_AssembleIt2()

von Andy bei AutoIt.de

Version vom 11.06.2023

Wie _AssembleIt2() funktioniert

_AssembleIt2() assembliert und führt x32 und x64 Code in Assemblersprache in AutoIt aus. Der Assembler ist FASM in der Version 1.72 und in AssembleIt() enthalten. Somit können (fast) sämtliche Funktionen von FASM wie bspw. Makros in AutoIt-Code eingebunden und ausgeführt werden.

Scite als Editor ist zwingend erforderlich!

Zunächst muss in Scite mit

#AutoIt3Wrapper_UseX64=n der 32-Bit-Modus, oder mit

#AutoIt3Wrapper_UseX64=y der 64-Bit-Modus ausgewählt werden.

Dann muss

#include <Assembleit2_64.au3> eingebunden werden.

Der zu assemblierende Code wird zwischen #cs und #ce eingefasst. Hinter #cs muss der Funktionsname, der später in _AssembleIt2() verwendet wird, angegeben werden. Es können mehrere Bereiche mit unterschiedlichen Funktionsnamen verwendet werden. Die Aufrufkonventionen müssen beachtet werden, Standard ist cdecl. Beim 64-Bit-Modus die Aufrufkonvention beachten!

Beispiel für AutoIt Code in Scite:

```
#cs IntegerAddition          ;Funktionsname
    use32                    ;für 32-Bit-Code oder use64 für 64-Bit-Code
    mov eax,[esp+4]          ;erster Parameter vom Stack
    mov ebx,[esp+8]          ;zweiter Parameter vom Stack
    add eax,ebx              ;addieren
    ret                      ;Rückgabe in eax bzw. rax, _AssembleIt2() bereinigt den Stack!
#ce
#cs FloatSubtraktion         ;Funktionsname
    Finit                    ;Init FPU and clear FPU stack
    fld dword[esp+8]         ; ST0 = [esp+8]
    fld dword[esp+4]         ; ST0 = [esp+4]        ST1 = [esp+8]
    fsub ST0 , ST1           ;ST0 = ST0 – ST1       ST0 = [esp+4] – [esp+8]
    ret                      ;bei „Float“ Rückgabe ST0, _AssembleIt2() bereinigt den Stack!
#ce
```

Es gibt folgende Funktionsparameter für _AssembleIt2():

_AssembleIt2(Rückgabetyp, Funktionsname, Typ1, Parameter1, Typ2, Parameter2,……)

Rückgabetyp und Datentypen sind bspw. INT, UINT, FLOAT, PTR und weitere, siehe FASM Hilfedatei und AutoIt Hilfedatei zu DllStructCreate().

Für oben genannte Beispiele ergibt das folgende Zeilen AutoIt-Code:

Local $a = 5.3, $b=3.1 ;Float muss zwingend mit Nachkommastellen angegeben werden

$Ergebnis_Addition = _AssembleIt2(„int“, „ IntegerAddition “, „int”, $a, “int”, $b) ; 5 + 3 = 8

$Ergebnis_Subtraktion = _AssembleIt2(„float“, „FloatSubtraktion“, „float”, $a, “float”, $b) ; 5.3 – 3.1 = 2.2

_AssembleIt2() assembliert den Code und führt ihn aus, als Ergebnis für die IntegerAddition erhält man 8, bei der FloatSubtraktion 2.2

Wenn man eine andere Aufrufkonvention benutzen möchte, wird diese Information zusammen mit dem Rückgabetyp dem Assembler mitgeteilt:

$ret = _AssembleIt2(„uint:cdecl“, „StringFunktion“, „str”, $a) ;Standard cdecl

$ret = _AssembleIt2(„uint:fastcall“, „Funktion“, „ptr”, $a) ;fastcall

Der Assembler verarbeitet die Prozessorbefehle genau so ab, wie sie im Programm stehen. Daher arbeitet auch der Prozessor die Befehle so ab, wie sie im Programm stehen! Es findet keine Fehlerüberprüfung oder sonstiger Eingriff in das Programm statt. Der Programmierer ist, wie immer bei Assemblerprogrammen, komplett für seinen Code verantwortlich!

Syntaxfehler werden vom Assembler erkannt, _AssembleIt() zeigt ein Infofenster und springt in Scite zum fehlerhaften Code.

Wenn die Aufrufkonventionen nicht beachtet werden, crashed AutoIt! Wenn der Stackpointer bei der Rückgabe nicht auf den richtigen Speicherbereich zeigt, crashed AutoIt! Wenn in einen nicht reservierten Speicherbereich geschrieben oder davon gelesen wird, crashed AutoIt!

