2. x86 and x64 Processors, Assembly Language

Registers

- x64(AMD64) extends on x86 registers
- General purpose registers:

rax	🟦	eax	=	🟨	ax	=	🟨	ah	al
rbx	🟦	ebx	=	🟨	bx	=	🟨	bh	bl
rcx	🟦	ecx	=	🟨	cx	=	🟨	ch	cl
rdx	🟦	edx	=	🟨	dx	=	🟨	dh	dl
rsi	🟦	esi	=	🟨	si	=	🟦		sil
rdi	🟦	edi	=	🟨	di	=	🟦		dil
rbp	🟦	ebp	=	🟨	bp	=	🟦		bpl
rsp	🟦	esp	=	🟨	sp	=	🟦		spl
r8	🟦	r8d	=	🟦	r8w	=	🟦		r8b
r9	🟦	r9d	=	🟦	r9w	=	🟦		r9b
r10	🟦	r10d	=	🟦	r10w	=	🟦		r10b
r11	🟦	r11d	=	🟦	r11w	=	🟦		r11b
r12	🟦	r12d	=	🟦	r12w	=	🟦		r12b
r13	🟦	r13d	=	🟦	r13w	=	🟦		r13b
r14	🟦	r14d	=	🟦	r14w	=	🟦		r14b
r15	🟦	r15d	=	🟦	r15w	=	🟦		r15b

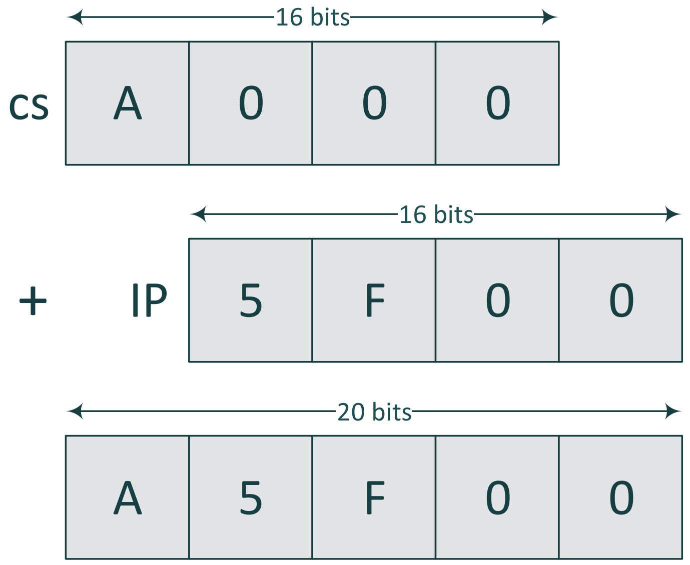
🟦 x64 registers 🟨 x86 registers

Modes of Operation

- Real Mode
  - Compatibility with older 16 bit processors
  - PCs still start in real mode (MBR, non UEFI)
- Protected Mode (x86)
  - Supports memory protection through segmentation and paging
- 64 bit mode (x64)
  - Segmentation not available, flat address space used
  - Processor treats segment base of CS, DS, ES, and SS are as zero

Real Mode

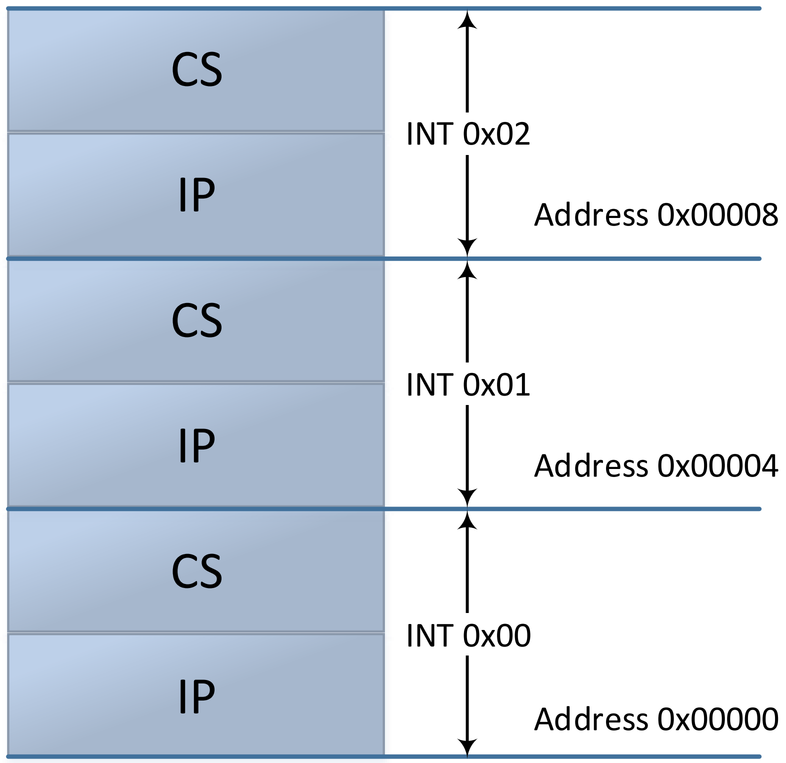
- 20 bits physical addresses => 1 MB available
- Logical addresses:
  - 32 bits (16 bit segment + 16 offset) converted to 20 bits
  - Ex: CS:IP



  - Different logical addresses can point to same physical address
    07C0:0000 = 0000:7C00 => 07C00
  - No memory protection

Real Mode IVT

- Interrupt Vector Table starting at 00000
- IVT contains 256 seg:offset pointers (1024 bytes)
- Some interrupts used as “low level” APIs, dos functions, bios functions

