_AssembleIt2()

by Andy on AutoIt.de

Version from 11.06.2023

How _AssembleIt2() works

_AssembleIt2() assembles and executes x32 and x64 code in assembly language in AutoIt. The assembler is FASM version 1.72 and integrated in AssembleIt(). This means that (almost) all functions of FASM, e.g. Macros can be included in AutoIt code and executed.

Scite as an editor is mandatory!

First of all, Scite must be used with

#AutoIt3Wrapper_UseX64=n to use the 32-bit mode, or with

#AutoIt3Wrapper_UseX64=y to use the 64-bit-mode.

Then you have to include

#include <Assembleit2_64.au3>.

The code to be assembled is enclosed between #cs and #ce. After #cs, the function name that will later be used in _AssembleIt2() must be specified. Multiple scopes with different function names can be used. The calling conventions must be observed, default is cdecl. In 64-bit mode, observe the calling convention!

Example of AutoIt code in Scite:

```
#cs IntegerAddition             ; Function name
        use32                   ;for 32-bit code or use64 for 64-bit code
        mov eax,[esp+4]         ;first parameter of the stack
        mov ebx,[esp+8]         ;second parameter of the stack
        add eax,ebx             ;add
        right                   ; Return in eax or rax, _AssembleIt2() cleans up the stack!
#ce
#cs FloatSubtraction            ; Function name
        Ends                    ;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]
        right                   ;at "float" return ST0, _AssembleIt2() cleans up the stack!
#ce
```

The following function parameters are available for _AssembleIt2():

_AssembleIt2(Returntype, Functionname, Type1, Parameter1, Type2, Parameter2,……)

Return type and data types are e.g. INT, UINT, FLOAT, PTR and others, see FASM help file and AutoIt help file for DllStructCreate().

For the above examples, this results in the following lines of AutoIt code:

Local $a = 5.3, $b=3.1 ;Float must be specified with decimal places

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

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

_AssembleIt2() assembles the code and executes it, the result for the IntegerAddition is 8, for the FloatSubtraction 2.2

If you want to use a different calling convention, this information is communicated to the assembler together with the return type:

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

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

The assembler processes the processor instructions exactly as they appear in the program. Therefore, the processor also processes the instructions as they are in the program! There is no error checking or other intervention in the program. As always with assembly language programs, the programmer is completely responsible for his code!

Syntax errors are detected by the assembler, _AssembleIt2() shows an info window and jumps to the faulty code line in Scite.

If the calling conventions are not respected, AutoIt will crash! If the stack pointer does not point to the correct memory area when it is returned, AutoIt will crash! When writing to or reading from an unreserved memory area, AutoIt will crash!

The example scripts show various methods for fast processing of data. When using _AssembleIt2(), the low latency of only a few milliseconds makes it possible to increase the speed by a factor of 1000 compared to pure AutoIt code, depending on the algorithm.

_AssembleIt2() is able to create binary code, which can then be called in AutoIt without _AssembleIt2(). This is the fastest way to execute machine code.

AutoIt variables in assembly code

With this use of _AssembleIt2(), AutoIt variables can also be used as input parameters in the assembly code . Example:

mov eax, $Autoitvariable1 ;first parameter

mov ebx, $AutoItvariable2 ;second parameter

In _AssembleIt2(), the input types and variables can then be omitted.

Local $Autoitvariable1=22, $Autoitvariable2 = 33

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

_AssembleIt2() assembles and executes the script in a few milliseconds, but the greatest speed gain comes from using the assembled code directly from memory via DllCallAddress(). More on that later.

***mov $AutoItvariable , ecx ; * is not possible! ***

64-bit mode

In 64-bit mode, the use of the debugger is currently not possible!

The FASM .dll built into _AssembleIt() is only available in a 32-bit version and cannot be executed by a 64-bit script. However, the 32-bit FASM .dll is capable of assembling 64-bit code.

In order for the 64-bit assembly code of _AssembleIt() to be executed, the 32-bit auxiliary file "AssembleIt2_Helper64.au3" is required.

This file, if it does not exist in the current file path, is created as "AssembleIt2_Helper64.EXE" and started.

The AutoItscript passes the 64-bit assembly code to the "AssembleIt2_Helper64.EXE", where the code is assembled in 32-bit mode and returned to _AssembleIt() via a memory area shared by both programs. After returning the 64-bit-code, AssembleIt2() execute this code in 64-bit mode.

