Praticando com Assembly

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Desenvolvimento de Exploits
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

→ Montaremos um código em assembly usando o próprio bloco de notas do windows

→ o comentário no código é feito por meio do ponto e vírgula ;

-------------------------bloco de notas: desec.asm -----------------------

```
global _main

section .text
_main:
   NOP; NOP = No Operation → ele passa e n faz nada
   NOP
   NOP
   NOP
   NOP
   NOP
   NOP
   NOP
   NOP
   MOV EAX, 41424344h ; ABCD em hex [aqui começam as intruções]
   MOV BX, 4141h;  estamos movendo os valores para dentro de BX - 16 bits
   MOV CH, 41h; CH é a parte alta do registrador 1 byte
   MOV DL, 41h; DL é a parte baixa do registrador
   XOR EAX, EAX;
   XOR EBX, EBX
   XOR ECX, ECX
   XOR EDX, EDX
```

→ Salvamos o arquivo como desec.asm e agora, pelo cmd, vamos gerar o arquivo .obj por meio do nasm (assembler)

```
nasm -f win32 desec.asm
```

```
C:\Users\catulo\Desktop>nasm -f win32 desec.asm

C:\Users\catulo\Desktop>dir
 O volume na unidade C não tem nome.
 O Número de Série do Volume é 0A70-774F

 Pasta de C:\Users\catulo\Desktop

04/04/2024  23:19    <DIR>          .
04/04/2024  23:19    <DIR>          ..
02/04/2024  17:43                63 catulinho.cpp
02/04/2024  17:43           104.045 catulinho.exe
04/04/2024  23:15               279 desec.asm
04/04/2024  23:16               287 desec.asm.txt
04/04/2024  23:19               220 desec.obj
02/04/2024  17:29             1.065 Dev-C++.lnk
               6 arquivo(s)        105.959 bytes
               2 pasta(s)   95.068.139.520 bytes disponíveis
```

→ Aqui podemos ver que foi gerado o arquivo de extensão obj
→ Para decompilar, usaremos o objdump

```
objdump -d -M intel desec.obj
```

```
C:\Users\catulo\Desktop>objdump -d -M intel desec.obj

desec.obj:     file format pe-i386


Disassembly of section .text:

00000000 <_main>:
   0:   90                      nop
   1:   90                      nop
   2:   90                      nop
   3:   90                      nop
   4:   90                      nop
   5:   90                      nop
   6:   90                      nop
   7:   90                      nop
   8:   90                      nop
   9:   b8 44 43 42 41          mov    eax,0x41424344
   e:   66 bb 41 41             mov    bx,0x4141
  12:   b5 41                   mov    ch,0x41
  14:   b2 41                   mov    dl,0x41
  16:   31 c0                   xor    eax,eax
  18:   31 db                   xor    ebx,ebx
  1a:   31 c9                   xor    ecx,ecx
  1c:   31 d2                   xor    edx,edx
```

→ Caso não quiséssemos ver na sintaxe intel, e sim na AT&T, bastaria não passar o -M:

```
objdump -d desec.obj
```

```
C:\Users\catulo\Desktop>objdump -d desec.obj

desec.obj:     file format pe-i386


Disassembly of section .text:

00000000 <_main>:
   0:   90                      nop
   1:   90                      nop
   2:   90                      nop
   3:   90                      nop
   4:   90                      nop
   5:   90                      nop
   6:   90                      nop
   7:   90                      nop
   8:   90                      nop
   9:   b8 44 43 42 41          mov    $0x41424344,%eax
   e:   66 bb 41 41             mov    $0x4141,%bx
  12:   b5 41                   mov    $0x41,%ch
  14:   b2 41                   mov    $0x41,%dl
  16:   31 c0                   xor    %eax,%eax
  18:   31 db                   xor    %ebx,%ebx
  1a:   31 c9                   xor    %ecx,%ecx
  1c:   31 d2                   xor    %edx,%edx
```

→ Agora vamos linkar:

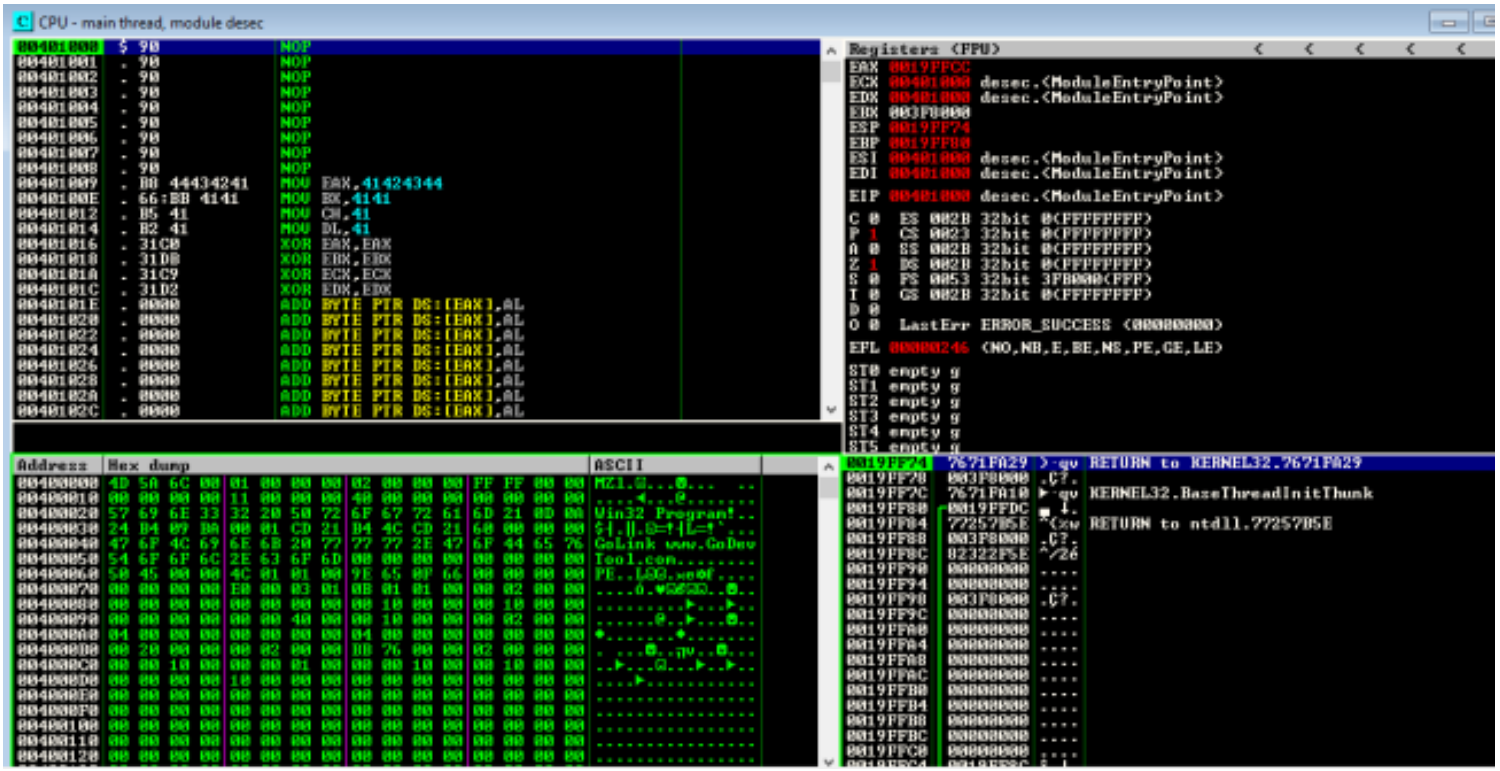
```
golink /entry _main desec.obj
```

→ o /entry serve para indicar o "entry point" (ponto de entrada do programa), no caso é a main