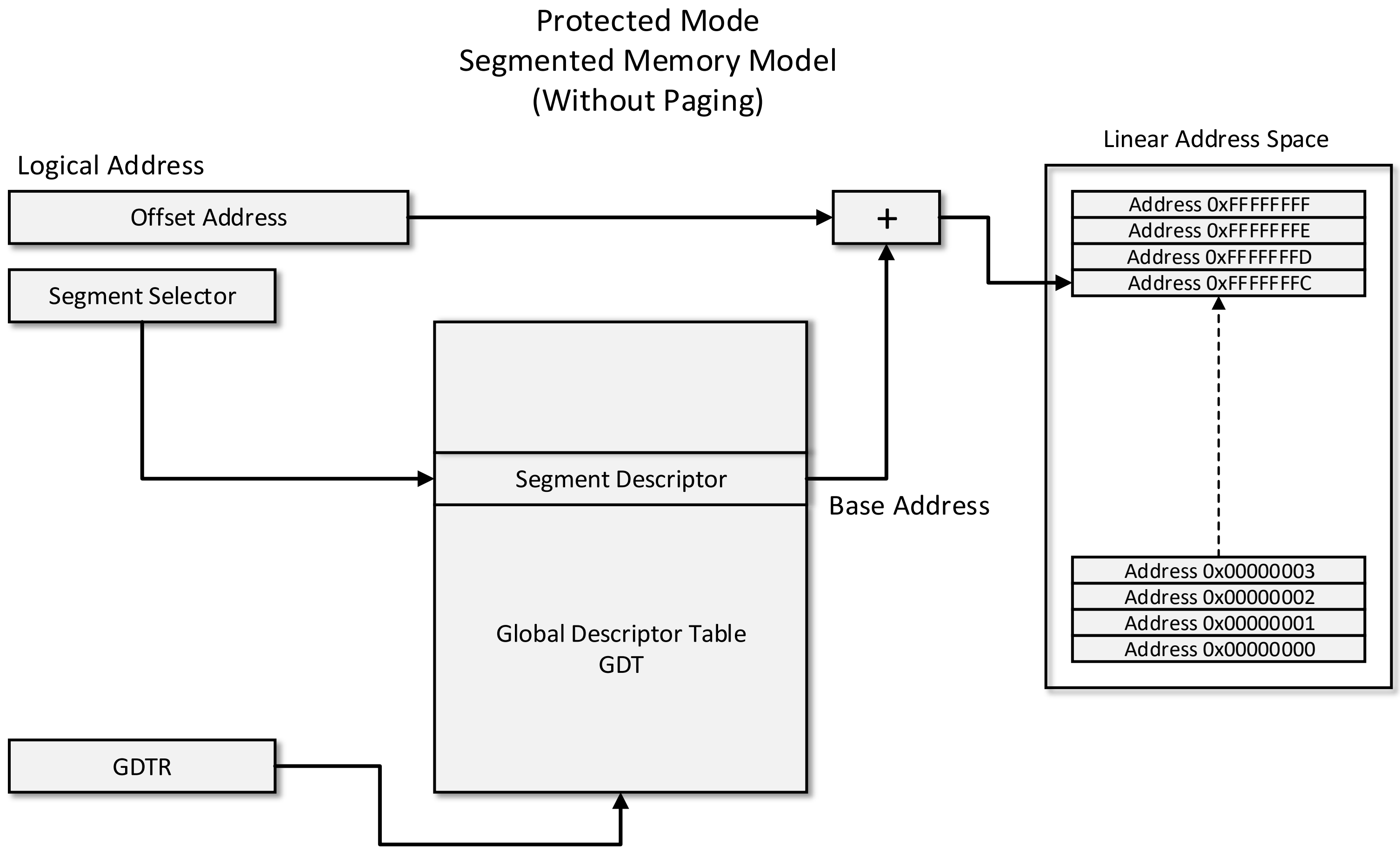


Protected Mode (x86)

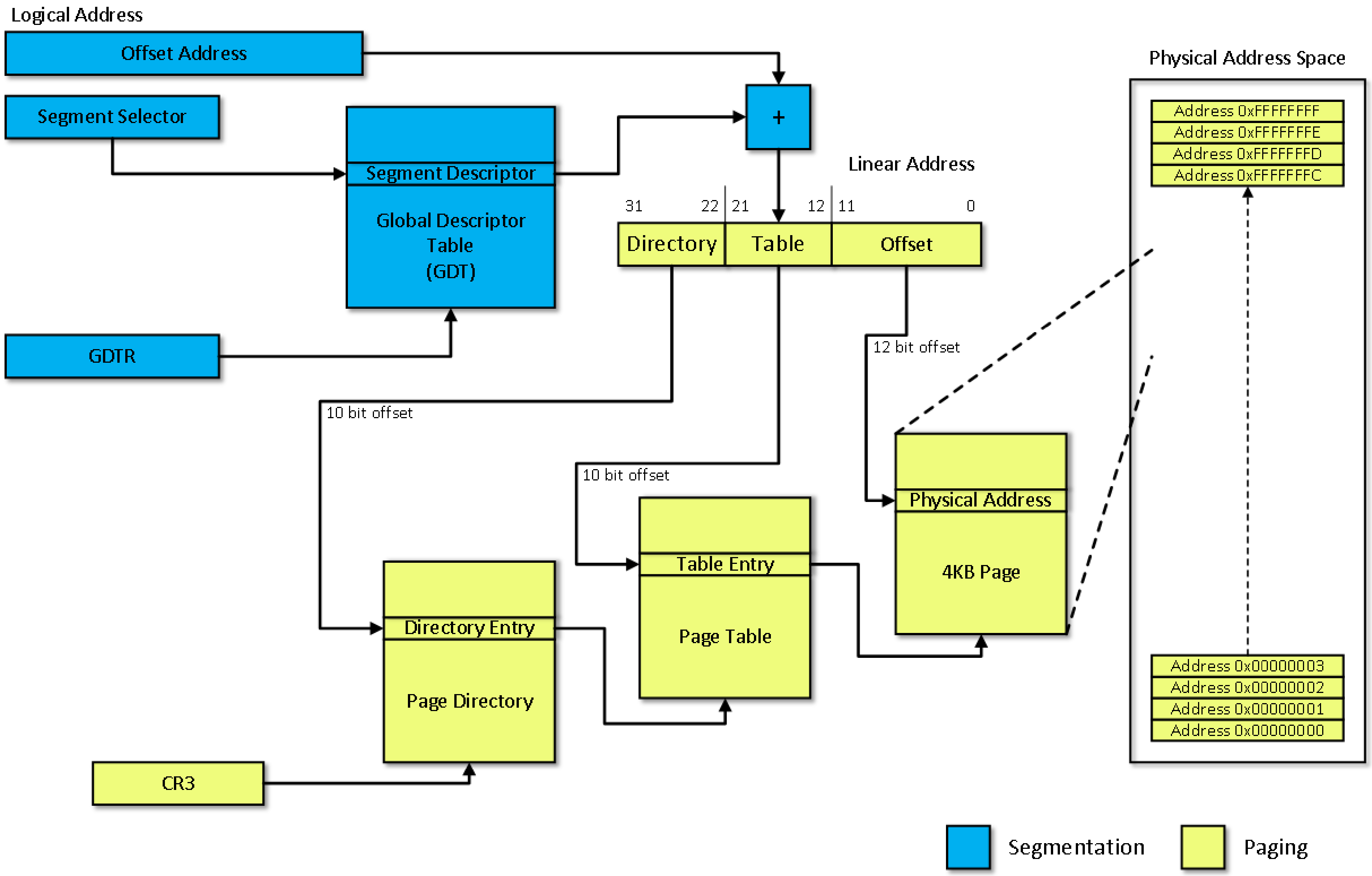
- 32 bits addresses => 4 GB
- Uses 4 privilege levels (rings)
- Segmentation is mandatory
- Paging is optional
  - If paging is not used, a linear address is a physical address.
  - If paging is enabled the address space is divided into pages of 4KB
  - A linear address is no longer a physical address
  - Pages can be mapped in physical memory or stored on disk
  - PFN (Page Frame Number) – 20 bit number specifying a page of 4KB in physical memory: 0x12345 (PFN) == 0x12345000 (physical mem)
  - Physical addresses stored in CR3, PDE, PTE are PFNs

Protected Mode Segmentation

- Segment registers are indices in GDT or LDT
- A descriptor in GDT defines a memory segment: base address, privilege level, code/data, etc.

