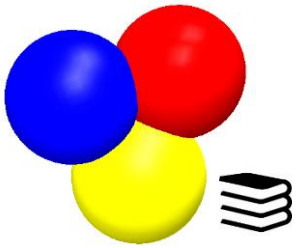

ObjAsm

x86 / x64

ObjMem Library

1. Introduction

ObjAsm components and projects take advantage from optimized assembler routines. This document describes those functions and procedures located in the open-source static library called **ObjMem.lib**. This precompiled code can be used in an OOP or plain assembler environment. All necessary files are provided in the **ObjAsm** package.

Implementation details can be checked at the source code level by reading the comments on each file.

Source code directories are divided in **32** bit, **64** bit and **Common** code. This last directory contains the bitness, platform, and/or string encoding independent code. The filename suffix of those files indicates their purpose.

The **ObjMem** library comes with an include file called **ObjMem.inc**, that conditional assembly to ensure that for each assembly target only the intended functions are visible. It also selects accordingly the matching equates and string encoding aliases.

2. Contents

3. Acknowledgements

I would like to express my very great appreciation to all whose valuable and constructive contributions made this work possible. Thank you!

Corrections, comments, suggestions, contributions, etc. may be sent to the MASM32 Forum, or directly mailed to:

ObjAsm@gmx.net

G. Friedrich,

August, 2022

4. Nomenclature

The following list describes the rules used to create the library:

1. X prefixes are used to denote a variable or register that can change according to the bitness assembly target. Example: xax means rax in 64 bits, while eax in 32 bits.
2. T file suffixes are used to denote a neutral string encoding.
3. X file suffixes are used to denote bitness-neutral code.
4. P file suffixes denote platform-independent code, usually leaf procedures.
5. Other file suffixes were used to identify the purpose of the code.

5. Abbreviations

BNC: Bitness Neutral Code
COM: Component Object Model
DLL: Dynamic Link Library
GUID: Globally Unique Identifier
HLL: High Level Language
ID: Identifier
IID: Interface Identifier
ZTC: Zero Terminating Character

6. Library Build

There are some .cmd files in the main directory of ObjMem that make the build easier. In most cases, **MakeObjMem3264.cmd** will do the job.

The **ObjMem.api** file contains the API definitions for the **RadASM** 2.x autocomplete feature. It must be copied to the …\RadASM\Masm folder and RadASM restarted.

Note: The BuildTest.cmd file is for procedure testing purposes only. Edit the file with the filename you want to check.

