



Goliath .NET CPU Emulator

* * * * *

Implementazione di Adler32
da IDA a GOLIATH

```
.686
.model flat,stdcall

option casemap:none

.const

ADLER32_BASE equ 65521
ADLER32_NMAX equ 5552

.code

align dword

Adler32 proc uses edi esi ebx lpBuffer:DWORD, dwBufLen:DWORD, dwAdler:DWORD

mov eax,dwAdler
mov ecx,dwBufLen
mov ebx,dwAdler
and eax,0FFFFh
shr ebx,16
mov esi,lpBuffer
jmp @F

.repeat
mov edi,ADLER32_NMAX
.if ecx<edi
mov edi,ecx
.endif
    sub ecx,edi
.repeat
movzx edx,byte ptr [esi]
add eax,edx
inc esi
add ebx,eax
dec edi
.until ZERO?
    mov edi,ADLER32_BASE
    xor edx,edx
    div edi
    push edx
    mov eax,ebx
    sub edx,edx
    div edi
    mov ebx,edx
    pop eax
@@:

    test ecx,ecx
.until ZERO?
shl ebx,16
add eax,ebx
ret
Adler32 endp

end
```

F . A . Q . :

*Ma come posso sfruttare
al meglio "Goliath .NET
CPU Emulator" se non
conosco bene l'assembly
x86? :-o*

...Prelevando il Codice
da un qualsiasi
Disassembler e/o Debugger! :-)

ad esempio:

IDA Pro Disassembler
v4.9 Freeware

<http://www.hex-rays.com/idapro/>

File Edit Jump Search View Options Windows Help

Text

En 0101 0101 0101 "s" * N X off # 'x' S M K /- ~ :

IDA View-A Hex View-A Exports Imports Names Functions Strings Structures En Enums

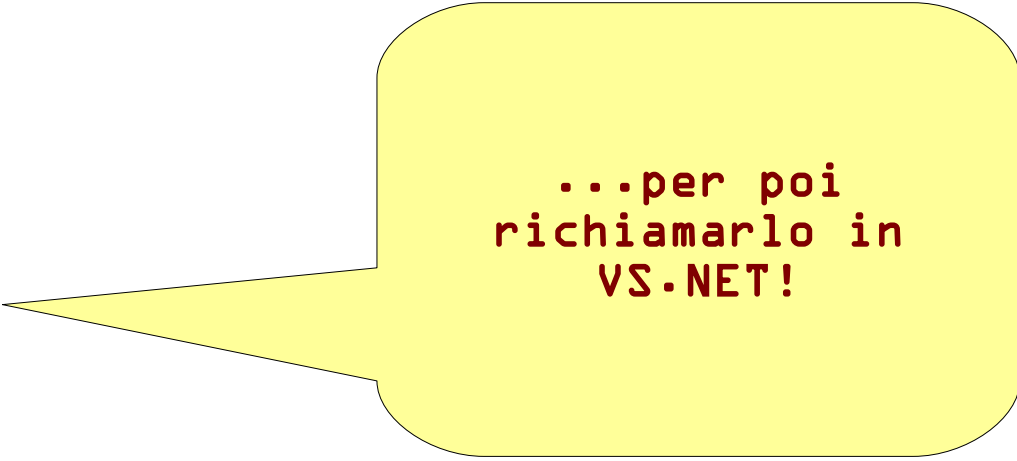
```
.text:00000000
.text:00000000 ; __stdcall Adler32(x, x, x)
.text:00000000 public _Adler32@12
.text:00000000 _Adler32@12 proc near
.text:00000000
.text:00000000 arg_0 = dword ptr 8
.text:00000000 arg_4 = dword ptr 0Ch
.text:00000000 arg_8 = dword ptr 10h
.text:00000000
.text:00000000 push ebp
.text:00000001 mov ebp, esp
.text:00000003 push edi
.text:00000004 push esi
.text:00000005 push ebx
.text:00000006 mov eax, [ebp+arg_8]
.text:00000009 mov ecx, [ebp+arg_4]
.text:0000000C mov ebx, [ebp+arg_8]
.text:0000000F and eax, 0FFFFh
.text:00000014 shr ebx, 10h
.text:00000017 mov esi, [ebp+arg_0]
.text:0000001A jmp short loc_47
.text:0000001C ;
.text:0000001C loc_1C: ; CODE XREF: Adler32(x,x,x)+49↓j
.text:0000001C mov edi, 15B0h
.text:00000021 cmp ecx, edi
.text:00000023 jnb short loc_27
.text:00000025 mov edi, ecx
.text:00000027 loc_27: ; CODE XREF: Adler32(x,x,x)+23↑j
.text:00000027 sub ecx, edi
.text:00000029 loc_29: ; CODE XREF: Adler32(x,x,x)+32↓j
.text:00000029 movzx edx, byte ptr [esi]
.text:0000002C add eax, edx
```