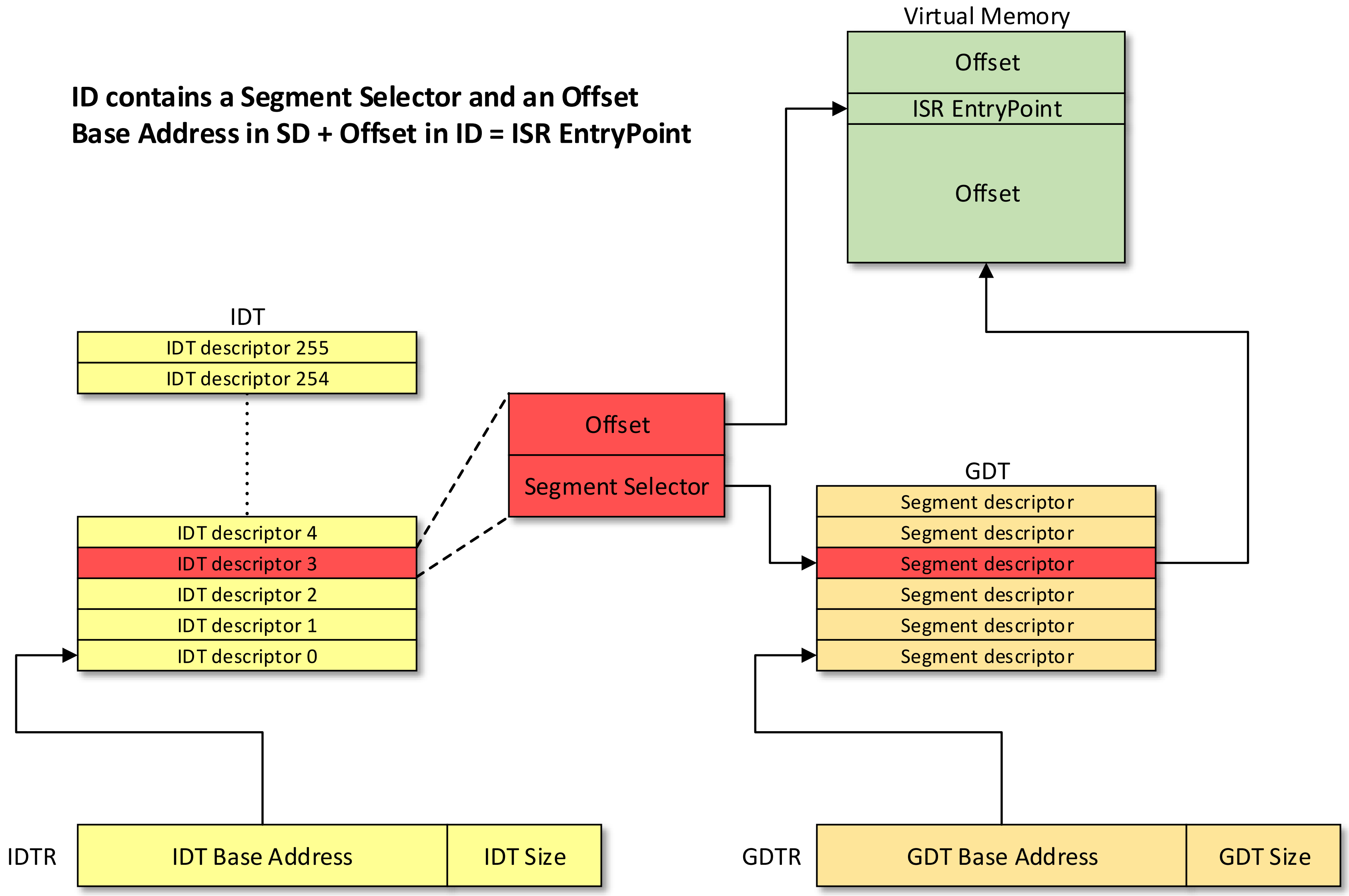


Protected Mode Paging (x86)



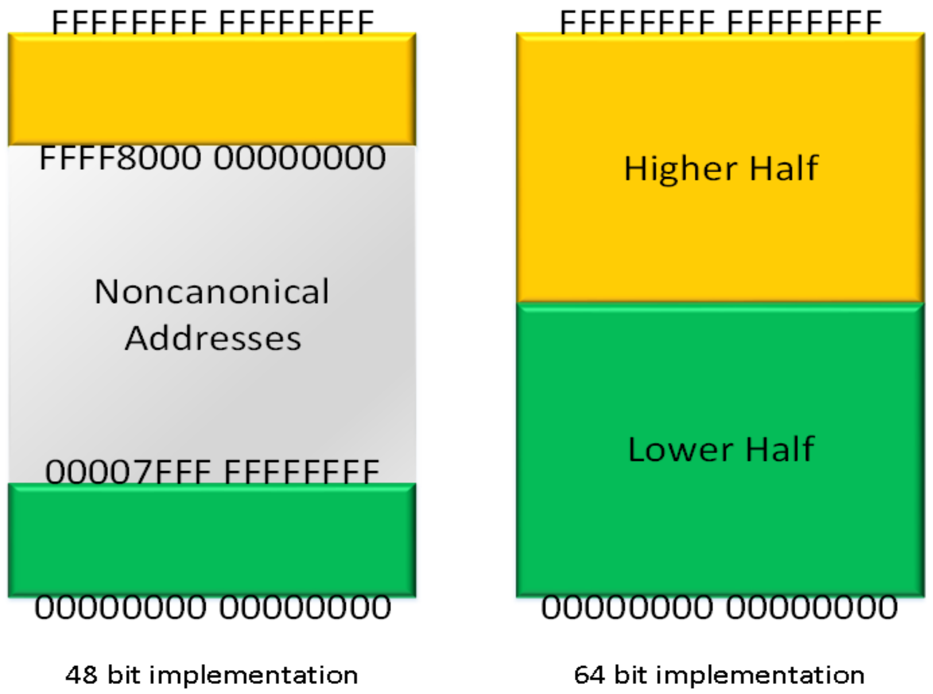
Protected Mode Interrupt Tables (x86)

ID contains a Segment Selector and an Offset
Base Address in SD + Offset in ID = ISR EntryPoint



64 Bit Mode

- Flat model, segments start at 0 and span all memory
- Paging similar to x86 but with more stages
- Pointers on 64 bits but only 48 bits used for physical addresses in current processors
  - 256 TB address space
- Canonical and non-canonical addresses