7. 32 Bit Code

```
Procedure:  aCRC32C
File:       \ObjAsm\Code\ObjMem\32\aCRC32C.asm
Purpose:    Compute the CRC-32C (Castagnoli), using the polynomial 11EDC6F41h from an aligned
            memory block.
Arguments:  Arg1: → Aligned memory block.
            Arg2: Memory block size in BYTEs.
Return:     eax = CRC32C.


Procedure:  ActivatePrevInstanceA
File:       \ObjAsm\Code\ObjMem\32\ActivatePrevInstanceA.asm
Purpose:    Activate a previously existing instance of an application.
Arguments:  Arg1: → ANSI application name.
            Arg2: → ANSI class name.
Return:     eax = TRUE if activated, otherwise FALSE.


Procedure:  ActivatePrevInstanceW
File:       \ObjAsm\Code\ObjMem\32\ActivatePrevInstanceW.asm
Purpose:    Activate a previously existing instance of an application.
Arguments:  Arg1: → WIDE application name.
            Arg2: → WIDE class name.
Return:     eax = TRUE if activated, otherwise FALSE.


Procedure:  AreVisualStylesEnabled
File:       \ObjAsm\Code\ObjMem\32\AreVisualStylesEnabled.asm
Purpose:    Determine if there is an activated theme for the running application
Arguments:  None.
Return:     eax = TRUE if the application is themed, otherwise FALSE.


Procedure:  bin2dwordA
File:       \ObjAsm\Code\ObjMem\32\bin2dwordA.asm
Purpose:    Load an ANSI string binary representation of a DWORD.
Arguments:  Arg1: → ANSI binary string.
Return:     eax = DWORD.


Procedure:  bin2dwordW
File:       \ObjAsm\Code\ObjMem\32\bin2dwordW.asm
Purpose:    Load an WIDE string binary representation of a DWORD.
Arguments:  Arg1: → Wide binary string.
Return:     eax = DWORD.


Procedure:  bin2qwordA
File:       \ObjAsm\Code\ObjMem\32\bin2qwordA.asm
Purpose:    Load an ANSI string binary representation of a QWORD.
Arguments:  Arg1: → ANSI binary string.
Return:     edx::eax = QWORD.


Procedure:  bin2qwordW
File:       \ObjAsm\Code\ObjMem\32\bin2qwordW.asm
Purpose:    Compute a WIDE string binary representation of a QWORD.
Arguments:  Arg1: → WIDE binary string.
Return:     edx::eax = QWORD.


Procedure:  Bmp2Rgn
File:       \ObjAsm\Code\ObjMem\32\Bmp2Rgn.asm
Purpose:    Create a GDI region based on a device dependant or independent bitmap (DDB or DIB).
            This region is defined by the non transparent area delimited by the transparent color.
Arguments:  Arg1: Bitmap handle.
            Arg2: RGB transparet color.
Return:     eax = Region handle or zero if failed.


Procedure:  BStrAlloc
File:       \ObjAsm\Code\ObjMem\32\BStrAlloc.asm
Purpose:    Allocate space for a BStr with n characters. The length field is set to zero.
Arguments:  Arg1: Character count.
Return:     eax → New allocated BStr or NULL if failed.
```

```
Procedure:  BStrCat
File:       \ObjAsm\Code\ObjMem\32\BStrCat.asm
Purpose:    Concatenate 2 BStrs.
Arguments:  Arg1: Destination BStr.
            Arg2: Source BStr.
Return:     Nothing.


Procedure:  BStrCatChar
File:       \ObjAsm\Code\ObjMem\32\BStrCatChar.asm
Purpose:    Append a character to the end of a BStr.
Arguments:  Arg1: Destination BStr.
            Arg2: Wide character.
Return:     Nothing.


Procedure:  BStrCCatChar
File:       \ObjAsm\Code\ObjMem\32\BStrCCatChar.asm
Purpose:    Append a WIDE character to a BStr with length limitation.
Arguments:  Arg1: → Destination BStr.
            Arg2: → Wide character.
Return:     Nothing.


Procedure:  BStrCECat
File:       \ObjAsm\Code\ObjMem\32\BStrCECat.asm
Purpose:    Concatenate 2 BStrs with length limitation and return the ending zero character
            address. The destination string buffer should have at least enough room for the
            maximum number of characters + 1.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → Source BStr.
            Arg3: Maximal number of charachters that the destination string can hold including
            the zero terminating character.
Return:     eax → ZTC.


Procedure:  BStrCECopy
File:       \ObjAsm\Code\ObjMem\32\BStrCECopy.asm
Purpose:    Copy the the source BStr with length limitation and return the ZTC address.
            The destination buffer should hold the maximum number of characters + 1.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → Source BStr.
            Arg3: Maximal number of charachters.
Return:     eax → ZTC.


Procedure:  BStrCNew
File:       \ObjAsm\Code\ObjMem\32\BStrCNew.asm
Purpose:    Allocate a new copy of the source BStr with length limitation.
            If the pointer to the source string is NULL or points to an empty string, BStrCNew
            returns NULL and doesn't allocate any heap space. Otherwise, StrCNew makes a duplicate
            of the source string. The maximal size of the new string is limited to the second
            parameter.
Arguments:  Arg1: → Source BStr.
            Arg2: Maximal character count.
Return:     eax → New BStr copy.


Procedure:  BStrCopy
File:       \ObjAsm\Code\ObjMem\32\BStrCopy.asm
Purpose:    Copy a BStr to a destination buffer.
Arguments:  Arg1: Destination BStr buffer.
            Arg2: Source BStr.
Return:     Nothing.


Procedure:  BStrCScan
File:       \ObjAsm\Code\ObjMem\32\BStrCScan.asm
Purpose:    Scan from the beginning of a BStr for a character with length limitation.
Arguments:  Arg1: → Source WIDE string.
            Arg2: Maximal character count.
            Arg3: Wide character to search for.
Return:     eax → Character address or NULL if not found.


Procedure:  BStrDispose
File:       \ObjAsm\Code\ObjMem\32\BStrDispose.asm
Purpose:    Free the memory allocated for the string using BStrNew, BStrCNew, BStrLENew or
            BStrAlloc.
            If the pointer to the string is NULL, BStrDispose does nothing.
```

```
Arguments:  Arg1: → BStr.
Return:     Nothing.


Procedure:  BStrECat
File:       \ObjAsm\Code\ObjMem\32\BStrECat.asm
Purpose:    Append a BStr to another and return the address of the ending zero character.
            BStrCat does not perform any length checking. The destination buffer must have room
            for at least BStrLength(Destination) + BStrLength(Source) + 1 characters.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → Source BStr.
Return:     eax → ZTC.


Procedure:  BStrECatChar
File:       \ObjAsm\Code\ObjMem\32\BStrECatChar.asm
Purpose:    Append a WIDE character to a BStr and return the address of the ZTC.
            BStrECatChar does not perform any length checking. The destination buffer must have
            enough room for at least BStrLength(Destination) + 1 + 1 characters.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → WIDE character.
Return:     eax → ZTC.


Procedure:  BStrECopy
File:       \ObjAsm\Code\ObjMem\32\BStrECopy.asm
Purpose:    Copy a BStr to a buffer and return the address of the ZTC.
            Source and destination strings may overlap.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → Source BStr buffer.
Return:     eax → ZTC.


Procedure:  BStrEnd
File:       \ObjAsm\Code\ObjMem\32\BStrEnd.asm
Purpose:    Get the address of the ZTC that terminates the string.
Arguments:  Arg1: → Source BStr.
Return:     eax → ZTC.


Procedure:  BStrEndsWith
File:       \ObjAsm\Code\ObjMem\32\BStrEndsWith.asm
Purpose:    Compare the ending of a BSTR.
Arguments:  Arg1: → Analized BSTR.
            Arg2: → Suffix BSTR.
Return:     eax = TRUE of the ending matches, otherwise FALSE.


Procedure:  BStrFillChr
File:       \ObjAsm\Code\ObjMem\32\BStrFillChr.asm
Purpose:    Fill a preallocated BSTR with a character.
Arguments:  Arg1: → String.
            Arg2: Character.
            Arg3: Character Count.
Return:     Nothing.


Procedure:  BStrLeft
File:       \ObjAsm\Code\ObjMem\32\BStrLeft.asm
Purpose:    Extract the left n characters of the source BStr.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → Source BStr.
Return:     eax = Number of copied characters, not including the ZTC.


Procedure:  BStrLength
File:       \ObjAsm\Code\ObjMem\32\BStrLength.asm
Purpose:    Determine the length of a BStr not including the ZTC.
Arguments:  Arg1: → Source BStr.
Return:     eax = Length of the string in characters.


Procedure:  BStrLRTrim
File:       \ObjAsm\Code\ObjMem\32\BStrLRTrim.asm
Purpose:    Trim blank characters from the beginning and the end of a BStr.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → Source BStr.
Return:     Nothing.


Procedure:  BStrLTrim
```

```
File:       \ObjAsm\Code\ObjMem\32\BStrLTrim.asm
Purpose:    Trim blank characters from the beginning of a BStr.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → Source BStr.
Return:     Nothing.


Procedure:  BStrMid
File:       \ObjAsm\Code\ObjMem\32\BStrMid.asm
Purpose:    Extract a substring from a BStr string.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → Source Bstr.
            Arg3: Start character index. Index ranges [1 .. String length].
            Arg3: Character count.
Return:     Nothing.


Procedure:  BStrMove
File:       \ObjAsm\Code\ObjMem\32\BStrMove.asm
Purpose:    Move part of a BStr. The ZTC is not appended automatically.
            Source and destination strings may overlap.
Arguments:  Arg1: → Destination buffer.
            Arg2: → Source BStr.
            Arg3: Character count.
Return:     Nothing.


Procedure:  BStrNew
File:       \ObjAsm\Code\ObjMem\32\BStrNew.asm
Purpose:    Allocate an new copy of the source BStr.
            If the pointer to the source string is NULL or points to an empty string, BStrNew
            returns NULL and doesn't allocate any heap space. Otherwise, BStrNew makes a duplicate
            of the source string.
            The allocated space is Length(String) + 1 character.
Arguments:  Arg1: → Source BStr.
Return:     eax → New BStr copy.


Procedure:  BStrRepChr
File:       \ObjAsm\Code\ObjMem\32\BStrRepChr.asm
Purpose:    Create a new BSTR filled with a given char.
Arguments:  Arg1: Used character.
            Arg2: Repetition count.
Return:     xax → New BSTR or NULL if failed.


Procedure:  BStrRight
File:       \ObjAsm\Code\ObjMem\32\BStrRight.asm
Purpose:    Copy the right n characters from the source string into the destination buffer.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → Source BStr.
            Arg3: Character count.
Return:     Nothing.


Procedure:  BStrRTrim
File:       \ObjAsm\Code\ObjMem\32\BStrRTrim.asm
Purpose:    Trim blank characters from the end of a BStr.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → Source BStr.
Return:     Nothing.


Procedure:  BStrSize
File:       \ObjAsm\Code\ObjMem\32\BStrSize.asm
Purpose:    Determine the size of a BStr including the zero terminating character + leading DWORD.
Arguments:  Arg1: → Source BStr.
Return:     eax = String size including the length field and zero terminator in BYTEs.


Procedure:  BStrStartsWith
File:       \ObjAsm\Code\ObjMem\32\BStrStartsWith.asm
Purpose:    Compare the beginning of a BSTR.
Arguments:  Arg1: → Analized BSTR.
            Arg2: → Prefix BSTR.
Return:     eax = TRUE of the beginning matches, otherwise FALSE.


Procedure:  byte2hexA
File:       \ObjAsm\Code\ObjMem\32\byte2hexA.asm
Purpose:    Convert a BYTE to its hexadecimal ANSI string representation.
```

```
Arguments:  Arg1: → Destination ANSI string buffer.
            Arg2: BYTE value.
Return:     Nothing.
Notes:      The destination buffer must be at least 3 BYTEs large to allocate the output string