Die Beispielscripte zeigen diverse Methoden zur schnellen Verarbeitung von Daten. Bei der Verwendung von _AssembleIt2() ist durch die geringe Latenz von nur einigen Millisekunden je nach Algorithmus eine Geschwindigkeitssteigerung von Faktor 1000 gegenüber reinem AutoIt-Code möglich.

_AssembleIt2() ist in der Lage, Binärcode zu erstellen, der dann in AutoIt auch ohne _AssembleIt2() aufgerufen werden kann. Das ist die schnellste Methode zur Ausführung von Maschinencode.

AutoItvariablen im Assemblercode

Bei dieser Verwendung von _AssembleIt2() können im Assemblercode auch AutoItvariablen als **Eingabeparameter** genutzt werden. Beispiel:

mov eax, $Autoitvariable1 ;erster Parameter

mov ebx, $AutoItvariable2 ;zweiter Parameter

In _AssembleIt2() können dann die Eingabetypen und Variablen entfallen.

Local $Autoitvariable1=22, $Autoitvariable2 = 33

$Ergebnis_Addition = _AssembleIt2(„int“, „Addition“) ; 22 + 33 = 55

_AssembleIt2() assembliert und führt das Script in wenigen Millisekunden aus, der größte Geschwindigkeitsgewinn ergibt sich aber aus der Verwendung des assemblierten Codes direkt aus dem Speicher per DllCallAddress(). Dazu später.

***mov $AutoItvariable , ecx ; * ist nicht möglich! ***

64-Bit-Modus

Im 64-Bit-Modus ist die Verwendung des Debuggers aktuell nicht möglich!

Die in _AssembleIt() integrierte FASM.dll gibt es nur in einer 32-Bit-Version und kann nicht von einem 64-Bit-Script ausgeführt werden. Die 32-Bit-FASM.dll ist aber in der Lage, 64-Bit-Code zu assemblieren.

Damit der 64-Bit-Assemblercode von _AssembleIt() ausgeführt werden kann, ist die 32-Bit-Hilfsdatei „AssembleIt2_Helper64.au3“ nötig.

Diese Datei wird, wenn sie nicht im aktuellen Dateipfad existiert, als „AssembleIt2_Helper64.EXE“ erstellt und gestartet.

Das AutoItscript übergibt dabei den 64-Bit-Assemblercode an die „AssembleIt2_Helper64.EXE“, der Code wird dort im 32-Bit-Modus assembliert und über einen von beiden Programmen gemeinsam genutzten Speicherbereich an _AssembleIt() zurückgegeben und im 64-Bit-Modus ausgeführt.

Vor Beendigen des Scripts sollte diese Datei in AutoIt mit Processclose($AssembleIt2_Helper64pid) aus dem Speicher entfernt werden.