Assembly Language

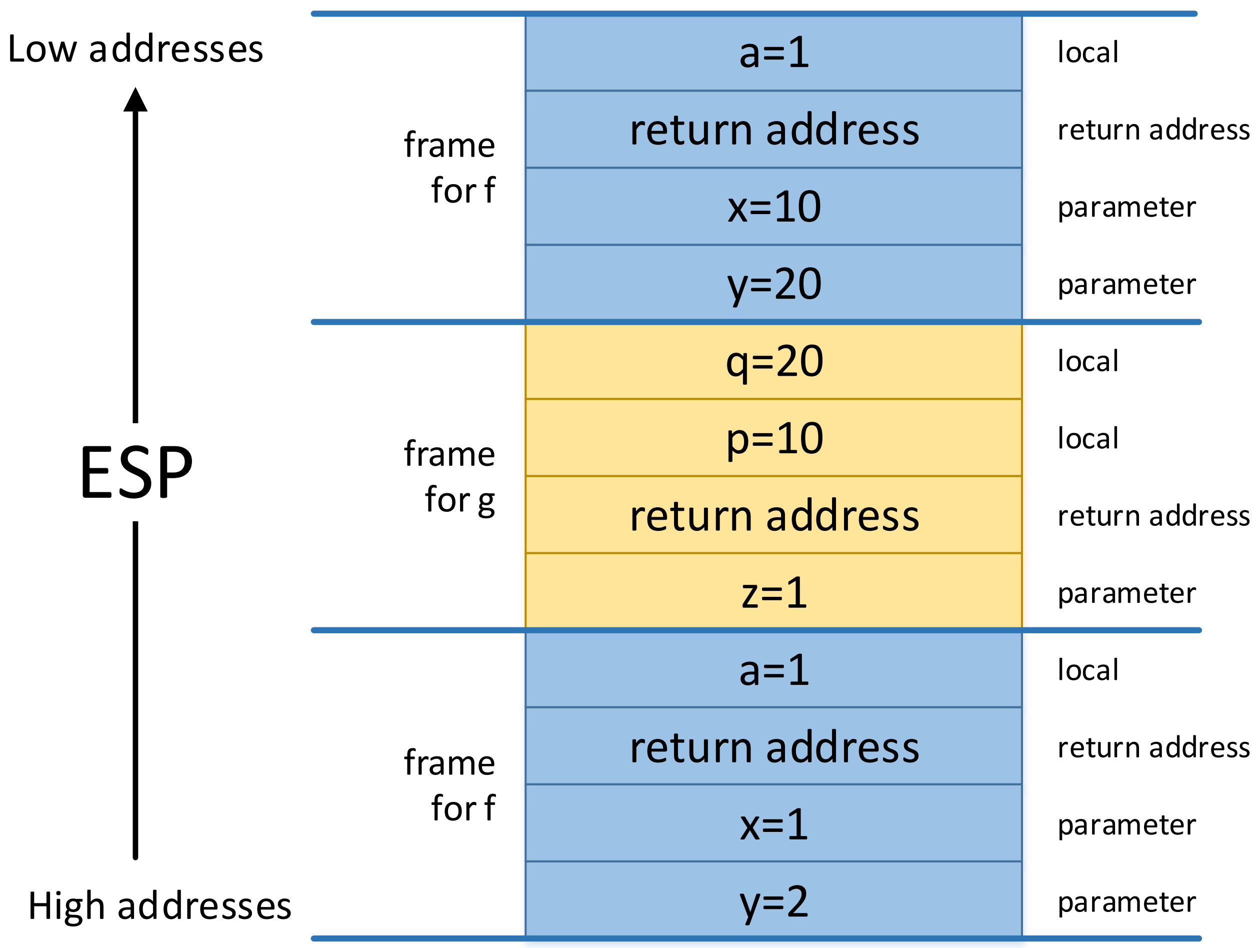
Transfer instructions
  - MOV, PUSH, POP, XCHG, LEA, MOVS
  - Do not change the flags
- Arithmetic and logical instructions
  - ADD, XOR, CMP, TEST, SHL
  - Change the flags
- Flow control instructions
  - Conditional jumps, based on the flags
  - JMP, CALL, RET, LOOP
- ESP contains pointer to the top of the stack
  - PUSH val ⇔ ESP=ESP-4 [ESP]=val
  - POP val ⇔ val=[ESP] ESP=ESP+4
  - CALL addr ⇔ PUSH retaddr EIP=addr
  - RET ⇔ POP EIP

Calling Conventions (x86)

- cdecl
  - Arguments passed on stack, right to left. The caller cleans the arguments from the stack, allowing variable argument lists (e.g. printf)
- stdcall
  - Arguments passed on stack, right to left. The callee cleans the stack
- Borland fastcall
  - First three arguments in EAX, EDX, ECX, remaining arguments pushed on stack. All arguments in left to right order. The callee cleans the stack
- Microsoft fastcall
  - First two arguments (left to right) in ECX, EDX. Remaining arguments pushed on stack right to left. The callee cleans the stack

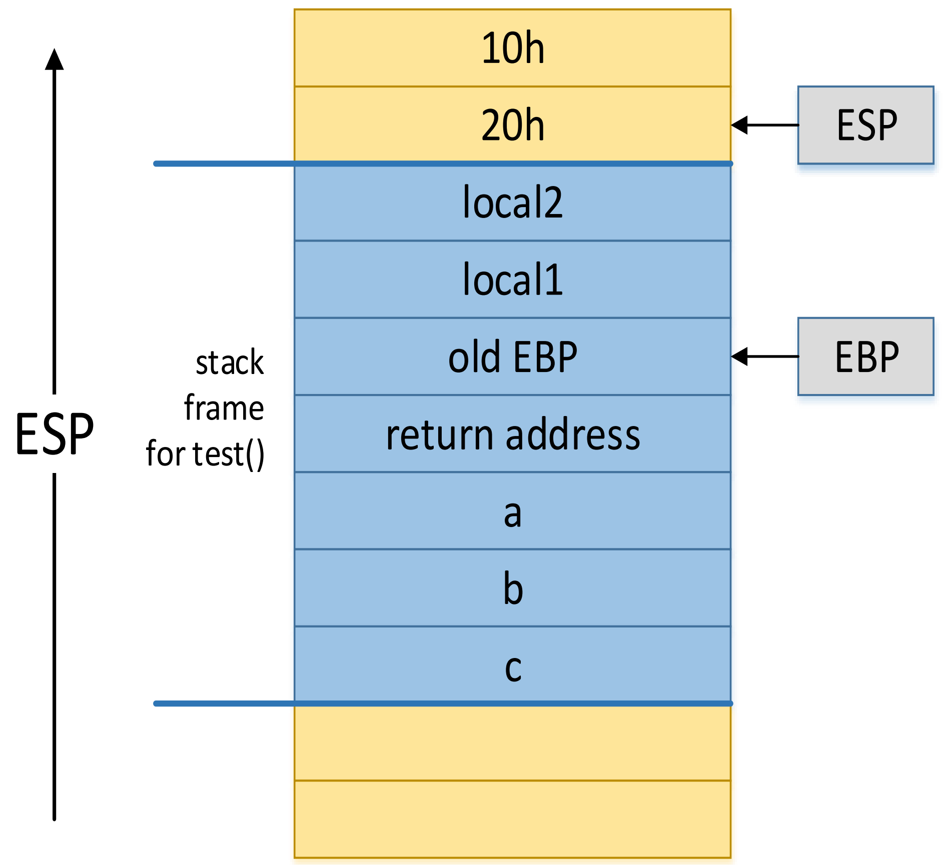
```
………………        ………………        ………………        ………………
push   5         push   5         push   4         push   5
push   4         push   4         push   5         push   4
push   3         push   3         mov    eax, 1    push   3
push   2         push   2         mov    edx, 2    mov    edx, 2
push   1         push   1         mov    ecx, 3    mov    ecx, 1
call   function  call   function  call   function  call   function
add    esp, 12   ………………        ………………        ………………
………………        ………………        ………………        ………………

cdecl            stdcall          Borland fastcall Microsoft fastcall
```