            (2 character BYTEs + ZTC = 3 BYTEs).


Procedure:  byte2hexW
File:       \ObjAsm\Code\ObjMem\32\byte2hexW.asm
Purpose:    Convert a BYTE to its hexadecimal WIDE string representation.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: BYTE value.
Return:     Nothing.
Notes:      The destination buffer must be at least 5 BYTEs large to allocate the output string
            (2 character WORDs + ZTC = 5 BYTEs).


Procedure:  CalcVarianceDW
File:       \ObjAsm\Code\ObjMem\32\CalcVarianceDW.asm
Purpose:    Calculate the MSE of an array of DWORDs.
Arguments:  Arg1: → Array of DWORDs.
            Arg2: DWORD Array count.
            Arg3: → Variance.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:
               https://www.mun.ca/biology/scarr/Simplified_calculation_of_variance.html#:~:text=A%
            20more%20straightforward%20calculation%20recognizes,i2%20)%20%2F%20n%20%2D%202
            https://www.mun.ca/biology/scarr/Mean_&_Variance.html#:~:text=easily%20calculated%20as
Formulas:   Var = Y2/N-(Y/N)^2  or  (Y2*N-Y^2)/N^2
            where Y: Sum(y), Y2: Sum(y^2), N:Population count = Array size.


Procedure:  CalcVarianceQW
File:       \ObjAsm\Code\ObjMem\32\CalcVarianceQW.asm
Purpose:    Calculate the MSE of an array of QWORDs.
Arguments:  Arg1: → Array of QWORDs.
            Arg2: DWORD Array count.
            Arg3: → Variance.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:
               https://www.mun.ca/biology/scarr/Simplified_calculation_of_variance.html#:~:text=A%
            20more%20straightforward%20calculation%20recognizes,i2%20)%20%2F%20n%20%2D%202
            https://www.mun.ca/biology/scarr/Mean_&_Variance.html#:~:text=easily%20calculated%20as
Formulas:   Var = Y2/N-(Y/N)^2  or  (Y2*N-Y^2)/N^2
            where Y: Sum(y), Y2: Sum(y^2), N:Population count = Array size.


Procedure:  CalcVarianceR4
File:       \ObjAsm\Code\ObjMem\32\CalcVarianceR4.asm
Purpose:    Calculate the MSE of an array of REAL4s.
Arguments:  Arg1: → Array of REAL4s.
            Arg2: DWORD Array count.
            Arg3: → Variance.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:
               https://www.mun.ca/biology/scarr/Simplified_calculation_of_variance.html#:~:text=A%
            20more%20straightforward%20calculation%20recognizes,i2%20)%20%2F%20n%20%2D%202
            https://www.mun.ca/biology/scarr/Mean_&_Variance.html#:~:text=easily%20calculated%20as
Formulas:   Var = Y2/N-(Y/N)^2  or  (Y2*N-Y^2)/N^2
            where Y: Sum(y), Y2: Sum(y^2), N:Population count = Array size.


Procedure:  CalcVarianceR8
File:       \ObjAsm\Code\ObjMem\32\CalcVarianceR8.asm
Purpose:    Calculate the MSE of an array of REAL8s.
Arguments:  Arg1: → Array of REAL8s.
            Arg2: DWORD Array count.
            Arg3: → Variance.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:
               https://www.mun.ca/biology/scarr/Simplified_calculation_of_variance.html#:~:text=A%
            20more%20straightforward%20calculation%20recognizes,i2%20)%20%2F%20n%20%2D%202
            https://www.mun.ca/biology/scarr/Mean_&_Variance.html#:~:text=easily%20calculated%20as
Formulas:   Var = Y2/N-(Y/N)^2  or  (Y2*N-Y^2)/N^2
            where Y: Sum(y), Y2: Sum(y^2), N:Population count = Array size.


Procedure:  CenterForm
File:       \ObjAsm\Code\ObjMem\32\CenterForm.asm
Purpose:    Calculate the starting coordinate of a window based on the screen and the window size.
Arguments:  Arg1: Window size in pixel.
```

```
            Arg2: Screen size in pixel.
Return:     eax = Starting point in pixel.


Procedure:  ComEventsAdvice
File:       \ObjAsm\Code\ObjMem\32\ComEventsAdvice.asm
Purpose:    Notificate the Event source that pISink will recieve Events.
Arguments:  Arg1: → Any Source Object Interface.
            Arg2: → Sink IUnknown Interface.
            Arg3: → IID of the outgoing interface whose connection point object is being
            requested (defined by the Source to communicate and implemented by the Sink).
            Arg4: → ConnectionPoint interface pointer.
            Arg5: → DWORD Cookie.
Return:     eax = HRESULT.


Procedure:  ComEventsUnadvice
File:       \ObjAsm\Code\ObjMem\32\ComEventsUnadvice.asm
Purpose:    Notificate the Event source that pISource will NOT recieve Events any more.
Arguments:  Arg1: → Previous ConnectionPoint interface.
            Arg2: DWORD Cookie received from previous ComEventsAdvice call.
Return:     eax = HRESULT.


Procedure:  ComGetErrStrA
File:       \ObjAsm\Code\ObjMem\32\ComGetErrStrA.asm
Purpose:    Return a description ANSI string from a COM error code.
Arguments:  Arg1: COM error code.
Return:     eax → Error string.


Procedure:  ComGetErrStrW
File:       \ObjAsm\Code\ObjMem\32\ComGetErrStrW.asm
Purpose:    Return a description WIDE string from a COM error code.
Arguments:  Arg1: COM error code.
Return:     eax → Error string.


Procedure:  ComPtrAssign
File:       \ObjAsm\Code\ObjMem\32\ComPtrAssign.asm
Purpose:    First increment the reference count of the new interface and then release any
            existing interface pointer.
Arguments:  Arg1: → Old Interface pointer.
            Arg2: New Interface pointer.


Procedure:  CreatePathA
File:       \ObjAsm\Code\ObjMem\32\CreatePathA.asm
Purpose:    Create a path on the destination drive.
Arguments:  Arg1: → ANSI path string.
Return:     Nothing.


Procedure:  CreatePathW
File:       \ObjAsm\Code\ObjMem\32\CreatePathW.asm
Purpose:    Create a path on the destination drive.
Arguments:  Arg1: → Wide path string.
Return:     Nothing.


Procedure:  DbgClose
File:       \ObjAsm\Code\ObjMem\32\DbgClose.asm
Purpose:    Close the connection to the output device.
Arguments:  None.
Return:     Nothing.


Procedure:  DbgConOpen
File:       \ObjAsm\Code\ObjMem\32\DbgConOpen.asm
Purpose:    Open a new console for the calling process.
Arguments:  None.
Return:     eax = TRUE if it was opened, otherwise FALSE.


Procedure:  DbgLogOpen
File:       \ObjAsm\Code\ObjMem\32\DbgLogOpen.asm
Purpose:    Open a Log-File.
Arguments:  None.
Return:     eax = TRUE if it was opened, otherwise FALSE.
```

```
Procedure:  DbgOutApiErr
File:       \ObjAsm\Code\ObjMem\32\DbgOutApiErr.asm
Purpose:    Identify a API error with a string.
Arguments:  Arg1: Api error code obtained with GetLastError.
            Arg2: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgOutBitmap
File:       \ObjAsm\Code\ObjMem\32\DbgOutBitmap.asm
Purpose:    Send a bitmap to the Debug Center Window.
Arguments:  Arg1: Bitamp HANDLE.
            Arg2: → Destination window WIDE name.
Return:     Nothing.


Procedure:  DbgOutCmd
File:       \ObjAsm\Code\ObjMem\32\DbgOutCmd.asm
Purpose:    Send a command to a specific Debug window.
Arguments:  Arg1: Command ID [BYTE].
            Arg2: Target Debug Window WIDE name.
Return:     Nothing.


Procedure:  DbgOutComErr
File:       \ObjAsm\Code\ObjMem\32\DbgOutComErr.asm
Purpose:    Identify a COM error with a string.
Arguments:  Arg1: COM error ID.
            Arg2: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgOutComponent
File:       \ObjAsm\Code\ObjMem\32\DbgOutComponent.asm
Purpose:    Identify a COM-Component.
Arguments:  Arg1: → CSLID.
            Arg2: Foreground color.
            Arg2: → Destination Window WIDE name.


Procedure:  DbgOutComponentName
File:       \ObjAsm\Code\ObjMem\32\DbgOutComponentName.asm
Purpose:    Identify a COM-Component.
Arguments:  Arg1: → CSLID.
            Arg2: Foreground color.
            Arg2: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgOutFPU
File:       \ObjAsm\Code\ObjMem\32\DbgOutFPU.asm
Purpose:    Display the content of the FPU.
Arguments:  Arg1: → Destination Window WIDE name.
            Arg2: Text RGB color.
Return:     Nothing.


Procedure:  DbgOutFPU_UEFI
File:       \ObjAsm\Code\ObjMem\32\DbgOutFPU_UEFI.asm
Purpose:    Display the content of the FPU.
Arguments:  Arg1: → Destination Window WIDE name.
            Arg2: Text RGB color.
Return:     Nothing.


Procedure:  DbgOutInterface
File:       \ObjAsm\Code\ObjMem\32\DbgOutInterface.asm
Purpose:    Identify a COM-Interface.
Arguments:  Arg1: → CSLID.
            Arg2: Foreground color.
            Arg2: → Destination Window WIDE name.


Procedure:  DbgOutInterfaceName
File:       \ObjAsm\Code\ObjMem\32\DbgOutInterfaceName.asm
Purpose:    Identify a COM-Interface.
Arguments:  Arg1: → CSLID.
            Arg2: Foreground color.
            Arg2: → Destination window WIDE name.
Return:     Nothing.
```

```
Procedure:  DbgOutMem
File:       \ObjAsm\Code\ObjMem\32\DbgOutMem.asm
Purpose:    Output the content of a memory block.
Arguments:  Arg1: → Memory block.
            Arg2: Memory block size.
            Arg3: Representation format.
            Arg4: Memory output color.
            Arg5: Representation output color.
            Arg6: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgOutMem_UEFI
File:       \ObjAsm\Code\ObjMem\32\DbgOutMem_UEFI.asm
Purpose:    Output the content of a memory block.
Arguments:  Arg1: → Memory block.
            Arg2: Memory block size.
            Arg3: Representation format.
            Arg4: Memory output color.
            Arg5: Representation output color.
            Arg6: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgOutMsg
File:       \ObjAsm\Code\ObjMem\32\DbgOutMsg.asm
Purpose:    Identifies a windows message with a string.
Arguments:  Arg1: Windows message ID.
            Arg2: Foreground color.
            Arg3: → Destination window WIDE name.
Return:     Nothing.


Procedure:  DbgOutTextA
File:       \ObjAsm\Code\ObjMem\32\DbgOutTextA.asm
Purpose:    Sends an ANSI string to the debug output device.
Arguments:  Arg1: → Zero terminated ANSI string.
            Arg2: Color value.
            Arg3: Effect value (DBG_EFFECT_XXX).
            Arg4: → Destination window WIDE name.
Return:     Nothing.


Procedure:  DbgOutTextA_UEFI
File:       \ObjAsm\Code\ObjMem\32\DbgOutTextA_UEFI.asm
Purpose:    Sends an ANSI string to the debug output device.
Arguments:  Arg1: → Zero terminated ANSI string.
            Arg2: Color value.
            Arg3: Effect value (DBG_EFFECT_XXX).
            Arg4: → Destination window WIDE name.
Return:     Nothing.


Procedure:  DbgOutTextCA
File:       \ObjAsm\Code\ObjMem\32\DbgOutTextCA.asm
Purpose:    Send a counted ANSI string to the debug output device
Arguments:  Arg1: → Null terminated ANSI string.
            Arg2: Character count.
            Arg3: Color value.
            Arg4: Effect value (DBG_EFFECT_XXX).
            Arg5: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgOutTextCW
File:       \ObjAsm\Code\ObjMem\32\DbgOutTextCW.asm
Purpose:    Send a counted WIDE string to the debug output device
Arguments:  Arg1: → Null terminated WIDE string.
            Arg2: Character count.
            Arg3: Color value.
            Arg4: Effect value (DBG_EFFECT_XXX).
            Arg5: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgOutTextW
File:       \ObjAsm\Code\ObjMem\32\DbgOutTextW.asm
Purpose:    Send a WIDE string to the debug output device.
Arguments:  Arg1: → Zero terminated WIDE string.
            Arg2: Color value.
```

```
            Arg3: Effect value (DBG_EFFECT_XXX)
            Arg4: → Destination window WIDE name.