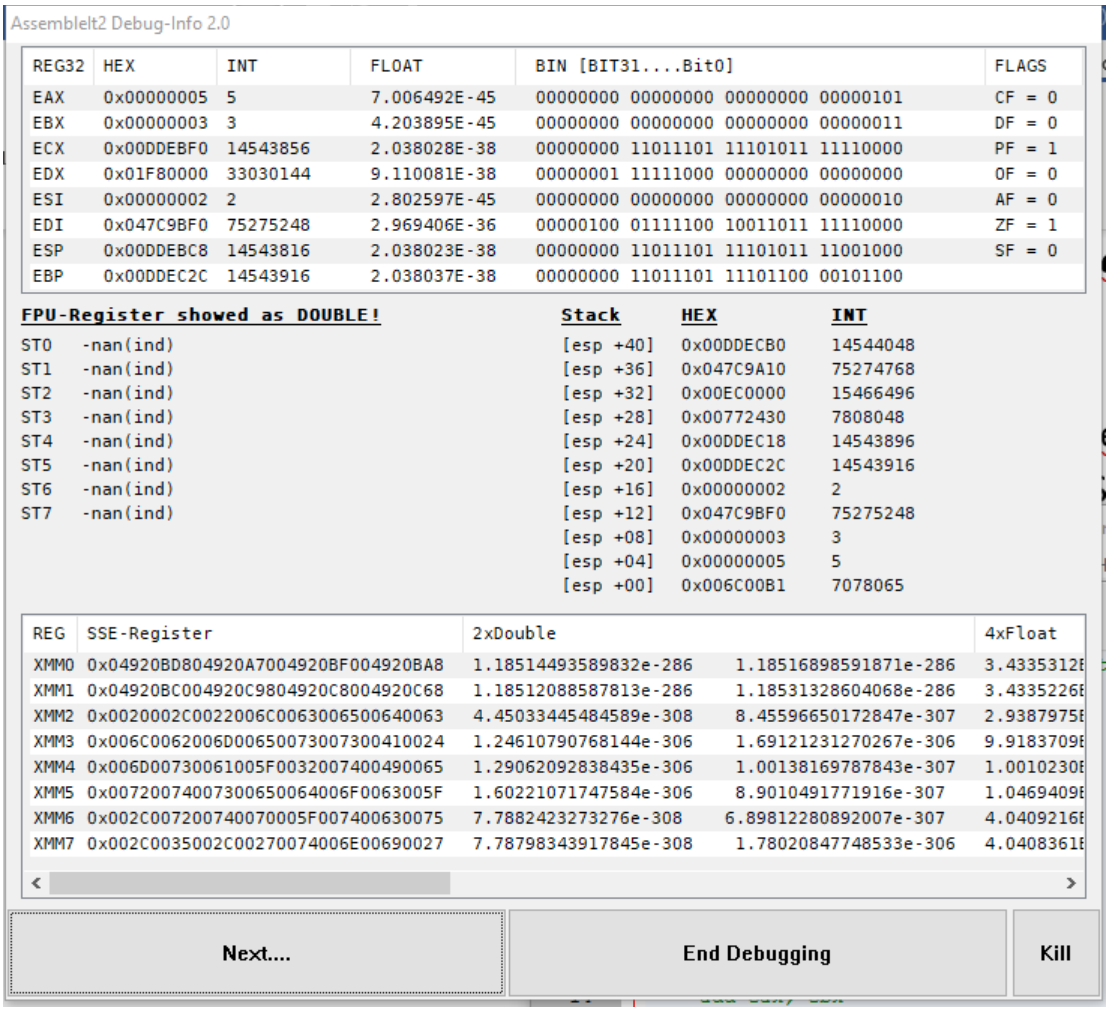
Im 64-Bit-Modus gelten die entsprechenden Aufrufkonventionen!

Debugger

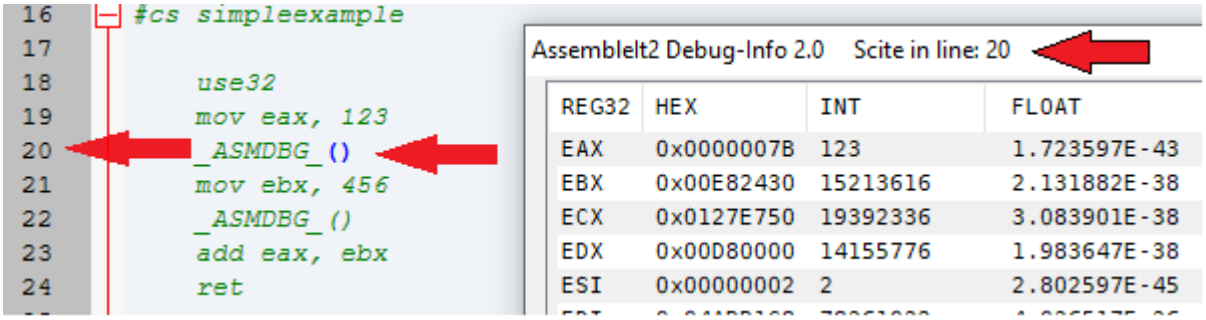
Um Assemblercode zu debuggen, ist der Zugriff auf Prozessorregister und Flags nötig. Beispiele für die Verwendung des Debuggers finden sich im Script „Example Debugger.au3“

Aktuell kann der Debugger nur im x86-Modus verwendet werden!

Mit dem Befehl ***_ASMDBG_()*** innerhalb des Assemblercodes wird das Debug-Fenster aufgerufen.



Der Button „Next“ führt den Assemblercode bis zum nächsten Vorkommen von ***_ASMDBG_()*** aus. Dabei wird die Position im Script in Scite aktualisiert, sichtbar durch die blauen Klammern **()**. Die aktuelle Zeile wird zusätzlich im Titelfenster angezeigt



Der Button „End Debugging“ überspringt alle weiteren Vorkommen von ***_ASMDBG_()*** im Code und führt den Assemblercode aus, bis er endet.