Function Stack Frames (x86)



```
function f(int x, int y)
{
        int a = 1;
        return g(a);
}

function g(int z)
{
        int p=10, q=20;
        return f(p, q);
}
..............................
..............................

f(1, 2);
```

Function Stack Frames (x86)

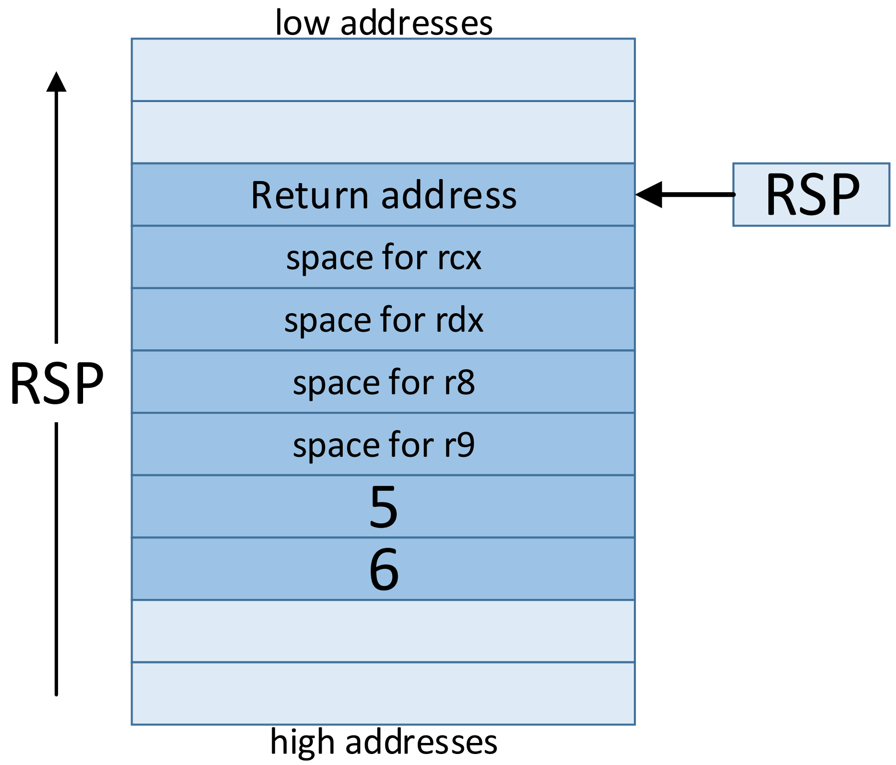


```
function test(int a, int b, int c)
{
        int local1, local2;
        ..................
        ..................
        return local1;
}

push    c
push    b
push    a
call    test
```

```
test proc
        push    ebp
        mov     ebp,esp
        sub     esp, 8
        ................
        mov     eax, [ebp+8]
        mov     ebx, [ebp+0Ch]
        mov     ecx, [ebp+10h]
        ................
        mov     eax, [ebp-4]
        mov     ebx, [ebp-8]
        ................
        push    20h
        push    10h
        call    addFunc
        ................
        mov     eax, [ebp-4]
        mov     esp, ebp
        pop     ebp
        ret     0Ch
test endp
```

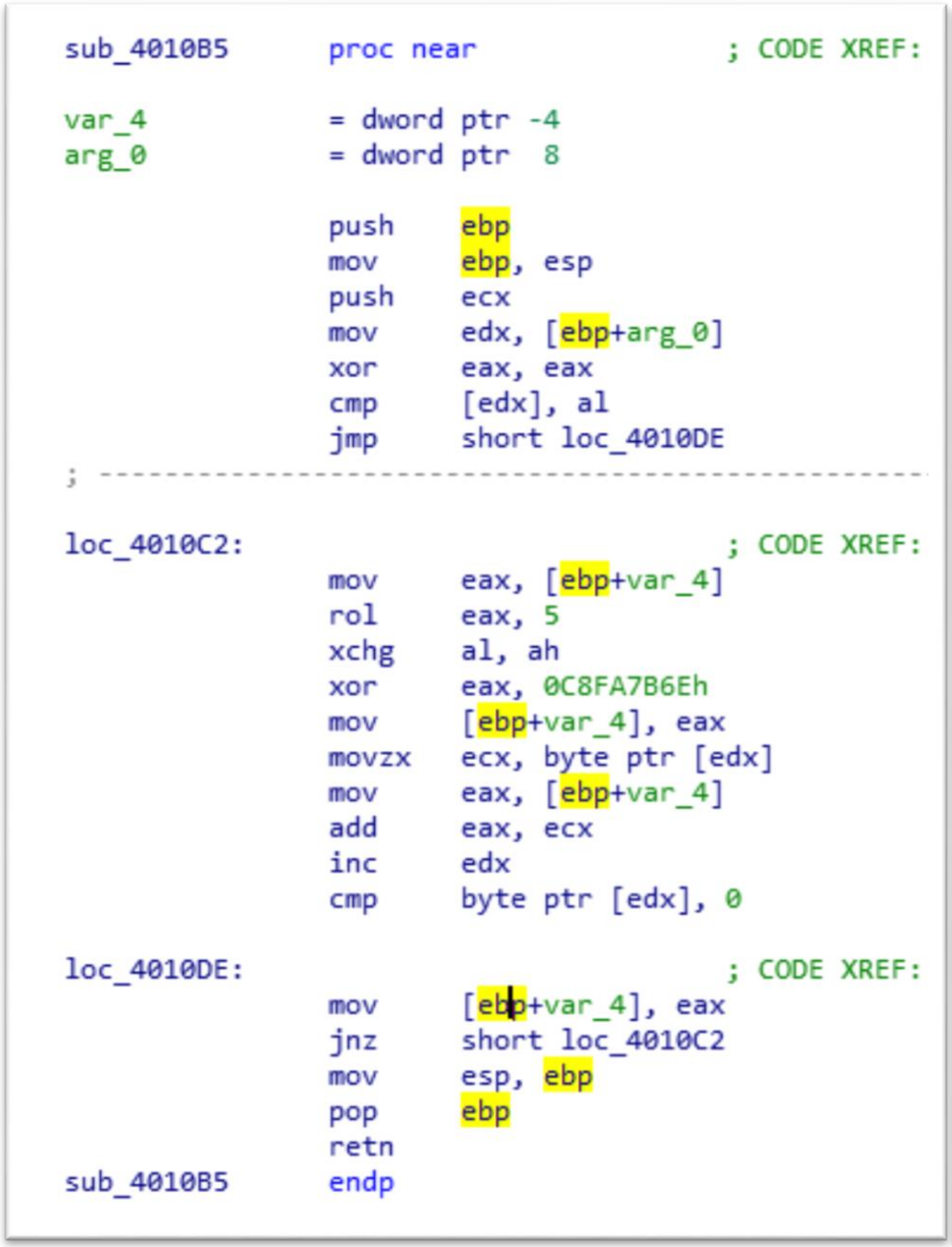
Calling Convention (x64)