→ Esse processo gera o executável desec.exe

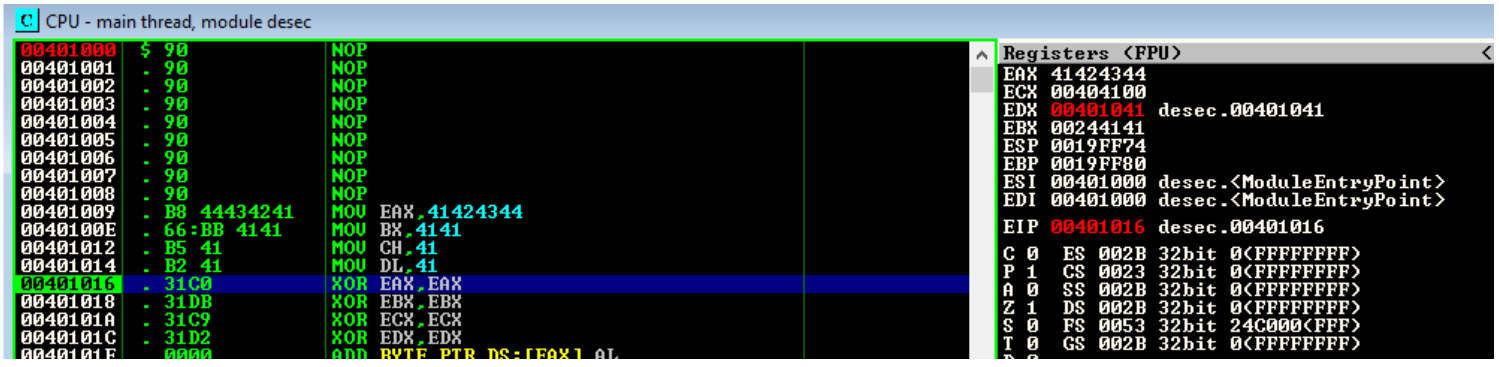
→ Posicionamos o desec.exe dentro do Immunity Debugger para analisar o código assembly em execução

→ Para melhorar a visualização do debugger, clicamos com o botão direito e selecionamos:
Appearence → Colors (all) → Dave's Black
Appearence → Font (all) → OEM fixed point

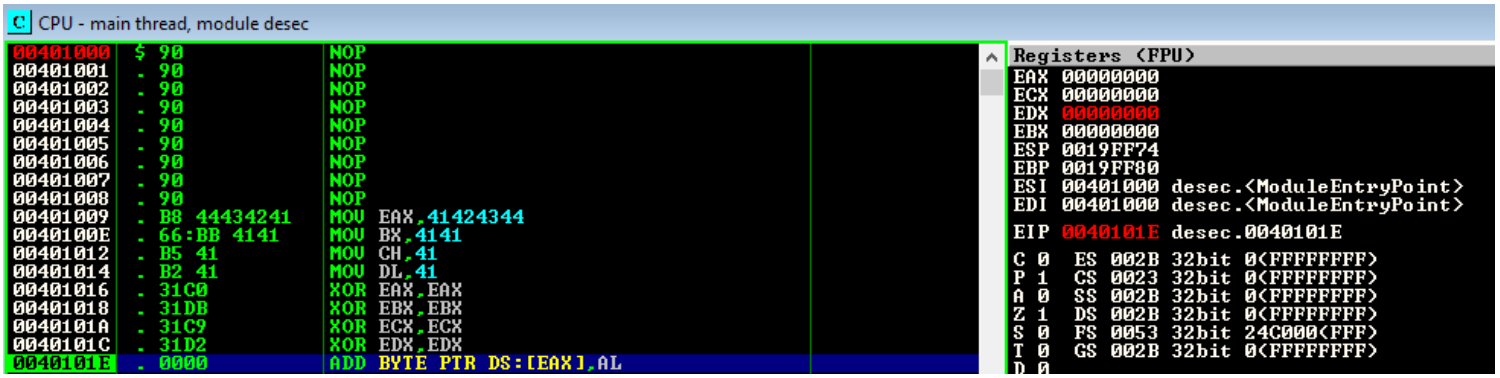


→ No canto superior esquerdo vemos o 401000 como endereço de memória da primeira instrução e o opcode sendo o 90(instrução) que é um nop.
→ No lado direito temos o EIP apontando para a próxima instrução a ser executada, no caso a 401000