Der Button „Kill“ beendet den Assemblercode und das AutoIt-Script.

In ***_ASMDBG_()*** können auch die Inhalte der Prozessorregister mit angegeben werden um Assembler-Code, bspw. in Schleifen, so lange auszuführen, bis der Inhalt des Prozessorregisters erreicht ist.

ASMDBG("$EAX=50") zeigt erst das Debuggerfenster, wenn der Inhalt des EAX-Registers 50 beträgt.

Aktuell können folgende Register benutzt werden:

"$EAX", "$EBX", "$ECX", "$EDX", "$ESI", "$EDI", "$ESP", "$EBP"

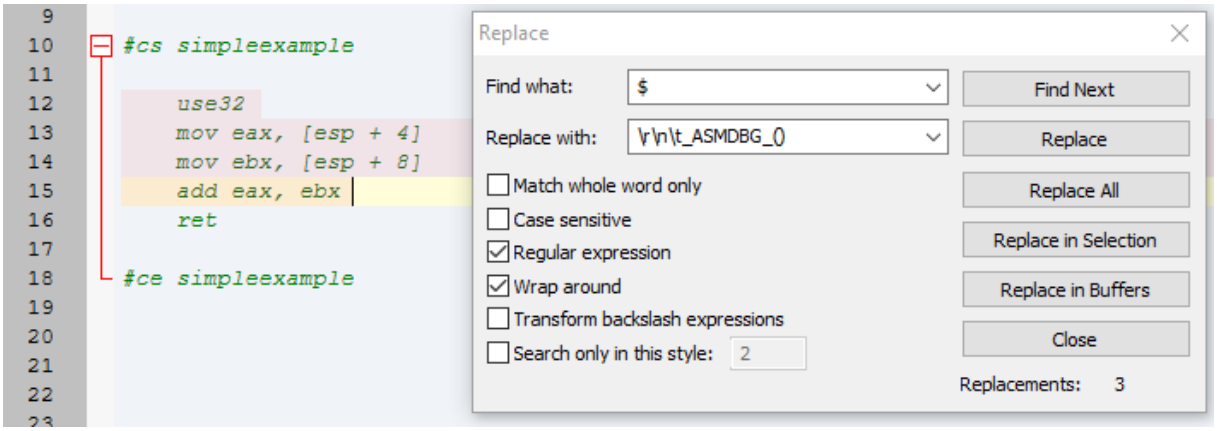
Es ist nicht zu empfehlen, sehr lange dauernde Schleifen auszuführen, wenn Registerinhalte angegeben werden.

```
mov eax,0
_schleife:                  ;for eax=0 to 10000
_ASMDBG_("$EAX=5000")       ;Debuggerfenster nur anzeigen, wenn eax=5000!
add eax,1                   ;eax = eax + 1
cmp eax,10000               ;eax = 10000?
jne _schleife               ;springe zu _schleife: wenn eax<>10000
```

führt dazu, dass das Debuggerfenster 5000 Mal aufgerufen wird und flackert!

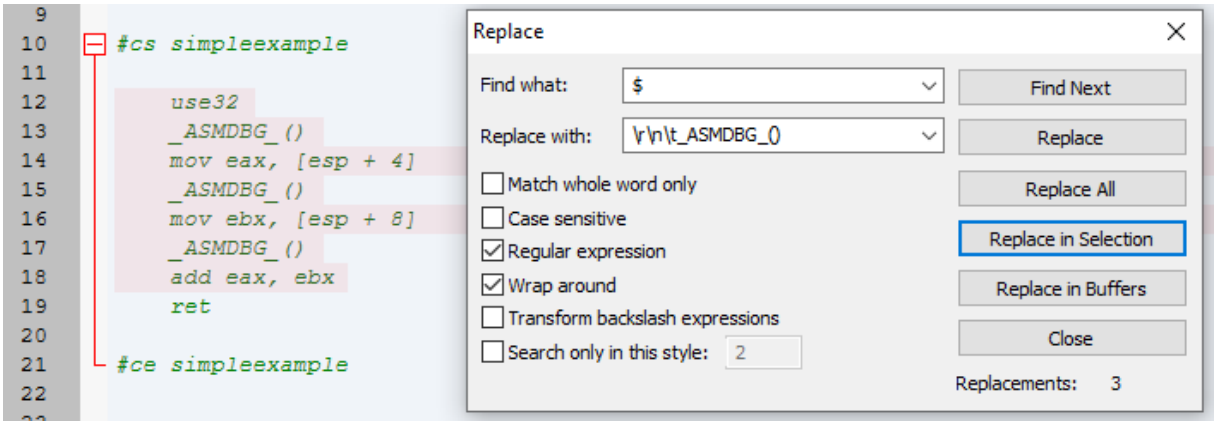
Mehrere _ASMDBG_() mit Find/Replace in Scite

Um ***_ASMDBG_()*** in einem Bereich einzufügen, kann man in Scite mit Ctrl-H das Suchen/Ersetzen-Fenster aufrufen und folgendes eintragen und den Haken bei Regular Expression setzen. Dann den Bereich im Script markieren



und dann den Button „Replace in Selection“ klicken.

