<u>Call code (THE CALLER):</u>

a). Saving the volatile resources (EAX, ECX, EDX, EFLAGS)
b). Passing parameters
c). Saving the returning address and performing the call

<u>Entry code (THE CALLEE – called subroutine):</u>

a). Building the new stackframe   PUSH EBP,
                                                                      MOV EBP, ESP

b). Allocating space for local variables   SUB ESP, nr_bytes
c). Saving non-volatile resources exposed to be modified

<u>Exit code (THE CALLEE):</u>

a). Restoring non-volatile resources
b). Freeing the space allocated for local variables [ADD ESP, nr_bytes_locals] – mentioned here just as a reverse for the above b) from the entry code, but not really necessary because deallocating the stackframe (mov esp, ebp) includes this action anyway from a practically point of view.

c). Deallocating the stackframe   MOV ESP, EBP (if we know exactly the size of the stackframe , ADD ESP, sizeof(stackframe) solves similarly…)
     and restoring the base of the   POP EBP
     caller stackframe (old EBP)   (a, b c – the reverse of the entry code)

d). Returning from the subroutine (RET) and deallocating passed parameters (if we have a STDCALL function)   -   (reverse of b + c from the call code)

It is still to be done the reverse of a) from call code. It is the task of the CALLER to do it together with a possible parameters take out from the stack (if it is a CDECL function).