return.
- **Far return** — A return to a calling procedure located in a different segment than the current code segment, sometimes referred to as an intersegment return.
- **Inter-privilege-level far return** — A far return to a different privilege level than that of the currently executing program or procedure.

The inter-privilege-level return type can only be executed in protected mode. See the section titled “Calling Procedures Using Call and RET” in Chapter 6 of the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1*, for detailed information on near, far, and inter-privilege-level returns.

When executing a near return, the processor pops the return instruction pointer (offset) from the top of the stack into the EIP register and begins program execution at the new instruction pointer. The CS register is unchanged.

When executing a far return, the processor pops the return instruction pointer from the top of the stack into the EIP register, then pops the segment selector from the top of the stack into the CS register. The processor then begins program execution in the new code segment at the new instruction pointer.

The mechanics of an inter-privilege-level far return are similar to an intersegment return, except that the processor examines the privilege levels and access rights of the code and stack segments being returned to determine if the control transfer is allowed to be made. The DS, ES, FS, and GS segment registers are cleared by the RET instruction during an inter-privilege-level return if they refer to segments that are not allowed to be accessed at the new privilege level. Since a stack switch also occurs on an inter-privilege level return, the ESP and SS registers are loaded from the stack.

If parameters are passed to the called procedure during an inter-privilege level call, the optional source operand must be used with the RET instruction to release the parameters on the return. Here, the parameters are released both from the called procedure's stack and the calling procedure's stack (that is, the stack being returned to).

In 64-bit mode, the default operation size of this instruction is the stack size, i.e. 64 bits.

Operation

(* Near return *)

IF instruction = Near return

THEN;

IF OperandSize = 32

THEN

IF top 4 bytes of stack not within stack limits

THEN #SS(0); FI;

EIP ← Pop();

ELSE

IF OperandSize = 64

THEN

IF top 8 bytes of stack not within stack limits

THEN #SS(0); FI;

RIP ← Pop();

ELSE (* OperandSize = 16 *)

IF top 2 bytes of stack not within stack limits

THEN #SS(0); FI;

tempEIP ← Pop();

tempEIP ← tempEIP AND 0000FFFFH;

IF tempEIP not within code segment limits

THEN #GP(0); FI;

EIP ← tempEIP;

FI;

FI;