...con un semplice copia-incolla! :-o

```

;__stdcall Adler32(x, x, x)
.text:00000000 public _Adler32@12
.text:00000000 _Adler32@12 proc near
.text:00000000
.text:00000000 arg_0 = dword ptr 8
.text:00000000 arg_4 = dword ptr 0Ch
.text:00000000 arg_8 = dword ptr 10h
.text:00000000
.text:00000000 push ebp
.text:00000001 mov ebp, esp
.text:00000003 push edi
.text:00000004 push esi
.text:00000005 push ebx
.text:00000006 mov eax, [ebp+arg_8]
.text:00000009 mov ecx, [ebp+arg_4]
.text:0000000C mov ebx, [ebp+arg_8]
.text:0000000F and eax, 0FFFFh
.text:00000014 shr ebx, 10h
.text:00000017 mov esi, [ebp+arg_0]
.text:0000001A jmp short loc_47
.text:0000001C ; -----
.text:0000001C loc_1C: ; CODE XREF: Adler32(x,x,x)+49j
.text:0000001C mov edi, 15B0h
.text:00000021 cmp ecx, edi
.text:00000023 jnb short loc_27
.text:00000025 mov edi, ecx
.text:00000027
.text:00000027 loc_27: ; CODE XREF: Adler32(x,x,x)+23j
.text:00000027 sub ecx, edi
.text:00000029
.text:00000029 loc_29: ; CODE XREF: Adler32(x,x,x)+32j
.text:00000029 movzx edx, byte ptr [esi]
.text:0000002C add eax, edx
.text:0000002E inc esi
.text:0000002F add ebx, eax
.text:00000031 dec edi
.text:00000032 jnz short loc_29
.text:00000034 mov edi, 0FFF1h
.text:00000039 xor edx, edx
.text:0000003B div edi
.text:0000003D push edx
.text:0000003E mov eax, ebx
.text:00000040 xor edx, edx
.text:00000042 div edi
.text:00000044 mov ebx, edx
.text:00000046 pop eax
.text:00000047
.text:00000047 loc_47: ; CODE XREF: Adler32(x,x,x)+1Aj
.text:00000047 test ecx, ecx
.text:00000049 jnz short loc_1C
.text:0000004B shl ebx, 10h
.text:0000004E add eax, ebx
.text:00000050 pop ebx
.text:00000051 pop esi
.text:00000052 pop edi
.text:00000053 leave
.text:00000054 retn 0Ch
.text:00000054 _Adler32@12 endp

```





```

adler32_jda.txt
; __stdcall Adler32(x, x, x)
.text:00000000      public _Adler32@12
.text:00000000      _Adler32@12      proc near
.text:00000000
.text:00000000      arg_0           = dword ptr 8
.text:00000000      arg_4           = dword ptr 0Ch
.text:00000000      arg_8           = dword ptr 10h
.text:00000000
.text:00000000      push          ebp
.text:00000001      mov           ebp, esp
.text:00000003      push          edi
.text:00000004      push          esi
.text:00000005      push          ebx
.text:00000006      mov           eax, [ebp+arg_8]
.text:00000009      mov           ecx, [ebp+arg_4]
.text:0000000C      mov           ebx, [ebp+arg_8]
.text:0000000F      and           eax, 0FFFFh
.text:00000014      shr           ebx, 10h
.text:00000017      mov           esi, [ebp+arg_0]
.text:0000001A      jmp          short loc_47
.text:0000001C ; -----
.text:0000001C      loc_1C:         ; C
.text:0000001C      mov           edi, 15B0h
.text:00000021      cmp           ecx, edi
.text:00000023      jnb          short loc_27
.text:00000025      mov           edi, ecx
.text:00000027
.text:00000027      loc_27:         ; C
.text:00000027      sub           ecx, edi
.text:00000029
.text:00000029      loc_29:         ; C
.text:00000029      movzx        edx, byte ptr [esi]
.text:0000002C      add           eax, edx
.text:0000002E      inc           esi
.text:0000002F      add           ebx, eax
.text:00000031      dec           edi
    
```

```

Module1.vb*
Module1
Main
Private Function Adler32(ByVal p_arr As Byte(), ByVal lwAdler As Integer) As
    Dim retValue As UInt32 = 0
    '[i] step 1: instantiate the CPU engine
    Dim emulator As Emulator = New Emulator
    '[i] step 2: instantiate the x86 application
    Dim application As IApplication = emulator.X86
    With application
        .InitCPU()
        .Logical.XOR(DWordPtr.ESI, DWordPtr.ESI)
        .DataTransfer.MOV(DWordPtr.EAX, lwAdler)
        .DataTransfer.MOV(DWordPtr.ECX, p_arr.Length)
        .DataTransfer.MOV(DWordPtr.EBX, lwAdler)
        .Logical.AND(DWordPtr.EAX, &HFFFF)
        .Shift.SHR(DWordPtr.EBX, &H10)
        GoTo loc_47
    loc_1C:
        .DataTransfer.MOV(DWordPtr.EDI, &H15B0)
        .BinaryArithmetic.CMP(DWordPtr.ECX, DWordPtr.EDI)
        If .ControlTransfer.JNB Then GoTo loc_27
        .DataTransfer.MOV(DWordPtr.EDI, DWordPtr.ECX)
    loc_27:
        .BinaryArithmetic.SUB(DWordPtr.ECX, DWordPtr.EDI)
    loc_29:
        .DataTransfer.MOVZX(DWordPtr.EDX, p_arr(.DWordPtr.ESI))
        .BinaryArithmetic.ADD(DWordPtr.EAX, DWordPtr.EDX)
        .BinaryArithmetic.INC(DWordPtr.ESI)
        .BinaryArithmetic.ADD(DWordPtr.EBX, DWordPtr.EAX)
        .BinaryArithmetic.DEC(DWordPtr.EDI)
        If .ControlTransfer.JNZ Then GoTo loc_29
        .DataTransfer.MOV(DWordPtr.EDI, &HFFF1)
        .Logical.XOR(DWordPtr.EDX, DWordPtr.EDX)
        .BinaryArithmetic.DIV(DWordPtr.EDI)
        .DataTransfer.PUSH(DWordPtr.EDX)
        .DataTransfer.MOV(DWordPtr.EAX, DWordPtr.EBX)
        .Logical.XOR(DWordPtr.EDX, DWordPtr.EDX)
        .BinaryArithmetic.DIV(DWordPtr.EDI)
    
```



Goliath .NET CPU Emulator