Before exiting the script, the helping file should be removed from memory in AutoIt using Processclose($AssembleIt 2_Helper64pid).

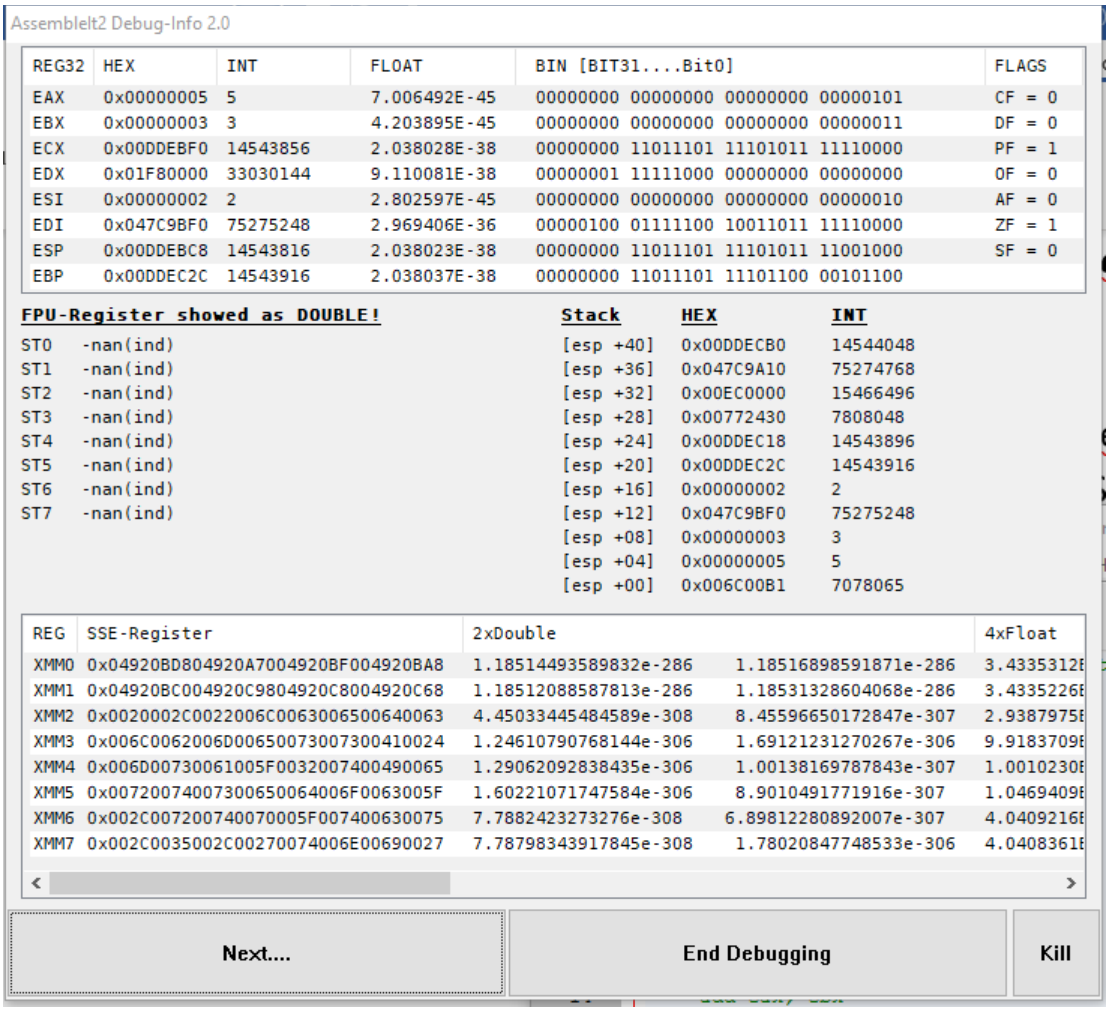
In 64-bit mode, the appropriate calling conventions apply!