Das führt dann zum Ersetzen von CR und LF in diesem Bereich mit TAB und ***_ASMDBG_()***



Assemblierten Code zurückgeben

Mit dem Rückgabetyp „retbinary“ ist es möglich, den assemblierten Code als Binärstring zurückgeben zu lassen. Man kann diesen Code dann in anderen Programmen verwenden, ihn direkt ausführen ohne Verwendung von AssembleIt2(), oder in einem Disassembler anzeigen lassen.

Wenn der Binärcode erstellt wird, werden die _ASMDBG_() Aufrufe des Debuggers im Code nicht beachtet!

```
#cs IntegerAddition          ;Funktionsname
    use32                    ;für 32-Bit-Code oder use64 für 64-Bit-Code
    mov eax,[esp+4]          ;erster Parameter vom Stack
    mov ebx,[esp+8]          ;zweiter Parameter vom Stack
    add eax,ebx              ;addieren
    ret                      ;Rückgabe in eax bzw. rax, _AssembleIt2() bereinigt den Stack!
#ce
```

$ret = _AssembleIt2(„retbinary“, „IntegerAddition“)

gibt in der Autoitvariable ***$ret = 0x8B4424048B5C240801D8C3*** zurück. Dieser Code kann im Speicher ohne die Verwendung von AutoIt aufgerufen werden.

AutoItscript mit ausführbarem Binärcode erstellen

_AssembleIt2() kann auch ein AutoItscript erstellen, um den Maschinencode direkt im Speicher auszuführen. Dazu ist als letzter Datentyp im Aufruf von _AssembleIt2() ***„retbinary“*** erforderlich, und zusätzlich als letzten Datentyp die AutoItvariable ***@ScriptLineNumber.***

$ret = _AssembleIt2(„uint“, „Funktion“, “uint”,$var, “retbinary”, @ScriptLineNumber)

```
#AutoIt3Wrapper_UseX64=n                        ;32Bit-Modus
#include-once
#include <assembleit2_64.au3>

#cs IntegerAddition                             ;Funktionsname
    use32                                       ;für 32-Bit-Code
    mov eax,[esp+4]                             ;erster Parameter vom Stack
    mov ebx,[esp+8]                             ;zweiter Parameter vom Stack
    add eax,ebx                                 ;addieren
    ret                                         ;Rückgabe in eax bzw. rax, _AssembleIt2() bereini
#ce IntegerAddition

Local $a = 5, $b = 3
$ret = _AssembleIt2("int", "IntegerAddition", "int", $a, "int", $b, "retbinary", @ScriptLineNumber)
ConsoleWrite("$ret = " & $ret & @CRLF)

;the following code of the function INTEGERADDITION is returned by AssebleIt2_64 and copied into clipboard!
Global $tCodeBuffer = DllStructCreate("byte[11]") ;reserve Memory for opcodes
DllStructSetData($tCodeBuffer, 1,"0x8B4424048B5C240801D8C3") ;write opcodes into memory
$ret=DllCallAddress("int:cdecl", DllStructGetPtr($tCodeBuffer), "int", $a, "int", $b)

$ret = 8
+>09:07:37 AutoIt3.exe ended.rc:0
+>09:07:37 AutoIt3Wrapper Finished.
```

_AssembleIt2() schreibt das erstellte Script in die Scite-Konsole und auch in das Clipboard. Das Script kann nun entweder im Programm den Aufruf von _AssembleIt2() ersetzen, oder in einem anderen Script die Funktion bereitstellen, ohne dort _AssembleIt2() aufrufen zu müssen!

```
$a=5
$b=3
Global $tCodeBuffer = DllStructCreate("byte[11]") ;reserve Memory for opcodes
DllStructSetData($tCodeBuffer, 1,"0x8B4424048B5C240801D8C3") ;write opcodes into memory
$ret=DllCallAddress("int:cdecl", DllStructGetPtr($tCodeBuffer), "int", $a, "int", $b)
ConsoleWrite("$ret = " & $ret[0] & @crlf)

+>Setting Hotkeys...--> Press Ctrl+Alt+Break to Restart or Ctrl+BREAK to Stop.
@$ret = 8
+>09:16:56 AutoIt3.exe ended.rc:0
```

DllCallAddress gibt ein Array zurück! In diesem sind die Aufrufparameter und der Rückgabewert enthalten.