Return:     Nothing.


Procedure:  DbgOutTextW_UEFI
File:       \ObjAsm\Code\ObjMem\32\DbgOutTextW_UEFI.asm
Purpose:    Send a WIDE string to the debug output device.
Arguments:  Arg1: → Zero terminated WIDE string.
            Arg2: Color value.
            Arg3: Effect value (DBG_EFFECT_XXX).
            Arg4: → Destination window WIDE name.
Return:     Nothing.


Procedure:  DbgShowObjectHeader
File:       \ObjAsm\Code\ObjMem\32\DbgShowObjectHeader.asm
Purpose:    Output heading object information.
Arguments:  Arg1: → Object Name.
            Arg2: → Instance.
            Arg3: Text RGB color.
            Arg3: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgShowObjectHeader_UEFI
File:       \ObjAsm\Code\ObjMem\32\DbgShowObjectHeader_UEFI.asm
Purpose:    Output heading object information.
Arguments:  Arg1: → Object Name.
            Arg2: → Instance.
            Arg3: Text RGB color.
            Arg3: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgShowTraceMethod
File:       \ObjAsm\Code\ObjMem\32\DbgShowTraceMethod.asm
Purpose:    Output trace information about a method.
Arguments:  Arg1: → Method Name.
            Arg2: Method count.
            Arg3: Method ticks.
            Arg4: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgWndOpen
File:       \ObjAsm\Code\ObjMem\32\DbgWndOpen.asm
Purpose:    Open Debug Center instance.
Arguments:  None.
Return:     eax = TRUE if it was opened, otherwise FALSE.


Procedure:  dec2dwordA
File:       \ObjAsm\Code\ObjMem\32\dec2dwordA.asm
Purpose:    Convert a decimal ANSI string to a DWORD.
Arguments:  Arg1: → Source ANSI string. Possible leading characters are " ", tab, "+" and "-",
            followed by a sequence of chars between "0".."9" and finalized by a ZTC.
            Other characters terminate the convertion returning zero.
Return:     eax = Converted DWORD.
            ecx = Conversion result. Zero if succeeded, otherwise failed.


Procedure:  dec2dwordW
File:       \ObjAsm\Code\ObjMem\32\dec2dwordW.asm
Purpose:    Convert a decimal WIDE string to a DWORD.
Arguments:  Arg1: → Source WIDE string. Possible leading characters are " ", tab, "+" and "-",
            followed by a sequence of chars between "0".."9" and finalized by a ZTC.
            Other characters terminate the convertion returning zero.
Return:     eax = Converted DWORD.
            ecx = Conversion result. Zero if succeeded, otherwise failed.


Procedure:  DisableCPUSerialNumber
File:       \ObjAsm\Code\ObjMem\32\DisableCPUSerialNumber.asm
Purpose:    Disable the reading of the CPU serial number.
Arguments:  None.
Return:     Nothing.


Procedure:  DllErr2StrA
File:       \ObjAsm\Code\ObjMem\32\DllErr2StrA.asm
```

```
Purpose:    Translate an error code to an ANSI string stored in a DLL.
Arguments:  Arg1: Error code.
            Arg2: → ANSI character buffer.
            Arg3: Buffer size in characters, inclusive ZTC.
            Arg4: → DLL ANSI name.
Return:     Nothing.


Procedure:  DllErr2StrW
File:       \ObjAsm\Code\ObjMem\32\DllErr2StrW.asm
Purpose:    Translate an error code to a WIDE string stored in a DLL.
Arguments:  Arg1: Error code.
            Arg2: → WIDE character buffer.
            Arg3: Buffer size in characters, inclusive ZTC.
            Arg4: → DLL WIDE name.
Return:     Nothing.


Procedure:  DrawTransparentBitmap
File:       \ObjAsm\Code\ObjMem\32\DrawTransparentBitmap.asm
Purpose:    Draw a bitmap with transparency on a device context.
Arguments:  Arg1: DC handle.
            Arg2: Bitmap handle to draw.
            Arg3; X start position on DC.
            Arg4: Y start position on DC.
            Arg5: RGB transparent color. Use TBM_FIRSTPIXEL to indicate that the pixel in the
            upper left corner contains the transparent color.
Return:     Nothing.
Note:       Original source by Microsoft.
            "HOWTO: Drawing Transparent Bitmaps (Q79212)"
            (http://support.microsoft.com/default.aspx?scid=kb;EN-US;q79212)
            Transcribed by Ernest Murphy.


Procedure:  dword2binA
File:       \ObjAsm\Code\ObjMem\32\dword2binA.asm
Purpose:    Convert a DWORD to its binary ANSI string representation.
Arguments:  Arg1: → Destination buffer.
            Arg2: DWORD value.
Return:     Nothing.
Note:       The destination buffer must be at least 33 BYTEs large to allocate the output string
            (32 character BYTEs + ZTC = 33 BYTEs).


Procedure:  dword2binW
File:       \ObjAsm\Code\ObjMem\32\dword2binW.asm
Purpose:    Convert a DWORD to its binary WIDE string representation.
Arguments:  Arg1: → Destination buffer.
            Arg2: DWORD value.
Return:     Nothing.
Note:       The destination buffer must be at least 66 BYTEs large to allocate the output string
            (32 character WORDs + ZTC = 66 BYTEs).


Procedure:  dword2hexA
File:       \ObjAsm\Code\ObjMem\32\dword2hexA.asm
Purpose:    Convert a DWORD to its hexadecimal ANSI string representation.
Arguments:  Arg1: → Destination buffer.
            Arg2: DWORD value.
Return:     Nothing.
Note:       The destination buffer must be at least 9 BYTEs large to allocate the output string
            (8 character BYTEs + ZTC = 9 BYTEs).


Procedure:  dword2hexW
File:       \ObjAsm\Code\ObjMem\32\dword2hexW.asm
Purpose:    Convert a DWORD to its hexadecimal WIDE string representation.
Arguments:  Arg1: → Destination buffer.
            Arg2: DWORD value.
Return:     Nothing.
Note:       The destination buffer must be at least 18 BYTEs large to allocate the output string
            (8 character WORDs + ZTC = 18 BYTEs).


Procedure:  Err2StrA
File:       \ObjAsm\Code\ObjMem\32\Err2StrA.asm
Purpose:    Translate a system error code to an ANSI string.
Arguments:  Arg1: Error code.
            Arg2: → ANSI string buffer.
            Arg3: Buffer size in characters, inclusive ZTC.
Return:     Nothing.
```

```
Procedure:  Err2StrW
File:       \ObjAsm\Code\ObjMem\32\Err2StrW.asm
Purpose:    Translate a system error code to a WIDE string.
Arguments:  Arg1: Error code.
            Arg2: → WIDE string buffer.
            Arg3: Buffer size in characters, inclusive ZTC.
Return:     Nothing.


Procedure:  ErrorMessageBoxA
File:       \ObjAsm\Code\ObjMem\32\ErrorMessageBoxA.asm
Purpose:    Show a Messagebox containing an error string in the locale language and an user str.
Arguments:  Arg1: Messagebox parent window HANDLE.
            Arg2: → User ANSI string.
            Arg3: Locale ID.
            Arg4: API error code.
Return:     Nothing.


Procedure:  ErrorMessageBoxW
File:       \ObjAsm\Code\ObjMem\32\ErrorMessageBoxW.asm
Purpose:    Show a messagebox containing an error string in the locale lenguage and an user str.
Arguments:  Arg1: Messagebox parent window HANDLE.
            Arg2: → User WIDE string.
            Arg3: Locale ID.
            Arg4: API error code.
Return:     Nothing.


Procedure:  FileExistA
File:       \ObjAsm\Code\ObjMem\32\FileExistA.asm
Purpose:    Check the existence of a file.
Arguments:  Arg1: → ANSI file name.
Return:     eax = TRUE if the file exists, otherwise FALSE.


Procedure:  FileExistW
File:       \ObjAsm\Code\ObjMem\32\FileExistW.asm
Purpose:    Check the existence of a file.
Arguments:  Arg1: → WIDE file name.
Return:     eax = TRUE if the file exists, otherwise FALSE.


Procedure:  FindFileA
File:       \ObjAsm\Code\ObjMem\32\FindFileA.asm
Purpose:    Search for a file in a list of paths.
Arguments:  Arg1: → File name.
            Arg2: → List of path strings. The end of the list is indicated with a ZTC.
            Arg3: → Buffer containing the full path and file name in which the file was found.
            Buffer length = MAX_PATH.
Return:     eax = Number of chars copied to the destination buffer. 0 if the file was not found.
            Example: invoke FindFile, $OfsCStr("free.inc"), $OfsCStr("\Here*",0), addr cBuf
            Search free.inc in all \Here and suddirectories.


Procedure:  FindFileW
File:       \ObjAsm\Code\ObjMem\32\FindFileW.asm
Purpose:    Search for a file in a list of paths.
Arguments:  Arg1: → File name.
            Arg2: → List of path strings. The end of the list is indicated with a ZTC.
            Arg3: → Buffer containing the full path and file name in which the file was found.
            Buffer length = MAX_PATH.
Return:     eax = Number of chars copied to the destination buffer. 0 if the file was not found.
            Example: invoke FindFile, $OfsCStr("free.inc"), $OfsCStr("\Here*",0), addr cBuf
            Search free.inc in all \Here and suddirectories.


Procedure:  FindModuleByAddrA
File:       \ObjAsm\Code\ObjMem\32\FindModuleByAddrA.asm
Purpose:    Find the module name from an address on a WinNT system.
Arguments:  Arg1: Address.
            Arg2: → ANSI module name buffer.
Return:     eax = Number of characters copied into the buffer.


Procedure:  FindModuleByAddrW
File:       \ObjAsm\Code\ObjMem\32\FindModuleByAddrW.asm
Purpose:    Find the module name from an address on a WinNT system.
Arguments:  Arg1: Address.
```

```
            Arg2: → WIDE module name buffer.
Return:     eax = Number of characters copied into the buffer.


Procedure:  GetAncestorID
File:       \ObjAsm\Code\ObjMem\32\GetAncestorID.asm
Purpose:    Retrieve the ancestor type ID of an object type ID.
Arguments:  Arg1: → Object class ID.
Return:     eax = Ancestor type ID or zero if not found.


Procedure:  GetBottomWindow
File:       \ObjAsm\Code\ObjMem\32\GetBottomWindow.asm
Purpose:    Get the Z order bottom child window HANDLE.
Arguments:  Arg1: Parenat HWND.
Return:     eax = Z order bottom child window HANDLE.


Procedure:  GetDlgBaseUnits
File:       \ObjAsm\Code\ObjMem\32\GetDlgBaseUnits.asm
Purpose:    Returns the Dialog Base Units.
Arguments:  Arg1: Dialog DC.
Return:     eax = X DBU.
            ecx = Y DBU.


Procedure:  GetExceptionStrA
File:       \ObjAsm\Code\ObjMem\32\GetExceptionStrA.asm
Purpose:    Translate an exception code to an ANSI string.
Arguments:  Arg1: Exception code.
Return:     eax → ANSI string.


Procedure:  GetExceptionStrW
File:       \ObjAsm\Code\ObjMem\32\GetExceptionStrW.asm
Purpose:    Translate an exception code to a WIDE string.
Arguments:  Arg1: Exception code.
Return:     eax → WIDE string.


Procedure:  GetFileHashA
File:       \ObjAsm\Code\ObjMem\32\GetFileHashA.asm
Purpose:    Compute the hash value from the content of a file.
Arguments:  Arg1: → Hash return value
            Arg2: → ANSI file name.
            Arg3: Hash type.
Return:     eax = 0 if succeeded.
Links:      http://www.masm32.com/board/index.php?topic=4322.msg32297#msg32297
Notes:      Original translation from MSDN library by Edgar Hansen
            It requires a fully qualified path to a file to generate a hash for and a pointer
            to a WIDE string buffer to hold the resulting hash in HEX (16 BYTEs for MDx, 20 BYTEs
            for SHAx) and an algorithm ID, for MD5 set dHashType to GFH_MD5.
            See ObjMem.inc GFH_xxx.


Procedure:  GetFileHashW
File:       \ObjAsm\Code\ObjMem\32\GetFileHashW.asm
Purpose:    Compute the hash value from the content of a file.
Arguments:  Arg1: → Hash return value
            Arg2: → WIDE file name.
            Arg3: Hash type.
Return:     eax = 0 if succeeded.
Links:      http://www.masm32.com/board/index.php?topic=4322.msg32297#msg32297