Debugger

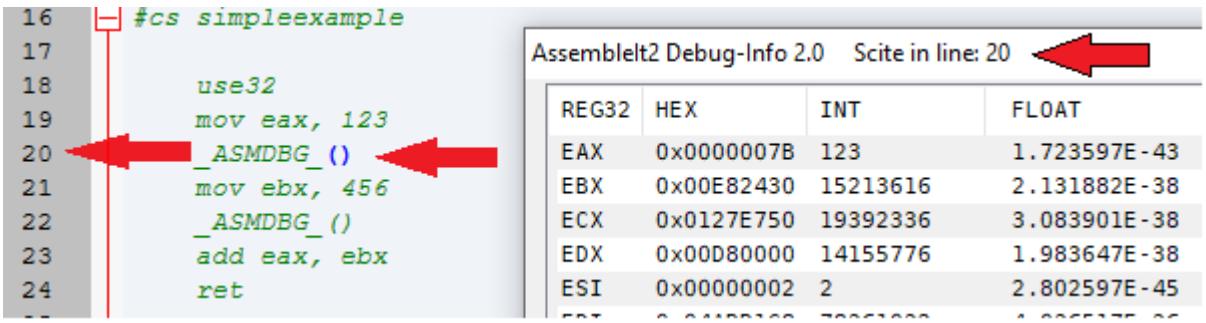
In order to debug assembly code, access to processor registers and flags is required. Examples of how to use the debugger can be found in the script "Example Debugger.au3"

Currently, the debugger can only be used in x86 mode!

The ***_ASMDBG_()*** command within the assembly code invokes the debug window.



The "Next" button executes the assembly code until the next occurrence of ***_ASMDBG_().*** The position in the script is updated in Scite, visible by the blue brackets **()**. The current line is also displayed in the title window



The "End Debugging" button skips all further occurrences of ***_ASMDBG_()*** in the code and executes the assembly code until it ends.

The "Kill" button terminates the assembly code and the AutoIt script.

In ***_ASMDBG_()*** you can also specify the contents of the processor registers in order to execute assembly code, e.g. in loops, until the contents of the processor register are reached.

ASMDBG("$EAX=50") does not show the debugger window until the contents of the EAX register are 50.

Currently, the following registers can be used:

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

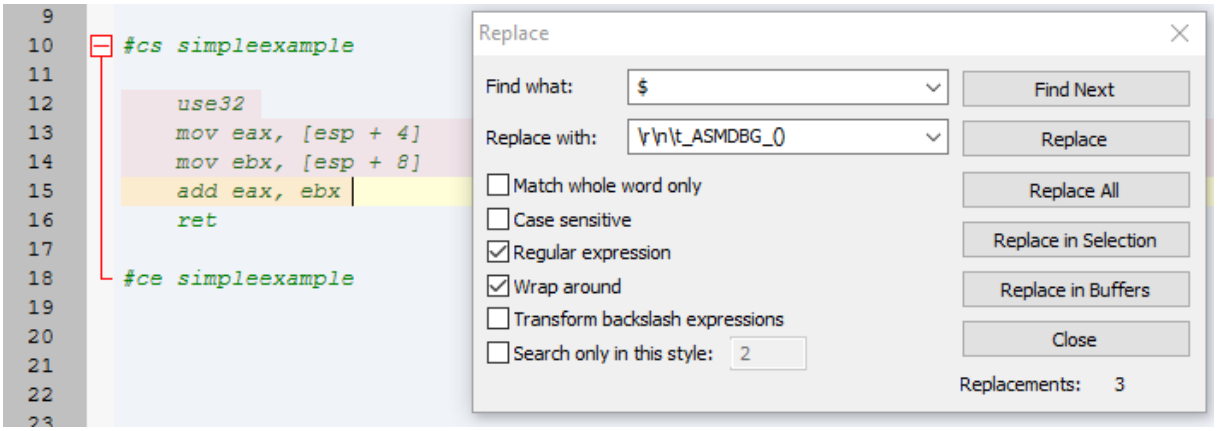
It is not recommended to perform very long loops when specifying register contents.

```
mov eax,0
_loop:                          ;for eax=0 to 10000
_ASMDBG_("$EAX=5000")           ;Debugger window only show if eax=5000!
add eax,1                       ;eax = eax + 1
cmp eax,10000                   ;eax = 10000?
jne _loop                       ;Jump To _loop: if EAX<>10000
```

causes the debugger window to be called 5000 times and flicker!