- fastcall
  - The only mode used
  - First 4 integer parameters in RCX, RDX, R8, R9, others pushed on stack
  - First 4 float parameters in XMM0, XMM1, XMM2, XMM3, others on stack
  - RBP usually not used for frame any more
  - The caller must allocate stack space for first 4 parameters even if not used
  - The caller clears the stack

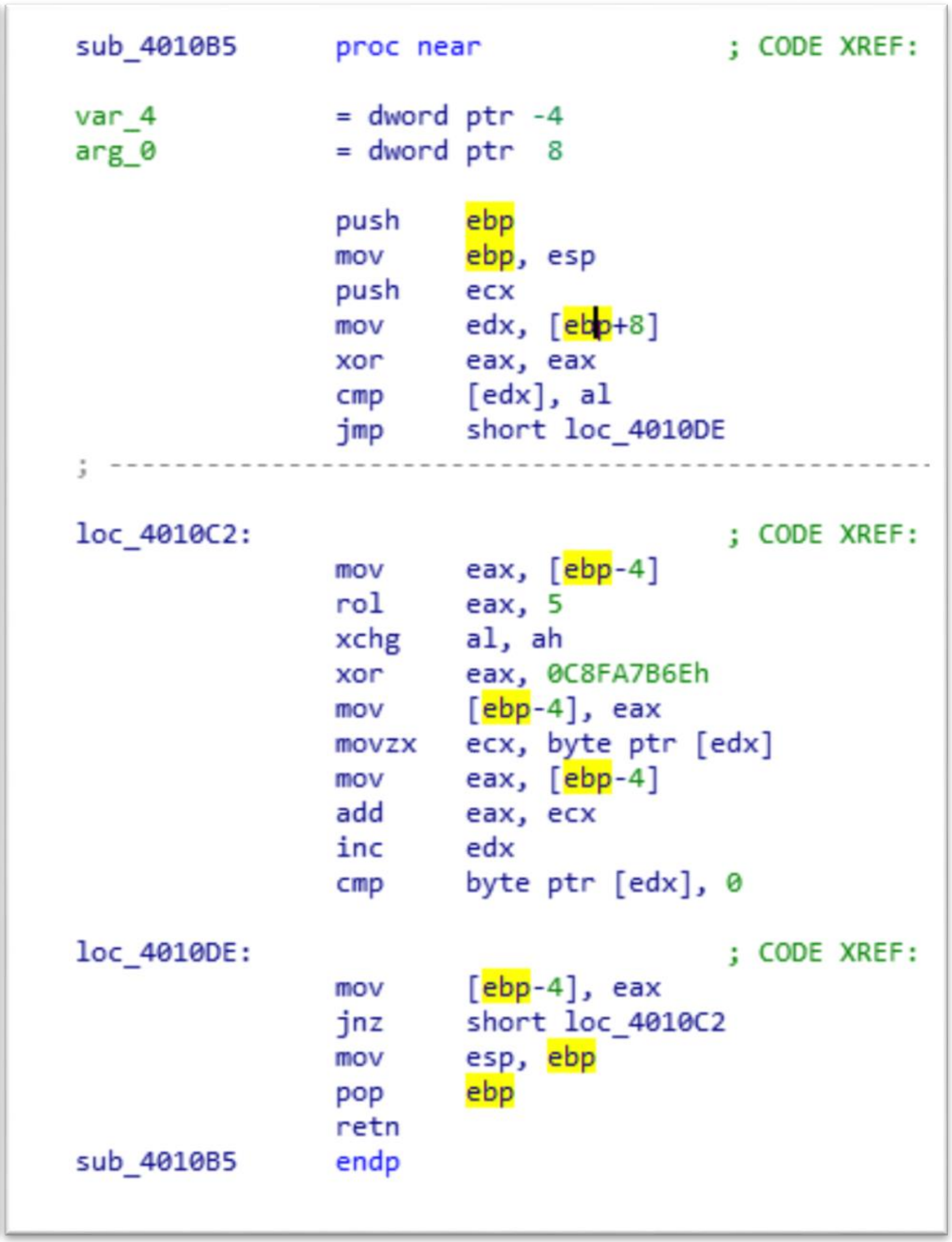


```
void function(int a1, int a2,
              int a3, int a4,
              int a5, int a6);

sub   rsp, 30h
mov   [rsp+28h], 6
mov   [rsp+20h], 5
mov   r9, 4
mov   r8, 3
mov   rdx, 2
mov   rcx, 1
call  function
add   rsp, 30h
```

