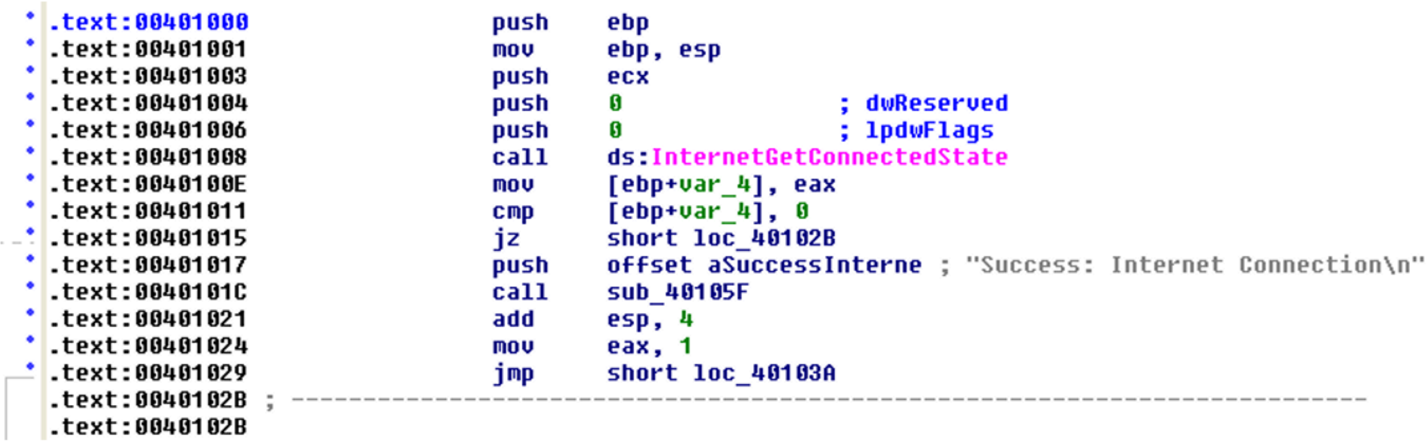
S10-L4

by Otman Hmich & Michael Andreoli

The following figure shows an excerpt of malware code. Identify the known constructs Assembly Language Exercise seen during the theoretical lesson.



```
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 push    ecx
.text:00401004                 push    0               ; dwReserved
.text:00401006                 push    0               ; lpdwFlags
.text:00401008                 call    ds:InternetGetConnectedState
.text:0040100E                 mov     [ebp+var_4], eax
.text:00401011                 cmp     [ebp+var_4], 0
.text:00401015                 jz      short loc_40102B
.text:00401017                 push    offset aSuccessInterne ; "Success: Internet Connection\n"
.text:0040101C                 call    sub_40105F
.text:00401021                 add     esp, 4
.text:00401024                 mov     eax, 1
.text:00401029                 jmp     short loc_40103A
.text:0040102B ; ---------------------------------------------------------------------------
.text:0040102B
```

Assembly language, is a low-level programming language closely related to a computer's architecture.

Low-Level: It is called "low-level" because it operates directly on the processor's instructions, using a language very close to machine code (which the CPU can execute directly).

Simple Instructions: Each instruction in assembly represents a very simple and specific operation, such as moving data between registers, performing calculations, or controlling the program's execution flow.

Registers and Memory: Operations work with registers (small memory units within the CPU) and memory addresses. For example, you can load a value into a register, perform a calculation with it, and then save the result in another register or in memory.

Hardware-Linked: Each type of processor (x86, ARM, etc.) has its own set of assembly instructions, which means that assembly code written for one type of processor will not work on another type without modifications.

Main Use: It is mainly used when precise control over hardware is needed, such as in operating systems, drivers, firmware, and some high-performance applications.

X86 REGISTERS:

EAX: accumulator register, used for arithmetic and logical operations.

EBX: base register, often used for data pointers.

ECX: counter register, used in loop operations.

EDX: data register, often used in input/output operations and multiplications.

STACK CREATION

.text:00401000 push ebp *CREATING THE STACK*

.text:00401001 mov ebp, esp *EBP BOTTOM OF STACK, ESP TOP OF STACK*

.text:00401003 push ecx *INSERTS ECX INTO THE STACK*

.text:00401004 push 0 ; dwReserved *INSERTS THE VALUE 0 INTO THE STACK*

.text:00401006 push 0 ; lpdwFlags *INSERTS THE VALUE 0 INTO THE STACK*

.text:00401008 call ds: InternetGetConnectedState CALL THE FUNCTION INTERNETGETCONNECTEDSTATE

.text:0040100E mov [ebp+var_4], eax *INSERTS THE VALUE OF EAX INSIDE VAR_4*

IF CYCLE

.text:00401011 cmp [ebp+var_4], 0 *WE COMPARE 0 WITH THE VALUE INSIDE VAR_4*

.text:00401015 jz short loc_40102B *JUMP CONDITIONED BY THE PREVIOUS COMPARISON, WE SKIP TO MEMORY LOCATION 40102B*

.text:00401017 push offset toSuccessInterne ; “Success: Internet Connection\n” *PUSH THE ESTABLISHED CONNECTION STRING INTO THE STACK*

.text:0040101C call sub_40105F *CALL THE FUNCTION SUB_40105F*

.text:00401021 add esp, 4 *WE ADD 4 TO THE ESP REGISTER VALUE*

.text:00401024 mov eax, 1 *WE REPLACE 1 TO THE VALUE OF THE EAX REGISTER*

.text:00401029 jump short loc_40103A *JUMP TO LOCATION 40103A*

.text:0040102B

```
.text:0040102B
```