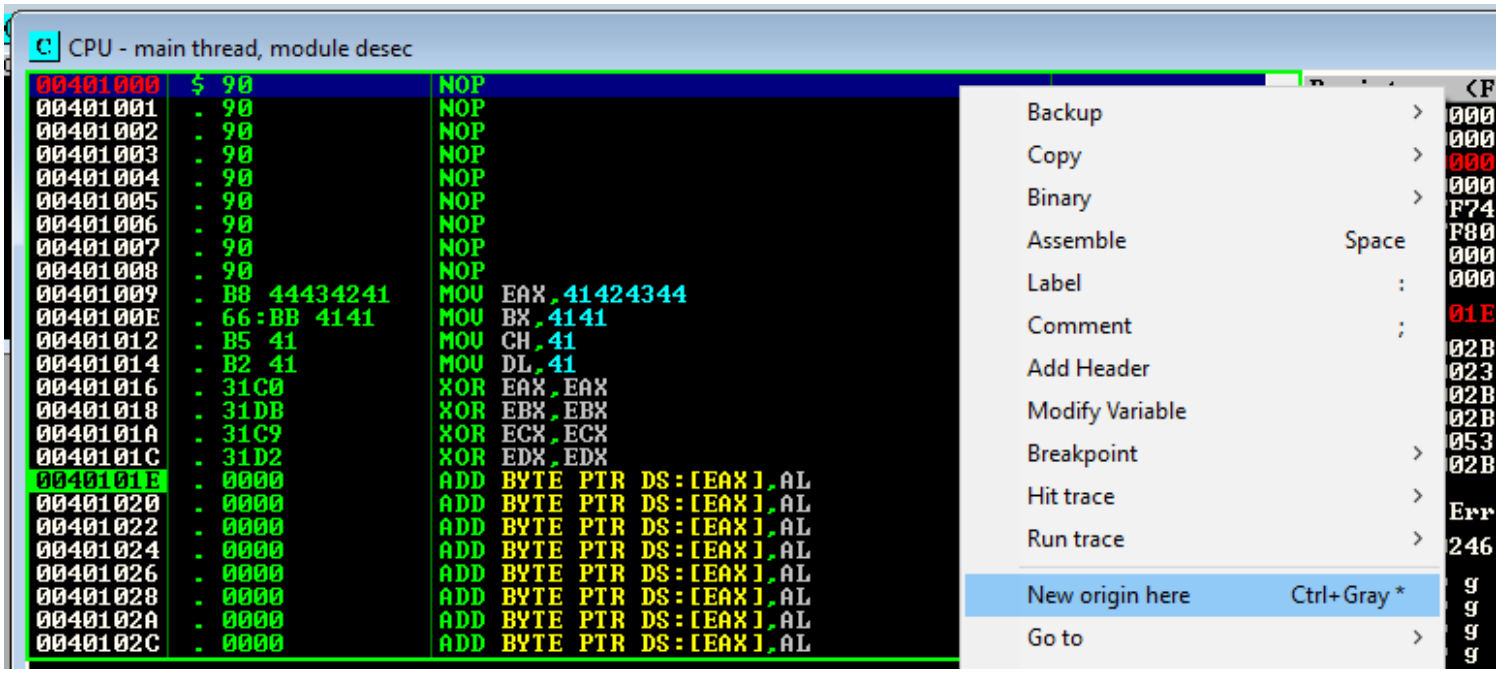
→ Para ir percorrendo o código linha a linha, vamos apertar a seta para baixo 
→ Poderemos observar que a execução de cada parte do código armazena nos registradores os parâmetros passados na linha de código
→ O XOR zera os registradores