IDA Example x86

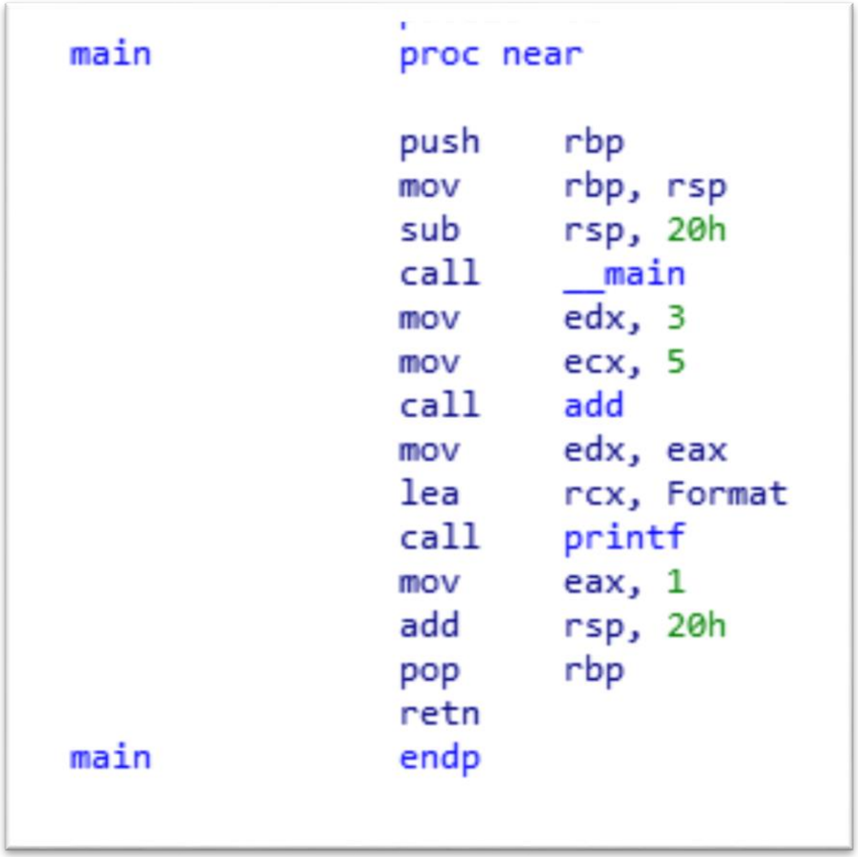


```
sub_4010B5      proc near               ; CODE XREF:

var_4           = dword ptr -4
arg_0           = dword ptr  8

                push    ebp
                mov     ebp, esp
                push    ecx
                mov     edx, [ebp+arg_0]
                xor     eax, eax
                cmp     [edx], al
                jmp     short loc_4010DE
; ---------------------------------------------------

loc_4010C2:                             ; CODE XREF:
                mov     eax, [ebp+var_4]
                rol     eax, 5
                xchg    al, ah
                xor     eax, 0C8FA7B6Eh
                mov     [ebp+var_4], eax
                movzx   ecx, byte ptr [edx]
                mov     eax, [ebp+var_4]
                add     eax, ecx
                inc     edx
                cmp     byte ptr [edx], 0

loc_4010DE:                             ; CODE XREF:
                mov     [ebp+var_4], eax
                jnz     short loc_4010C2
                mov     esp, ebp
                pop     ebp
                retn
sub_4010B5      endp
```

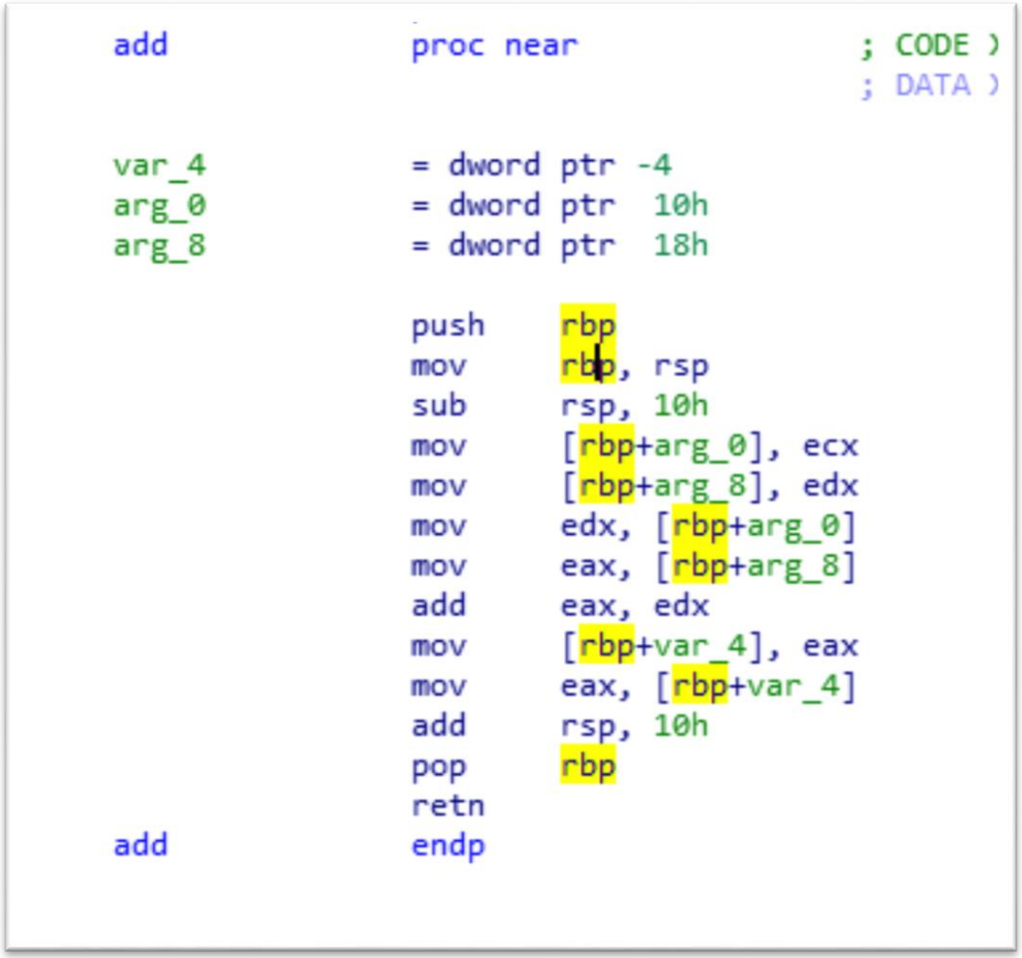


```
sub_4010B5      proc near               ; CODE XREF:

var_4           = dword ptr -4
arg_0           = dword ptr  8

                push    ebp
                mov     ebp, esp
                push    ecx
                mov     edx, [ebp+8]
                xor     eax, eax
                cmp     [edx], al
                jmp     short loc_4010DE
; ---------------------------------------------------

loc_4010C2:                             ; CODE XREF:
                mov     eax, [ebp-4]
                rol     eax, 5
                xchg    al, ah
                xor     eax, 0C8FA7B6Eh
                mov     [ebp-4], eax
                movzx   ecx, byte ptr [edx]
                mov     eax, [ebp-4]
                add     eax, ecx
                inc     edx
                cmp     byte ptr [edx], 0

loc_4010DE:                             ; CODE XREF:
                mov     [ebp-4], eax
                jnz     short loc_4010C2
                mov     esp, ebp
                pop     ebp
                retn
sub_4010B5      endp
```