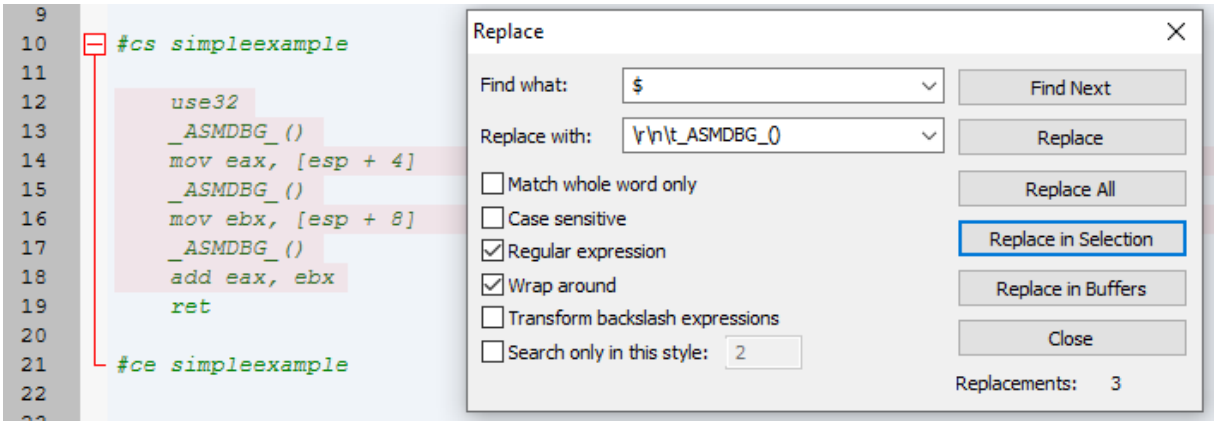
Multiple _ASMDBG_() with Find/Replace in Scite

To ***insert _ASMDBG_()*** multiple times in an area of your script, you can call up the Find/Replace window in Scite with Ctrl-H and enter the following: In the Find window insert a $ and in the replace window enter \r\n\t_ASMDBG_() and check the box for Regular Expression. Then select the area in the script where the _ASMDBG_() lines should be inserted.



Then click the "Replace in Selection" button.

This leads to replacing CR and LF in this area with TAB and ***_ASMDBG_(),*** shown the following picture:



Returning Assembled Code

With the return type "retbinary" it is possible to return the assembled code as a binary string. You can then use this code in other programs, run it directly without using AssembleIt2(), or display it in a disassembler.

When the binary code is built, the _ASMDBG_() calls of the debugger are ignored in the code!

```
#cs IntegerAddition          ; Funktionsname
    use32                    ;for 32-bit code or use64 for 64-bit code
    mov eax,[esp+4]          ;first parameter of the stack
    mov ebx,[esp+8]          ;second parameter of the stack
    add eax,ebx              ;add
    right                    ; Return in eax or rax, _AssembleIt2() cleans up the stack!
#ce
```

$ret = _AssembleIt2("retbinary", "IntegerAddition")

returns $ret = 0x8B4424048B5C240801D8C3 in the autoit variable. This code can be invoked in memory without the use of AutoIt.

Create AutoItscript with executable binary code

_AssembleIt2() can also create an AutoItscript to execute the machine code directly in memory. For this purpose, "***retbinary" is required as the last data type in the call to _AssembleIt2(),*** and the AutoItvariable @ScriptLineNumber must be used as the last data type***.***

$ret = _AssembleIt2("uint", "function", "uint",$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() writes the created script to the scite console and also to the clipboard. The script can now either replace the call to _AssembleIt2() in the program, or provide the function in another script without having to call _AssembleIt2() there!

```
$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 returns an array! This contains the call parameters and the return value.

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
```

Insert code into new script via Ctrl-V, adjust variables, execute:

```
#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
```

To execute the 64-bit code directly from memory, the "AssembleIt2_Helper64.EXE" file is no longer required!

Example scripts are included in the _AssembleIt2() folder.

Calling conventions

While _AssembleIt() does not explicitly have to call :cdecl for the return type, this is absolutely necessary for the execution of 64-bit code and the use of DllCalladdress()! If a different calling convention is used, it must always be specified!

$ret = _AssembleIt2("uint", "function", "uint",$var) ;no :cdecl required

$ret = _AssembleIt2("uint:cdecl", "function", "uint",$var) ; No :cdecl required

$ret = _AssembleIt2("uint:stdcall", "function", "uint",$var) ; stdcall required, remember to clean up stack!!

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