Notes:      Original translation from MSDN library by Edgar Hansen
            It requires a fully qualified path to a file to generate a hash for and a pointer
            to a WIDE string buffer to hold the resulting hash in HEX (16 BYTEs for MDx, 20 BYTEs
            for SHAx) and an algorithm ID, for MD5 set dHashType to GFH_MD5.
            See ObjMem.inc GFH_xxx.


Procedure:  GetFileLinesA
File:       \ObjAsm\Code\ObjMem\32\GetFileLinesA.asm
Purpose:    Return an array of line ending offsets of an ANSI text file.
Arguments:  Arg1: File HANDLE.
Return:     eax = Number of lines.
            ecx → Mem block containing an array of DWORD offsets.
            The user must dispose it using MemFree.

Notes:      - Lines must be terminated with the ANSI char sequence 13, 10 (CRLF).
            - The last line may not terminate with a CRLF.
```

```
Procedure:  GetLogProcCount
File:       \ObjAsm\Code\ObjMem\32\GetLogProcCount.asm
Purpose:    Return the number of logical CPUs on the current system.
Arguments:  None
Return:     eax = Number of logical processors.


Procedure:  GetObjectID
File:       \ObjAsm\Code\ObjMem\32\GetObjectID.asm
Purpose:    Retrieve the type ID of an object instance.
Arguments:  Arg1: → Object instance.
Return:     eax = Object class ID.


Procedure:  GetObjectTemplate
File:       \ObjAsm\Code\ObjMem\32\GetObjectTemplate.asm
Purpose:    Get the template address of an object type ID.
Arguments:  Arg1: Object type ID.
Return:     eax → Object template or NULL if not found.
            ecx = Object template size or zero if not found.


Procedure:  GetPrevInstanceA
File:       \ObjAsm\Code\ObjMem\32\GetPrevInstanceA.asm
Purpose:    Return a HANDLE to a previously running instance of an application.
Arguments:  Arg1: → ANSI application name.
            Arg2: → ANSI class name.
Return:     eax = Window HANDLE of the application instance or zero if failed.


Procedure:  GetPrevInstanceW
File:       \ObjAsm\Code\ObjMem\32\GetPrevInstanceW.asm
Purpose:    Return a HANDLE to a previously running instance of an application.
Arguments:  Arg1: → WIDE application name.
            Arg2: → WIDE class name.
Return:     eax = Window HANDLE of the application instance or zero if failed.


Procedure:  GetRawClientRect
File:       \ObjAsm\Code\ObjMem\32\GetRawClientRect.asm
Purpose:    Calculate the window client RECT including scrollbars, but without the room needed
            for the menubar.
Arguments:  Arg1: Window HANDLE
            Arg2: → RECT.
Return:     Nothing.


Procedure:  GUID2BStr
File:       \ObjAsm\Code\ObjMem\32\GUID2BStr.asm
Purpose:    Convert a GUID to a BStr.
Arguments:  Arg1: → Destination BStr Buffer. It must hold at least
            36 characters plus a ZTC.
            Arg2: → GUID.
Return:     Nothing.


Procedure:  GUID2StrA
File:       \ObjAsm\Code\ObjMem\32\GUID2StrA.asm
Purpose:    Convert a GUID to an ANSI string.
Arguments:  Arg1: → Destination ANSI string buffer.
            It must hold at least 36 characters plus a ZTC (= 37 BYTEs).
            Arg2: → GUID.
Return:     Nothing.


Procedure:  GUID2StrW
File:       \ObjAsm\Code\ObjMem\32\GUID2StrW.asm
Purpose:    Convert a GUID to a WIDE string.
Arguments:  Arg1: → Destination WIDE string Buffer.
            It must hold at least 36 characters plus a ZTC (= 74 BYTEs).
            Arg2: → GUID.
Return:     Nothing.


Procedure:  hex2dwordA
File:       \ObjAsm\Code\ObjMem\32\hex2dwordA.asm
Purpose:    Load an ANSI string hexadecimal representation of a DWORD.
Arguments:  Arg1: → ANSI hexadecimal string.
Return:     eax = DWORD.
```

```
Procedure:  hex2dwordW
File:       \ObjAsm\Code\ObjMem\32\hex2dwordW.asm
Purpose:    Load a WIDE string hexadecimal representation of a DWORD.
Arguments:  Arg1: → WIDE hex string.
Return:     eax = DWORD.


Procedure:  IsAdmin
File:       \ObjAsm\Code\ObjMem\32\IsAdmin.asm
Purpose:    Check if the current user has administrator rights.
Arguments:  None.
Return:     eax = TRUE or FALSE.


Procedure:  IsGUIDEqual
File:       \ObjAsm\Code\ObjMem\32\IsGUIDEqual.asm
Purpose:    Compare 2 GUIDs.
Arguments:  Arg1: → GUID1
            Arg2: → GUID2.
Return:     eax = TRUE if they are equal, otherwise FALSE.


Procedure:  IsHardwareFeaturePresent
File:       \ObjAsm\Code\ObjMem\32\IsHardwareFeaturePresent.asm
Purpose:    Check if a CPU hardware feature is present on the system.
Notes:      Check IHFP_xxx equates in ObjMem.inc file.
Arguments:  Arg1: CPUID feature ID.
Return:     eax = TRUE or FALSE.


Procedure:  IsPntInRect
File:       \ObjAsm\Code\ObjMem\32\IsPntInRect.asm
Purpose:    Check if a point is within a rect.
            If rect.left = rect.right = 0, the point.x is considdered inside. Idem for y coord.
Arguments:  Arg1: → POINT.
            ARg2: → RECT
Return:     eax = TRUE or FALSE.


Procedure:  IsProcessElevated
File:       \ObjAsm\Code\ObjMem\32\IsProcessElevated.asm
Purpose:    Check if the current process has elevated privileges.
Arguments:  Arg: Process HANDLE.
Return:     eax = TRUE or FALSE.
            Example: invoke GetCurrentProcess
            invoke IsProcessElevated, xax


Procedure:  IsScrollBarVisible
File:       \ObjAsm\Code\ObjMem\32\IsScrollBarVisible.asm
Purpose:    Determine if a Scrollbar is currently visible.
Arguments:  Arg1: Main window handle that the scrollbar belongs to.
            Arg2: Scrollbar type [SB_HORZ or SB_VERT].
Return:     eax = TRUE if the scrollbar is visible, otherwise FALSE.


Procedure:  IsWinNT
File:       \ObjAsm\Code\ObjMem\32\IsWinNT.asm
Purpose:    Detect if the OS is Windows NT based.
Arguments:  None.
Return:     eax = TRUE if OS is Windows NT based, otherwise FALSE.


Procedure:  LoadCommonControls
File:       \ObjAsm\Code\ObjMem\32\LoadCommonControls.asm
Purpose:    Invoke InitCommonControls with a correctly filled input structure.
Arguments:  Arg1: ICC_COOL_CLASSES, ICC_BAR_CLASSES, ICC_LISTVIEW_CLASSES, ICC_TAB_CLASSES,
            ICC_USEREX_CLASSES, etc.
Return:     Nothing.


Procedure:  Mem2HexA
File:       \ObjAsm\Code\ObjMem\32\Mem2HexA.asm
Purpose:    Convert the memory content into a hex ANSI string representation.
Arguments:  Arg1: → ANSI character buffer.
            Arg2: → Source memory.
            Arg3: Byte count.
Return:     Nothing.
```

```
Procedure:  Mem2HexW
File:       \ObjAsm\Code\ObjMem\32\Mem2HexW.asm
Purpose:    Convert the memory content into a hex WIDE string representation.
Arguments:  Arg1: → WIDE character buffer.
            Arg2: → Source memory.
            Arg3: Byte count.
Return:     Nothing.


Procedure:  MemAlloc_UEFI
File:       \ObjAsm\Code\ObjMem\32\MemAlloc_UEFI.asm
Purpose:    Allocate a memory block.
Arguments:  Arg1: Memory block attributes [0, MEM_INIT_ZERO].
            Arg2: Memory block size in BYTEs.
Return:     eax → Memory block or NULL if failed.


Procedure:  MemClone
File:       \ObjAsm\Code\ObjMem\32\MemClone.asm
Purpose:    Copy a memory block from a source to a destination buffer.
            Source and destination must NOT overlap.
            Destination buffer must be at least as large as number of BYTEs to copy, otherwise a
            fault may be triggered.
Arguments:  Arg1: → Destination buffer.
            Arg2: → Source buffer.
            Arg3: Number of BYTEs to copy.
Return:     Nothing.


Procedure:  MemComp
File:       \ObjAsm\Code\ObjMem\32\MemComp.asm
Purpose:    Compare 2 memory blocks.
            Both memory blocks must be at least as large as the maximal number of BYTEs to
            compare, otherwise a fault may be triggered.
Arguments:  Arg1: → Memory block 1.
            Arg2: → Memory block 2.
            Arg3: Maximal number of BYTEs to compare.
Return:     If MemBlock1 = MemBlock2, then eax <> 0.
            If MemBlock1 == MemBlock2, then eax = 0.


Procedure:  MemFillB
File:       \ObjAsm\Code\ObjMem\32\MemFillB.asm
Purpose:    Fill a memory block with a given byte value.
            Destination buffer must be at least as large as number of BYTEs to fill, otherwise a
            fault may be triggered.
Arguments:  Arg1: → Destination memory block.
            Arg2: Memory block size in BYTEs.
            Arg3: Byte value to fill.
Return:     Nothing.


Procedure:  MemFillW
File:       \ObjAsm\Code\ObjMem\32\MemFillW.asm
Purpose:    Fill a memory block with a given word value.
            Destination buffer must be at least as large as number of BYTEs to fill, otherwise a
            fault may be triggered.
Arguments:  Arg1: → Destination memory block.
            Arg2: Memory block size in BYTEs.
            Arg3: Word value to fill with.
Return:     Nothing.


Procedure:  MemFree_UEFI
File:       \ObjAsm\Code\ObjMem\32\MemFree_UEFI.asm
Purpose:    Dispose a memory block.
Arguments:  Arg1: → Memory block.
Return:     eax = EFI_SUCCESS or an UEFI error code.


Procedure:  MemReAlloc_UEFI
File:       \ObjAsm\Code\ObjMem\32\MemReAlloc_UEFI.asm
Purpose:    Shrink or expand a memory block.
Arguments:  Arg1: → Memory block
            Arg2: Memory block size in BYTEs.
            Arg3: New memory block size in BYTEs.
            Arg4: Memory block attributes [0, MEM_INIT_ZERO].
Return:     eax → New memory block.


Procedure:  MemShift
```

```
File:       \ObjAsm\Code\ObjMem\32\MemShift.asm
Purpose:    Copy a memory block from a source to a destination buffer.
            Source and destination may overlap.
            Destination buffer must be at least as large as number of BYTEs to shift, otherwise a
            fault may be triggered.
Arguments:  Arg1: → Destination buffer.
            Arg2: → Source buffer.
            Arg3: Number of BYTEs to shift.
Return:     eax = Number of BYTEs shifted.


Procedure:  MemSwap
File:       \ObjAsm\Code\ObjMem\32\MemSwap.asm
Purpose:    Exchange the memory content from a memory buffer to another.
            They must NOT overlap.
            Both buffers must be at least as large as number of BYTEs to exchange, otherwise a
            fault may be triggered.
Arguments:  Arg1: → Memory buffer 1.
            Arg2: → Memory buffer 2.
            Arg3: Number of BYTEs to exchange.
Return:     Nothing.


Procedure:  MemZero
File:       \ObjAsm\Code\ObjMem\32\MemZero.asm
Purpose:    Fill a memory block with zeros. A bit faster than MemFillB.
            The memory buffer must be at least as large as number of BYTEs to zero, otherwise a
            fault may be triggered.
Arguments:  Arg1: → Memory buffer.
            Arg2: Number of BYTEs to zero.
Return:     Nothing.


Procedure:  MoveWindowVisible
File:       \ObjAsm\Code\ObjMem\32\MoveWindowVisible.asm
Purpose:    On a multimonitor system, move a window but remain always in the visible region.
Arguments:  Arg1: HANDLE of the Window to move.
            Arg2: Target X position in pixel.
            Arg3: Target Y position in pixel.
Return:     Nothing.


Procedure:  MsgBoxA
File:       \ObjAsm\Code\ObjMem\32\MsgBoxA.asm
Purpose:    Show a customized MessageBox.
Arguments:  Arg1: Parent HANDLE.
            Arg2: → Markup text.
            Arg3: → Caption text.
            Arg4: Flags.
Return:     eax = Zero if failed, otherwise pressed button ID.
Note:       Caption, text etc. are transferred via a caption string which contains a header and
            the address of a MsgBoxInfo structure in text form.