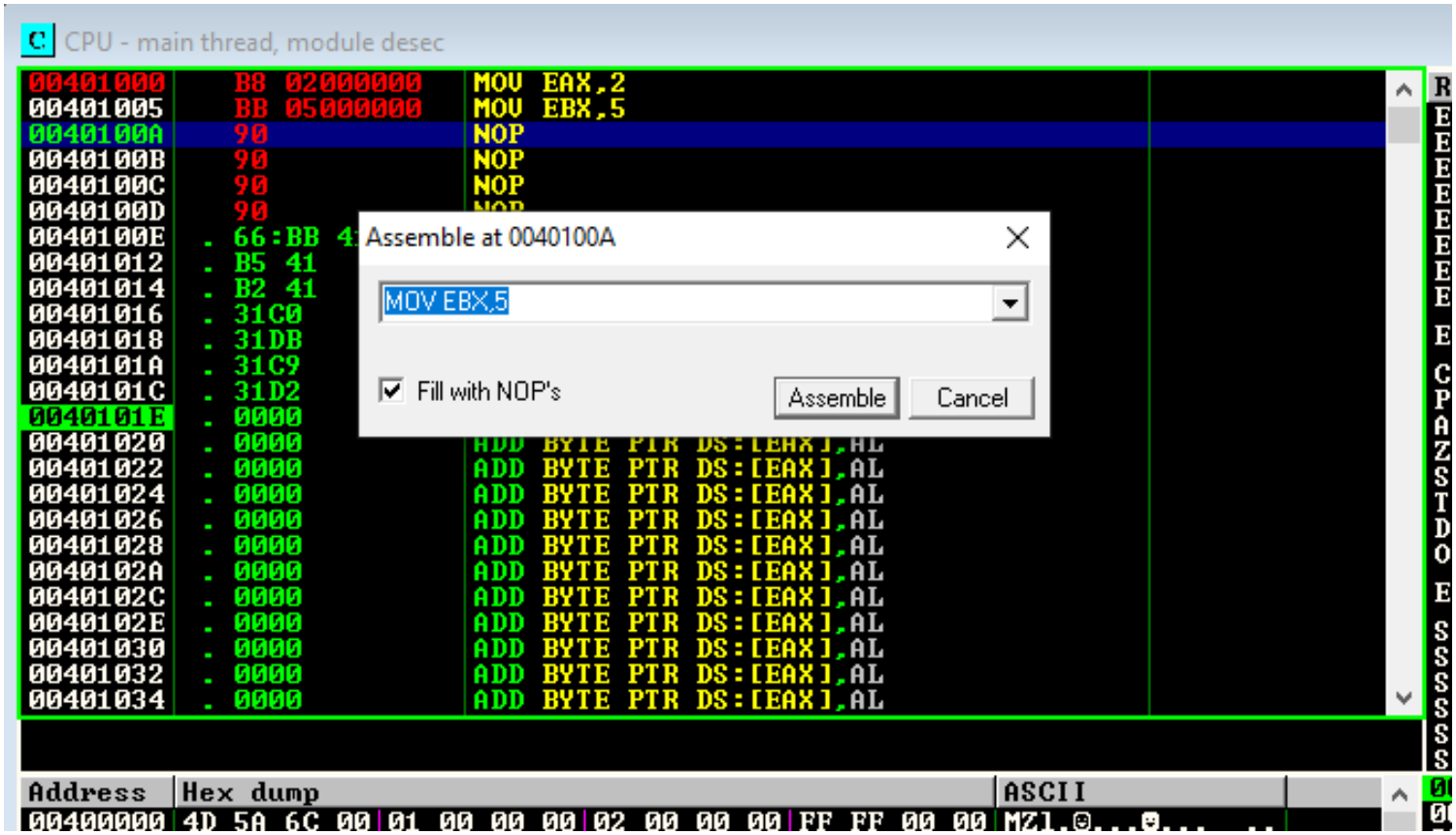
→ Após a execução dos 4 primeiros, temos os valores armazenados nos registradores EAX, ECX, EDX e EBX
→ O XOR irá zerá-los



→ Com o botão direito do mouse, podemos selecionar a opção de estabelecer novo início:



→ Para editar uma das linhas de código, basta clicar duas vezes em cima dela e editar na caixa de texto



INC EAX,EBX → soma EAX e EBX
CMP EAX,ECX → compara EAX e ECX

JMP - SALTAR
JNE - SALTAR SE NÃO IGUAL
JE - SALTAR SE IGUAL