64-Bit-Code:

```
#AutoIt3Wrapper_UseX64=y    ;64Bit-Modus
#include <assembleit2_64.au3>
#cs simpleexample
    use64
    push rdi                                ;save volatile registers , remember https://learn.microsoft.com/en-us/cpp/build/x64-calling-convention?v
    push rsi
    push rbx
    push rbp
    mov rax, rcx                            ;first variable from rxc
    mov rbx, rdx                            ;second variable from rdx
    add rax, rbx                            ;add them..
    pop rbp                                 ;pop saved volatile registers
    pop rbx
    pop rsi
    pop rdi
    ret                                     ;...eax/rax is returned
#ce simpleexample
$a = 5
$b = 3
$ret = _AssembleIt2("uint64", "simpleexample", "uint64", $a, "uint64", $b, "retbinary", @ScriptLineNumber)

;the following code of the function SIMPLEEXAMPLE is returned by _AssebleIt2() and copied into clipboard!
Local $aStructMem = DllCall("kernel32.dll", "ptr", "VirtualAlloc", "ptr", 0, "ulong_ptr", DllStructGetSize(DllStructCreate("byte[18]")), "dword", 4096, "dword", 64)
Local $tCodeBuffer = DllStructCreate("byte[18]", Number($aStructMem[0]))  ;Address aligned 64 and Memory is  $MEM_COMMIT, $PAGE_EXECUTE_READWRITE
DllStructSetData($tCodeBuffer, 1,"0x575653554889C84889D34801D85D5B5E5FC3") ;write opcodes into memory
$ret=DllCallAddress("uint64:cdecl", DllStructGetPtr($tCodeBuffer), "uint64", $a, "uint64", $b)

+>13:22:04 AutoIt3.exe ended.rc:0
```

Code per Ctrl-V in neues Script einfügen, Variablen anpassen, Ausführen:

```
#AutoIt3Wrapper_UseX64=y    ;64Bit-Modus
$a = 5
$b = 3
Local $aStructMem = DllCall("kernel32.dll", "ptr", "VirtualAlloc", "ptr", 0, "ulong_ptr", DllStructGetSize(DllStructCreate(
Local $tCodeBuffer = DllStructCreate("byte[18]", Number($aStructMem[0]))  ;Address aligned 64 and Memory is  $MEM_COMMIT, $
DllStructSetData($tCodeBuffer, 1,"0x575653554889C84889D34801D85D5B5E5FC3") ;write opcodes into memory
$ret=DllCallAddress("uint64:cdecl", DllStructGetPtr($tCodeBuffer), "uint64", $a, "uint64", $b)
ConsoleWrite('$ret = ' & $ret[0] & @CRLF )
```

```
+>Setting Hotkeys...--> Press Ctrl+Alt+Break to Restart or Ctrl+BREAK to Stop.
$ret = 8
+>13:26:57 AutoIt3.exe ended.rc:0
```

Für die Ausführung des 64-Bit-Codes direkt aus dem Speicher ist die Datei „AssembleIt2_Helper64.EXE“ nicht mehr erforderlich!

Aufrufkonventionen

Während bei _AssembleIt() beim Rückgabetyp nicht explizit :cdecl aufgerufen werden muß, ist dies bei der Ausführung von 64-Bit-Code und der Verwendung von DllCalladdress() zwingend notwendig! Bei Verwendung einer anderen Aufrufkonvention muss diese immer mit angegeben werden!

$ret = _AssembleIt2(„uint“, „Funktion“, “uint”,$var) ;kein :cdecl erforderlich

$ret = _AssembleIt2(„uint:cdecl“, „Funktion“, “uint”,$var) ; kein :cdecl erforderlich

$ret = _AssembleIt2(„uint:stdcall“, „Funktion“, “uint”,$var) ; stdcall erforderlich, Stack aufräumen!!

$ret=DllCallAddress("uint64:cdecl", DllStructGetPtr($tCodeBuffer), "uint64", $a) ; :cdecl erforderlich!