Procedure:  MsgBoxW
File:       \ObjAsm\Code\ObjMem\32\MsgBoxW.asm
Purpose:    Show a customized MessageBox.
Arguments:  Arg1: Parent HANDLE.
            Arg2: → Markup text.
            Arg3: → Caption text.
            Arg4: Flags.
Return:     eax = Zero if failed, otherwise pressed button ID.
Note:       Caption, text etc. are transferred via a caption string which contains a header and
            the address of a MsgBoxInfo structure in text form.


Procedure:  NetErr2StrA
File:       \ObjAsm\Code\ObjMem\32\NetErr2StrA.asm
Purpose:    Translate a network error code to an ANSI string.
Arguments:  Arg1: Error code.
            Arg2: → ANSI character buffer.
            Arg3: Buffer size in characters, inclusive ZTC.
Return:     Nothing.


Procedure:  NetErr2StrW
File:       \ObjAsm\Code\ObjMem\32\NetErr2StrW.asm
Purpose:    Translate a network error code to a WIDE string.
Arguments:  Arg1: Error code.
            Arg2: → WIDE string buffer.
            Arg3: Buffer size in characters, inclusive ZTC.
```

```
Return:     Nothing.


Procedure:  NewObjInst
File:       \ObjAsm\Code\ObjMem\32\NewObjInst.asm
Purpose:    Create an object instance from an object ID.
Arguments:  Arg1: Object ID.
Return:     eax → New object instance or NULL if failed.


Procedure:  NewObjInst_UEFI
File:       \ObjAsm\Code\ObjMem\32\NewObjInst_UEFI.asm
Purpose:    Create an object instance from an object ID.
Arguments:  Arg1: Object ID.
Return:     eax → New object instance or NULL if failed.


Procedure:  ParseA
File:       \ObjAsm\Code\ObjMem\32\ParseA.asm
Purpose:    Extract a comma separated substring from a source string.
Arguments:  Arg1: → Destination buffer. Must be large enough to hold the ANSI substring.
            Arg2: → Source ANSI string.
            Arg3: Zero based index of the requested ANSI substring.
Return:     eax = 1: success.
            2: insufficient number of components.
            3: non matching quotation marks.
            4: empty quote.


Procedure:  ParseW
File:       \ObjAsm\Code\ObjMem\32\ParseW.asm
Purpose:    Extract a comma separated substring from a source string.
Arguments:  Arg1: → Destination buffer. Must be large enough to hold the WIDE substring.
            Arg2: → Source WIDE string.
            Arg3: Zero based index of the requested WIDE substring.
Return:     eax = 1: success.
            2: insufficient number of components.
            3: non matching quotation marks.
            4: empty quote.


Procedure:  PdfViewA
File:       \ObjAsm\Code\ObjMem\32\PdfViewA.asm
Purpose:    Display a PDF document on a named destination.
Arguments:  Arg1: Parent HANDLE.
            Arg2: → PDF document.
            Arg3: → Destination.
Return:     eax = HINSTANCE. See ShellExecute return values.
            A value greater than 32 indicates success.


Procedure:  PdfViewW
File:       \ObjAsm\Code\ObjMem\32\PdfViewW.asm
Purpose:    Display a PDF document on a named destination.
Arguments:  Arg1: Parent HANDLE.
            Arg2: → PDF document.
            Arg3: → Destination.
Return:     eax = HINSTANCE. See ShellExecute return values.
            A value greater than 32 indicates success.


Procedure:  qword2binA
File:       \ObjAsm\Code\ObjMem\32\qword2binA.asm
Purpose:    Convert a QWORD to its binary ANSI string representation.
Arguments:  Arg1: → Destination buffer.
            Arg2: QWORD value.
Return:     Nothing.
Note:       The destination buffer must be at least 65 BYTEs large to allocate the output string
            (64 character BYTEs + ZTC = 65 BYTEs).


Procedure:  qword2binW
File:       \ObjAsm\Code\ObjMem\32\qword2binW.asm
Purpose:    Convert a QWORD to its binary WIDE string representation.
Arguments:  Arg1: → Destination buffer.
            Arg2: QWORD value.
Return:     Nothing.
Note:       The destination buffer must be at least 130 BYTEs large to allocate the output string
            (64 character WORDs + ZTC = 130 BYTEs).
```

```
Procedure:  qword2hexA
File:       \ObjAsm\Code\ObjMem\32\qword2hexA.asm
Purpose:    Convert a QWORD to its hexadecimal ANSI string representation.
Arguments:  Arg1: → Destination buffer.
            Arg2: QWORD value.
Return:     Nothing.
Note:       The destination buffer must be at least 17 BYTEs large to allocate the output string
            (16 character BYTEs + ZTC = 17 BYTEs).


Procedure:  qword2hexW
File:       \ObjAsm\Code\ObjMem\32\qword2hexW.asm
Purpose:    Convert a QWORD to its hexadecimal WIDE string representation.
Arguments:  Arg1: → Destination buffer.
            Arg2: QWORD value.
Return:     Nothing.
Note:       The destination buffer must be at least 34 BYTEs large to allocate the output string
            (16 character WORDs + ZTC = 34 BYTEs).


Procedure:  RadixSortF32
File:       \ObjAsm\Code\ObjMem\32\RadixSortF32.asm
Purpose:    Ascending sort of an array of single precision floats (REAL4) using a modified
            "4 passes radix sort" algorithm.
Arguments:  Arg1: → Array of single precision floats.
            Arg2: Number of single precision floats contained in the array.
            Arg3: → Memory used for the sorting process or NULL. The buffer size must be at least
            the size of the input array. If NULL, a memory chunk is allocated automatically.
Return:     eax = TRUE if succeeded, otherwise FALSE.
Notes:      - For short arrays, the shadow array can be placed onto the stack, saving the
            expensive memory allocation/deallocation API calls. To achieve this, the proc
            must be modified and stack probing must be included.
Links:      - http://www.codercorner.com/RadixSortRevisited.htm
            - http://en.wikipedia.org/wiki/Radix_sort


Procedure:  RadixSortF64
File:       \ObjAsm\Code\ObjMem\32\RadixSortF64.asm
Purpose:    Ascending sort of an array of double precision floats (REAL8) using a modified
            "8 passes radix sort" algorithm.
Arguments:  Arg1: → Array of double precision floats (REAL8).
            Arg2: Number of double precision floats contained in the array.
            Arg3: → Memory used for the sorting process or NULL. The buffer size must be at least
            the size of the input array. If NULL, a memory chunk is allocated automatically.
Return:     eax = TRUE if succeeded, otherwise FALSE.
Notes:      - For short arrays, the shadow array can be placed onto the stack, saving the
            expensive memory allocation/deallocation API calls. To achieve this, the proc
            must be modified and stack probing must be included.
Links:      - http://www.codercorner.com/RadixSortRevisited.htm
            - http://en.wikipedia.org/wiki/Radix_sort


Procedure:  RadixSortI32
File:       \ObjAsm\Code\ObjMem\32\RadixSortI32.asm
Purpose:    Ascending sort of an array of SDWORDs using a modified "4 passes radix sort"
            algorithm.
Arguments:  Arg1: → Array of SDWORDs.
            Arg2: Number of SDWORDs contained in the array.
            Arg3: → Memory used for the sorting process or NULL. The buffer size must be at least
            the size of the input array. If NULL, a memory chunk is allocated automatically.
Return:     eax = TRUE if succeeded, otherwise FALSE.
Notes:      - Original code from r22.
            http://www.asmcommunity.net/board/index.php?topic=24563.0
            - For short arrays, the shadow array can be placed onto the stack, saving the
            expensive memory allocation/deallocation API calls. To achieve this, the proc
            must be modified and stack probing must be included.
Links:      - http://www.codercorner.com/RadixSortRevisited.htm
            - http://en.wikipedia.org/wiki/Radix_sort


Procedure:  RadixSortPtrF32
File:       \ObjAsm\Code\ObjMem\32\RadixSortPtrF32.asm
Purpose:    Ascending sort of an array of POINTERs to structures containing a
            single precision float (REAL4) key using a modified "4 passes radix sort" algorithm.
Arguments:  Arg1: → Array of POINTERs.
            Arg2: Number of POINTERs contained in the array.
            Arg3: offset of the REAL4 key within the hosting structure.
            Arg4: → Memory used for the sorting process or NULL. The buffer size must be at least
            the size of the input array. If NULL, a memory chunk is allocated automatically.
Return:     eax = TRUE if succeeded, otherwise FALSE.
Notes:      - For short arrays, the shadow array can be placed onto the stack, saving the
```

```
            expensive memory allocation/deallocation API calls. To achieve this, the proc
            must be modified and stack probing must be included.
Links:      - http://www.codercorner.com/RadixSortRevisited.htm
            - http://en.wikipedia.org/wiki/Radix_sort


Procedure:  RadixSortPtrF64
File:       \ObjAsm\Code\ObjMem\32\RadixSortPtrF64.asm
Purpose:    Ascending sort of an array of POINTERs to structures containing a
            double precision float (REAL8) key using a modified "8 passes radix sort" algorithm.
Arguments:  Arg1: → Array of POINTERs.
            Arg2: Number of POINTERs contained in the array.
            Arg3: offset of the REAL8 key within the hosting structure.
            Arg4: → Memory used for the sorting process or NULL. The buffer size must be at least
            the size of the input array. If NULL, a memory chunk is allocated automatically.
Return:     eax = TRUE if succeeded, otherwise FALSE.
Notes:      - For short arrays, the shadow array can be placed onto the stack, saving the
            expensive memory allocation/deallocation API calls. To achieve this, the proc
            must be modified and stack probing must be included.
Links:      - http://www.codercorner.com/RadixSortRevisited.htm
            - http://en.wikipedia.org/wiki/Radix_sort


Procedure:  RadixSortPtrI32
File:       \ObjAsm\Code\ObjMem\32\RadixSortPtrI32.asm
Purpose:    Ascending sort of an array of POINTERs to structures containing a SDWORD key using a
            modified "4 passes radix sort" algorithm.
Arguments:  Arg1: → Array of POINTERs.
            Arg2: Number of POINTERs contained in the array.
            Arg3: offset of the SDWORD key within the hosting structure.
            Arg4: → Memory used for the sorting process or NULL. The buffer size must be at least
            the size of the input array. If NULL, a memory chunk is allocated automatically.
Return:     eax = TRUE if succeeded, otherwise FALSE.
Notes:      - Original code from r22.
            http://www.asmcommunity.net/board/index.php?topic=24563.0
            - For short arrays, the shadow array can be placed onto the stack, saving the
            expensive memory allocation/deallocation API calls. To achieve this, the proc
            must be modified and stack probing must be included.
Links:      - http://www.codercorner.com/RadixSortRevisited.htm
            - http://en.wikipedia.org/wiki/Radix_sort


Procedure:  RadixSortPtrUI32
File:       \ObjAsm\Code\ObjMem\32\RadixSortPtrUI32.asm
Purpose:    Ascending sort of a POINTER array to structures containing a DWORD key using the
            "4 passes radix sort" algorithm.
Arguments:  Arg1: → Array of POINTERs.
            Arg2: Number of POINTERs contained in the array.
            Arg3: offset of the DWORD key within the hosting structure.