IDA Example x64

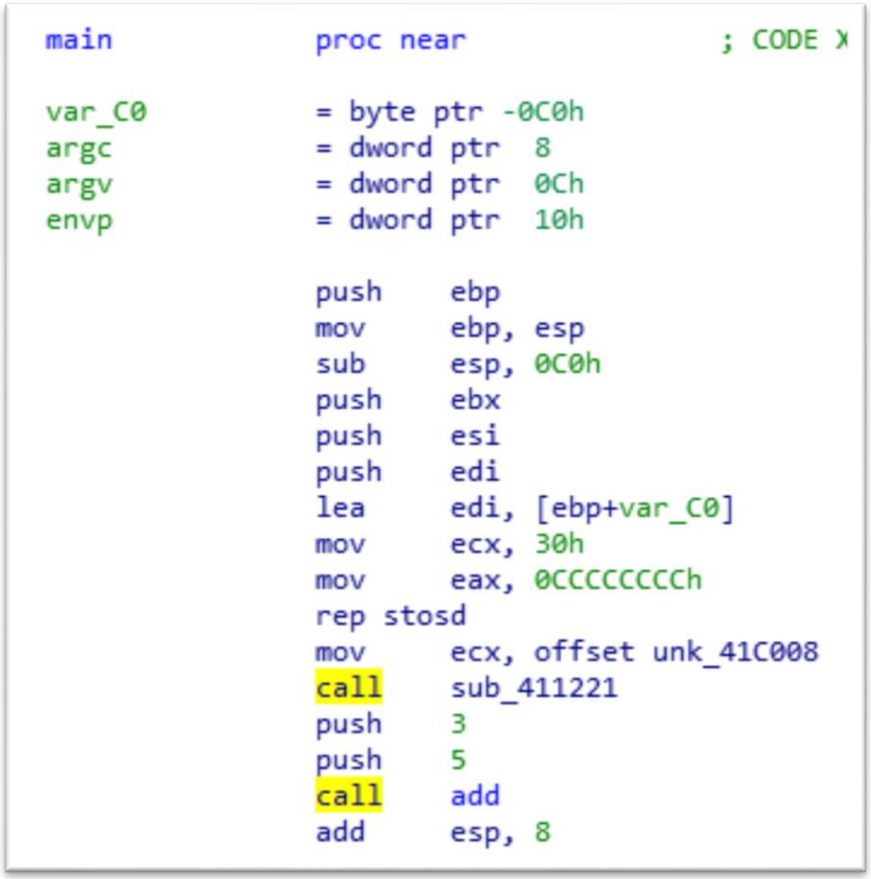


```
main                proc near

                    push    rbp
                    mov     rbp, rsp
                    sub     rsp, 20h
                    call    __main
                    mov     edx, 3
                    mov     ecx, 5
                    call    add
                    mov     edx, eax
                    lea     rcx, Format
                    call    printf
                    mov     eax, 1
                    add     rsp, 20h
                    pop     rbp
                    retn
main                endp
```

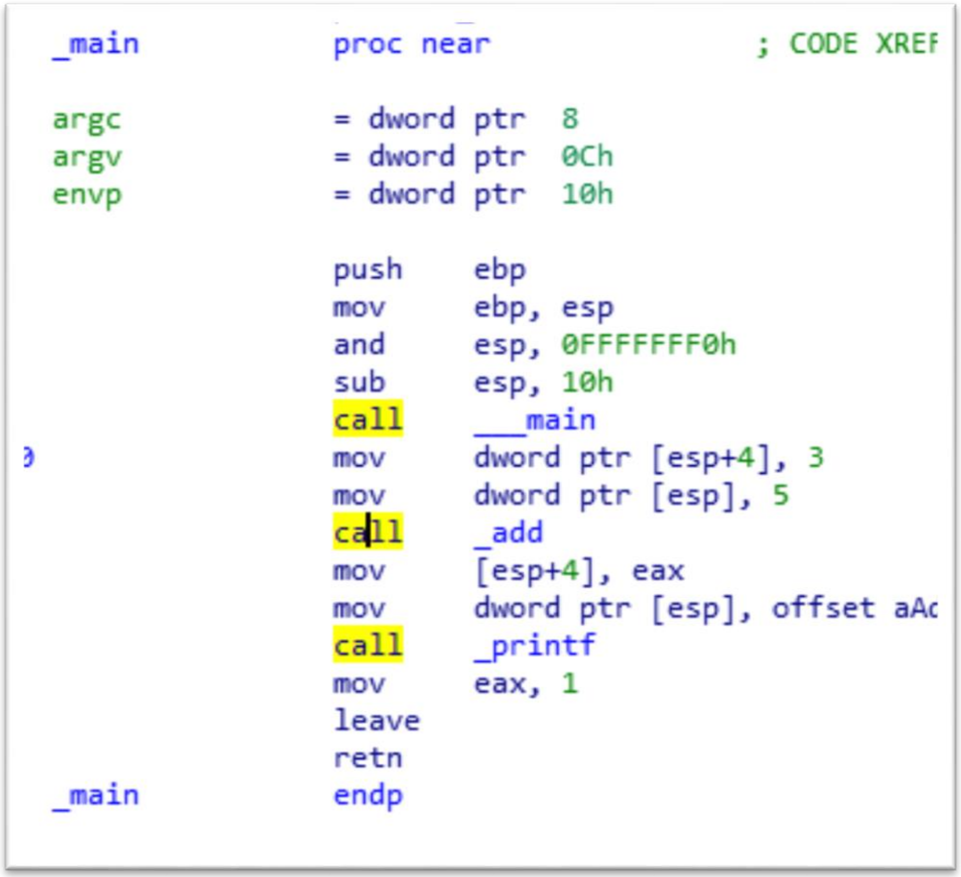


```
add                 proc near                        ; CODE
                                                     ; DATA

var_4               = dword ptr -4
arg_0               = dword ptr  10h
arg_8               = dword ptr  18h

                    push    rbp
                    mov     rbp, rsp
                    sub     rsp, 10h
                    mov     [rbp+arg_0], ecx
                    mov     [rbp+arg_8], edx
                    mov     edx, [rbp+arg_0]
                    mov     eax, [rbp+arg_8]
                    add     eax, edx
                    mov     [rbp+var_4], eax
                    mov     eax, [rbp+var_4]
                    add     rsp, 10h
                    pop     rbp
                    retn
add                 endp
```

Visual C vs GCC



```
main            proc near                ; CODE X

var_C0          = byte ptr -0C0h
argc            = dword ptr  8
argv            = dword ptr  0Ch
envp            = dword ptr  10h

                push    ebp
                mov     ebp, esp
                sub     esp, 0C0h
                push    ebx
                push    esi
                push    edi
                lea     edi, [ebp+var_C0]
                mov     ecx, 30h
                mov     eax, 0CCCCCCCCh
                rep stosd
                mov     ecx, offset unk_41C008
                call    sub_411221
                push    3
                push    5
                call    add
                add     esp, 8
```



```
_main           proc near                ; CODE XREF

argc            = dword ptr  8
argv            = dword ptr  0Ch
envp            = dword ptr  10h

                push    ebp
                mov     ebp, esp
                and     esp, 0FFFFFFF0h
                sub     esp, 10h
                call    ___main
                mov     dword ptr [esp+4], 3
                mov     dword ptr [esp], 5
                call    _add
                mov     [esp+4], eax
                mov     dword ptr [esp], offset aAc
                call    _printf
                mov     eax, 1
                leave
                retn
_main           endp
```