            Arg4: → Memory used for the sorting process or NULL. The buffer size must be at least
            the size of the input array. If NULL, a memory chunk is allocated automatically.
Return:     eax = TRUE if succeeded, otherwise FALSE.
Notes:      - Original code from r22.
            http://www.asmcommunity.net/board/index.php?topic=24563.0
            - For short arrays, the shadow array can be placed onto the stack, saving the
            expensive memory allocation/deallocation API calls. To achieve this, the proc
            must be modified and stack probing must be included.
Links:      - http://www.codercorner.com/RadixSortRevisited.htm
            - http://en.wikipedia.org/wiki/Radix_sort


Procedure:  RadixSortUI32
File:       \ObjAsm\Code\ObjMem\32\RadixSortUI32.asm
Purpose:    Ascending sort of an array of DWORDs using the "4 passes radix sort" algorithm.
Arguments:  Arg1: → Array of DWORDs.
            Arg2: Number of DWORDs contained in the array.
            Arg3: → Memory used for the sorting process or NULL. The buffer size must be at least
            the size of the input array. If NULL, a memory chunk is allocated automatically.
Return:     eax = TRUE if succeeded, otherwise FALSE.
Notes:      - Original code from r22.
            http://www.asmcommunity.net/board/index.php?topic=24563.0
            - For short arrays, the shadow array can be placed onto the stack, saving the
            expensive memory allocation/deallocation API calls. To achieve this, the proc
            must be modified and stack probing must be included.
Links:      - http://www.codercorner.com/RadixSortRevisited.htm
            - http://en.wikipedia.org/wiki/Radix_sort


Procedure:  Random32
File:       \ObjAsm\Code\ObjMem\32\Random32.asm
Purpose:    Generate a random 32 bit number in a given range [0..Limit-1].
```

```
            Park Miller random number algorithm. Written by Jaymeson Trudgen (NaN) and optimized
            by Rickey Bowers Jr. (bitRAKE).
Arguments:  Arg1: Range limit (max. = 07FFFFFFFh).
Return:     eax = Random number in the range [0..Limit-1].


Procedure:  Real4ToHalf
File:       \ObjAsm\Code\ObjMem\32\Real4ToHalf.asm
Purpose:    Convert a REAL4 to an HALF.
Arguments:  Arg1: REAL4 value.
Return:     ax = HALF.
Note:       alternative code using VCVTPS2PH:
            movss xmm0, r4Value
            VCVTPS2PH xmm1, xmm0, 0
            movd eax, xmm1


Procedure:  RGB24To16ColorIndex
File:       \ObjAsm\Code\ObjMem\32\RGB24To16ColorIndex.asm
Purpose:    Map a 24 bit RGB color to a 16 color palette index.
Arguments:  Arg1: RGB color.
Return:     eax = Palette index.


Procedure:  sdword2decA
File:       \ObjAsm\Code\ObjMem\32\sdword2decA.asm
Purpose:    Convert a signed DWORD to its decimal ANSI string representation.
Arguments:  Arg1: → Destination ANSI string buffer.
            Arg2: SDWORD value.
Return:     eax = Number of BYTEs copied to the destination buffer, including the ZTC.
Note:       The destination buffer must be at least 12 BYTEs large to allocate the output string
            (Sign + 10 ANSI characters + ZTC = 12 BYTEs).


Procedure:  sdword2decW
File:       \ObjAsm\Code\ObjMem\32\sdword2decW.asm
Purpose:    Convert a signed DWORD to its decimal WIDE string representation.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: SDWORD value.
Return:     eax = Number of BYTEs copied to the destination buffer, including the ZTC.
Note:       The destination buffer must be at least 24 BYTEs large to allocate the output string
            (Sign + 10 WIDE characters + ZTC = 24 BYTEs).


Procedure:  SendChildrenMessage
File:       \ObjAsm\Code\ObjMem\32\SendChildrenMessage.asm
Purpose:    Callback procedure for EnumChildWindows that sends a message to a child window.
Arguments:  Arg1: Child window HANDLE.
            Arg2: → CHILD_MSG structure.
Return:     eax = Always TRUE (continue the enumeration).


Procedure:  SetClientSize
File:       \ObjAsm\Code\ObjMem\32\SetClientSize.asm
Purpose:    Set the client window size.
Arguments:  Arg1: Target window handle.
            Arg2: Client area width in pixel.
            Arg3: Client area height in pixel.
Return:     Nothing.


Procedure:  SetExceptionMessageA
File:       \ObjAsm\Code\ObjMem\32\SetExceptionMessageA.asm
Purpose:    Install a final exception handler that displays a messagebox showing detailed
            exception
            information and a user text.
Arguments:  Arg1: → User ANSI message string.
            Arg2: → Messagebox ANSI title string.
            Arg3: → Callback procedure fired when an exception reaches the final handler.
            If the callback returns zero, the messagebox is displayed, otherwise
            EXCEPTION_EXECUTE_HANDLER is passed to the OS without showing the messagebox.
            If this parameter is NULL, the messgebox is always displayed.
Return:     Nothing.


Procedure:  SetExceptionMessageW
File:       \ObjAsm\Code\ObjMem\32\SetExceptionMessageW.asm
Purpose:    Install a final exception handler that displays a messagebox showing detailed
            exception
            information and a user text.
Arguments:  Arg1: → User wide message string.
```

```
            Arg2: → Messagebox WIDE title string.
            Arg3: → Callback procedure fired when an exception reaches the final handler.
            If the callback returns zero, the messagebox is displayed, otherwise
            EXCEPTION_EXECUTE_HANDLER is passed to the OS without showing the messagebox.
            If this parameter is NULL, the messgebox is always displayed.
Return:     Nothing.


Procedure:  SetPrivilegeTokenA
File:       \ObjAsm\Code\ObjMem\32\SetPrivilegeTokenA.asm
Purpose:    Enable privilege tokens.
Arguments:  Arg1: Process handle.
            Arg2: → Privilege name (ANSI string).
            Arg3: Eanble = TRUE, disable = FALSE
Return:     eax = Zero if failed.


Procedure:  SetPrivilegeTokenW
File:       \ObjAsm\Code\ObjMem\32\SetPrivilegeTokenW.asm
Purpose:    Enable privilege tokens.
Arguments:  Arg1: Process handle.
            Arg2: → Privilege name (ANSI string).
            Arg3: Eanble = TRUE, disable = FALSE
Return:     eax = Zero if failed.


Procedure:  SetShellAssociationA
File:       \ObjAsm\Code\ObjMem\32\SetShellAssociationA.asm
Purpose:    Set association for a file extension.
Arguments:  Arg1: TRUE = system wide association, FALSE = user account only.
            Arg2: → File extension (without dot).
            Arg3: → Verb ("open", "print", "play", "edit", etc.). This verb is displayed
            in the explorer context menu of a file with this extension.
            Arg4: → Application to associate with (full path).
            Arg5: → Application arguments, usually $OfsCStr("%1").
Return:     eax = HRESULT.
Note:       dGlobal = TRUE requires adminitrative rights.


Procedure:  SetShellAssociationW
File:       \ObjAsm\Code\ObjMem\32\SetShellAssociationW.asm
Purpose:    Set association for a file extension.
Arguments:  Arg1: TRUE = system wide association, FALSE = user account only.
            Arg2: → File extension (without dot).
            Arg3: → Verb ("open", "print", "play", "edit", etc.). This verb is displayed
            in the explorer context menu of a file with this extension.
            Arg4: → Application to associate with (full path).
            Arg5: → Application arguments, usually $OfsCStr("%1").
Return:     eax = HRESULT.
Note:       dGlobal = TRUE requires adminitrative rights.


Procedure:  SetShellPerceivedTypeA
File:       \ObjAsm\Code\ObjMem\32\SetShellPerceivedTypeA.asm
Purpose:    Set shell perception of a file type.
Arguments:  Arg1: TRUE = system wide perseption, FALSE = user account only.
            Arg2: → File extension (without dot).
            Arg3: → Type (Folder, Text, Image, Audio, Video, Compressed, Document, System,
            Application, Gamemedia, Contacts)
Return:     eax = HRESULT.
Note:       To retrieve the perceived type use the AssocGetPerceivedType API.
            dGlobal = TRUE requires adminitrative rights.


Procedure:  SetShellPerceivedTypeW
File:       \ObjAsm\Code\ObjMem\32\SetShellPerceivedTypeW.asm
Purpose:    Set shell perception of a file type.
Arguments:  Arg1: TRUE = system wide perseption, FALSE = user account only.
            Arg2: → File extension (without dot).
            Arg3: → Type (Folder, Text, Image, Audio, Video, Compressed, Document, System,
            Application, Gamemedia, Contacts)
Return:     eax = HRESULT.
Note:       To retrieve the perceived type use the AssocGetPerceivedType API.
            dGlobal = TRUE requires adminitrative rights.


Procedure:  ShortToLongPathNameA
File:       \ObjAsm\Code\ObjMem\32\ShortToLongPathNameA.asm
Purpose:    Allocate a new ANSI string containing the long path of a short path string.
Arguments:  Arg1: → Short path ANSI string.
Return:     eax → Long path ANSI string or NULL if failed.
```

```
Procedure:  SLR_Calc_AB_DW
File:       \ObjAsm\Code\ObjMem\32\SLR_Calc_AB_DW.asm
Purpose:    Calculate the Slope (A) and Intercept (B) values of the linear equation y = A*x + B
            that minimize mean squared error (MSE) of a DWORD array.
Arguments:  Arg1: → SLR_DATA structure.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
            If an FPU exception occurs, the results are NaN.
Formulas:   A = (XY*N-X*Y)/Q
            B = (Y-A*X)/N


Procedure:  SLR_Calc_AB_MSE_DW
File:       \ObjAsm\Code\ObjMem\32\SLR_Calc_AB_MSE_DW.asm
Purpose:    Calculate the Slope (A) and Intercept (B) values of the linear equation y = A*x + B
            that minimize mean squared error (MSE) and the MSE value of a DWORD array.
Arguments:  Arg1: → SLR_DATA structure.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
            If an FPU exception occurs, the results are NaN.
Formulas:   A = (XY*N-X*Y)/Q
            B = (Y-A*X)/N
            MSE = (Y2 - 2*A*XY - 2*B*Y + A^2*X2 + 2*A*B*X)/N + B^2


Procedure:  SLR_Calc_AB_MSE_QW
File:       \ObjAsm\Code\ObjMem\32\SLR_Calc_AB_MSE_QW.asm
Purpose:    Calculate the Slope (A) and Intercept (B) values of the linear equation y = A*x + B
            that minimize mean squared error (MSE) and the MSE value of a QWORD array.
Arguments:  Arg1: → SLR_DATA structure.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
            If an FPU exception occurs, the results are NaN.
Formulas:   A = (XY*N-X*Y)/Q
            B = (Y-A*X)/N
            MSE = (Y2 - 2*A*XY - 2*B*Y + A^2*X2 + 2*A*B*X)/N + B^2


Procedure:  SLR_Calc_AB_MSE_R4
File:       \ObjAsm\Code\ObjMem\32\SLR_Calc_AB_MSE_R4.asm
Purpose:    Calculate the Slope (A) and Intercept (B) values of the linear equation y = A*x + B
            that minimize mean squared error (MSE) and the MSE value of a REAL4 array.
Arguments:  Arg1: → SLR_DATA structure.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
            If an FPU exception occurs, the results are NaN.
Formulas:   A = (XY*N-X*Y)/Q
            B = (Y-A*X)/N
            MSE = (Y2 - 2*A*XY - 2*B*Y + A^2*X2 + 2*A*B*X)/N + B^2


Procedure:  SLR_Calc_AB_MSE_R8
File:       \ObjAsm\Code\ObjMem\32\SLR_Calc_AB_MSE_R8.asm
Purpose:    Calculate the Slope (A) and Intercept (B) values of the linear equation y = A*x + B
            that minimize mean squared error (MSE) and the MSE value of a REAL8 array.
Arguments:  Arg1: → SLR_DATA structure.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
```

```
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
            If an FPU exception occurs, the results are NaN.
Formulas:   A = (XY*N-X*Y)/Q
            B = (Y-A*X)/N
            MSE = (Y2 - 2*A*XY - 2*B*Y + A^2*X2 + 2*A*B*X)/N + B^2


Procedure:  SLR_Calc_AB_QW
File:       \ObjAsm\Code\ObjMem\32\SLR_Calc_AB_QW.asm
Purpose:    Calculate the Slope (A) and Intercept (B) values of the linear equation y = A*x + B
            that minimize mean squared error (MSE) of a QWORD array.
Arguments:  Arg1: → SLR_DATA structure.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
            If an FPU exception occurs, the results are NaN.
Formulas:   A = (XY*N-X*Y)/Q
            B = (Y-A*X)/N


Procedure:  SLR_Calc_AB_R4
File:       \ObjAsm\Code\ObjMem\32\SLR_Calc_AB_R4.asm
Purpose:    Calculate the Slope (A) and Intercept (B) values of the linear equation y = A*x + B
            that minimize mean squared error (MSE) of a REAL4 array.
Arguments:  Arg1: → SLR_DATA structure.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
            If an FPU exception occurs, the results are NaN.
Formulas:   A = (XY*N-X*Y)/Q
            B = (Y-A*X)/N


Procedure:  SLR_Calc_AB_R8
File:       \ObjAsm\Code\ObjMem\32\SLR_Calc_AB_R8.asm
Purpose:    Calculate the Slope (A) and Intercept (B) values of the linear equation y = A*x + B
            that minimize mean squared error (MSE) of a REAL8 array.
Arguments:  Arg1: → SLR_DATA structure.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
            If an FPU exception occurs, the results are NaN.
Formulas:   A = (XY*N-X*Y)/Q
            B = (Y-A*X)/N


Procedure:  SLR_Init
File:       \ObjAsm\Code\ObjMem\32\SLR_Init.asm
Purpose:    Calculate in advance the invariant coefficients of a Simple Linear Regression (X, X2,
            Q)
Arguments:  Arg1: → SLR_DATA structure.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
Formulas:   X = N*(N-1)/2
            X2 = X*(2*N - 1)/3
            Q = N^2*(N^2-1)/12


Procedure:  sqword2decA
File:       \ObjAsm\Code\ObjMem\32\sqword2decA.asm
Purpose:    Convert a signed QWORD to its decimal ANSI string representation.
```

```
Arguments:  Arg1: → Destination ANSI string buffer.
            Arg2: SQWORD value.
Return:     eax = Number of BYTEs copied to the destination buffer, including the ZTC.
Note:       The destination buffer must be at least 21 BYTEs large to allocate the output string
            (Sign + 19 ANSI characters + ZTC = 21 BYTEs).


Procedure:  sqword2decW
File:       \ObjAsm\Code\ObjMem\32\sqword2decW.asm
Purpose:    Convert a signed SQWORD to its decimal WIDE string representation.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: SQWORD value.
Return:     eax = Number of BYTEs copied to the destination buffer, including the ZTC.
Note:       The destination buffer must be at least 42 BYTEs large to allocate the output string
            (Sign + 19 WIDE characters + ZTC = 42 BYTEs).


Procedure:  St0ToStrA
File:       \ObjAsm\Code\ObjMem\32\St0ToStrA.asm
Purpose:    Create an ANSI string representation of the content of the st(0) FPU register.
Arguments:  Arg1: → Destination buffer.
            Arg2: Minimal number of places from the start of string up to the decimal point.
            (f_NOR only)
            Arg3: Number of decimal places after the decimal point.
            Arg4: Format flag (f_NOR, f_SCI, f_TRIM, f_ALIGNED) defined in fMath.inc
Return:     eax = Result code f_OK, f_ERROR, f_NAN, ...
Notes:      - Based on the work of Raymond Filiatreault (FpuLib).
            - st4, st5, st6 and st7 must be empty.
            - f_NOR: regular output format
            - f_SCI: Scientific output format
            - f_TRIM: Trim zeros on the right
            - f_ALIGN: Add a heading space to align the output with other negative numbers
            - f_PLUS: like f_ALIGN, but using a + character.


Procedure:  St0ToStrW
File:       \ObjAsm\Code\ObjMem\32\St0ToStrW.asm
Purpose:    Create a WIDE string representation of the content of the st(0) FPU register.
Arguments:  Arg1: → Destination buffer.
            Arg2: Minimal number of places from the start of string up to the decimal point.
            (f_NOR only)
            Arg3: Number of decimal places after the decimal point.
            Arg4: Format flag (f_NOR, f_SCI, f_TRIM, f_ALIGNED) defined in fMath.inc
Return:     eax = Result code f_OK, f_ERROR, f_NAN, ...
Notes:      - Based on the work of Raymond Filiatreault (FpuLib).
            - st4, st5, st6 and st7 must be empty.
            - f_NOR: regular output format
            - f_SCI: Scientific output format
            - f_TRIM: Trim zeros on the right
            - f_ALIGN: Add a heading space to align the output with other negative numbers
            - f_PLUS: like f_ALIGN, but using a + character.


Procedure:  StkGrdCallback
File:       \ObjAsm\Code\ObjMem\32\StkGrdCallback.asm
Purpose:    StackGuard notification callback procedure.
            It is called when StackGuard is active and a stack overrun was detected.
            It displays a MessageBox asking to abort. If yes, then Exitprocess is called
            immediately.
Arguments:  None.
            Returns: ZERO flag set if NO was pressed


Procedure:  Str2BStrA
File:       \ObjAsm\Code\ObjMem\32\Str2BStrA.asm
Purpose:    Convert a ANSI string into a BStr.
Arguments:  Arg1: → Destination BStr buffer = Buffer address + sizeof DWORD.
            Arg2: → Source ANSI string.
Return:     Nothing.


Procedure:  Str2BStrW
File:       \ObjAsm\Code\ObjMem\32\Str2BStrW.asm
Purpose:    Convert a ANSI string into a BStr.
Arguments:  Arg1: → Destination BStr buffer = Buffer address + sizeof DWORD.
            Arg2: → Source WIDE string.
Return:     Nothing.


Procedure:  StrA2StrW
File:       \ObjAsm\Code\ObjMem\32\StrA2StrW.asm
```

```
Purpose:    Convert a ANSI string into a WIDE string.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: → Source ANSI string.
Return:     Nothing.


Procedure:  StrAllocA
File:       \ObjAsm\Code\ObjMem\32\StrAllocA.asm
Purpose:    Allocate space for an ANSI string with n characters.
Arguments:  Arg1: Character count without the ZTC.
Return:     eax → New allocated ANSI string or NULL if failed.


Procedure:  StrAllocA_UEFI
File:       \ObjAsm\Code\ObjMem\32\StrAllocA_UEFI.asm
Purpose:    Allocate space for a string with n characters.
Arguments:  Arg1: Character count without the ZTC.
Return:     eax → New allocated string or NULL if failed.


Procedure:  StrAllocW
File:       \ObjAsm\Code\ObjMem\32\StrAllocW.asm
Purpose:    Allocate space for a WIDE string with n characters.
Arguments:  Arg1: Character count without the ZTC.
Return:     eax → New allocated WIDE string or NULL if failed.


Procedure:  StrAllocW_UEFI
File:       \ObjAsm\Code\ObjMem\32\StrAllocW_UEFI.asm
Purpose:    Allocate space for a string with n characters.
Arguments:  Arg1: Character count without the ZTC.
Return:     eax → New allocated string or NULL if failed.


Procedure:  StrCatA
File:       \ObjAsm\Code\ObjMem\32\StrCatA.asm
Purpose:    Concatenate 2 ANSI strings.
Arguments:  Arg1: Destination ANSI buffer.
            Arg2: Source ANSI string.
Return:     eax = Number of added BYTEs.


Procedure:  StrCatCharA
File:       \ObjAsm\Code\ObjMem\32\StrCatCharA.asm
Purpose:    Append a character to the end of an ANSI string.
Arguments:  Arg1: Destination ANSI buffer.
            Arg2: ANSI character.
Return:     Nothing.


Procedure:  StrCatCharW
File:       \ObjAsm\Code\ObjMem\32\StrCatCharW.asm
Purpose:    Append a character to the end of a WIDE string.
Arguments:  Arg1: Destination WIDE buffer.
            Arg2: WIDE character.
Return:     Nothing.


Procedure:  StrCatW
File:       \ObjAsm\Code\ObjMem\32\StrCatW.asm
Purpose:    Concatenate 2 WIDE strings.
Arguments:  Arg1: Destination WIDE string.
            Arg2: Source WIDE string.
Return:     Nothing.


Procedure:  StrCCatA
File:       \ObjAsm\Code\ObjMem\32\StrCCatA.asm
Purpose:    Concatenate 2 ANSI strings with length limitation.
            The destination string buffer should have at least enough room for the maximum number
            of characters + 1.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
            Arg3: Maximal number of charachters that the destination string can hold including the
            ZTC.
Return:     eax = Number of added BYTEs.


Procedure:  StrCCatCharA
File:       \ObjAsm\Code\ObjMem\32\StrCCatCharA.asm
Purpose:    Append a character to the end of an ANSI string with length limitation.
```

```
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → ANSI character.
            Arg3: Maximal number of characters that fit into the destination buffer.
Return:     Nothing.


Procedure:  StrCCatCharW
File:       \ObjAsm\Code\ObjMem\32\StrCCatCharW.asm
Purpose:    Append a character to the end of a WIDE string with length limitation.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Wide character.
            Arg3: Maximal number of characters that fit into the destination buffer.
Return:     Nothing.


Procedure:  StrCCatW
File:       \ObjAsm\Code\ObjMem\32\StrCCatW.asm
Purpose:    Concatenate 2 WIDE strings with length limitation.
            The destination string buffer should have at least enough room for the maximum number
            of characters + 1.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
            Arg3: Maximal number of charachters that the destination string can hold including the
            ZTC.
Return:     eax = Number of added BYTEs.


Procedure:  StrCCompA
File:       \ObjAsm\Code\ObjMem\32\StrCCompA.asm
Purpose:    Compare 2 ANSI strings with case sensitivity up to a maximal number of characters.
Arguments:  Arg1: → ANSI string 1.
            Arg2: → ANSI string 2.
            Arg3: Maximal number of characters to compare.
Return:     If string 1 < string 2, then eax < 0.
            If string 1 = string 2, then eax = 0.
            If string 1 > string 2, then eax > 0.


Procedure:  StrCCompW
File:       \ObjAsm\Code\ObjMem\32\StrCCompW.asm
Purpose:    Compare 2 WIDE strings with case sensitivity up to a maximal number of characters.
Arguments:  Arg1: → WIDE string 1.
            Arg2: → WIDE string 2.
            Arg3: Maximal number of characters to compare.
Return:     If string 1 < string 2, then eax < 0.
            If string 1 = string 2, then eax = 0.
            If string 1 > string 2, then eax > 0.


Procedure:  StrCCopyA
File:       \ObjAsm\Code\ObjMem\32\StrCCopyA.asm
Purpose:    Copy the the source ANSI string with length limitation.
            The destination buffer should be big enough to hold the maximum number of
            characters + 1.
Arguments:  Arg1: → Destination buffer.
            Arg2: → Source ANSI string.
            Arg3: Maximal number of charachters to copy, excluding the ZTC.
Return:     eax = Number of copied BYTEs, including the ZTC.


Procedure:  StrCCopyW
File:       \ObjAsm\Code\ObjMem\32\StrCCopyW.asm
Purpose:    Copy the the source WIDE string with length limitation.
            The destination buffer should be big enough to hold the maximum number of
            characters + 1.
Arguments:  Arg1: → Destination buffer.
            Arg2: → Source WIDE string.
            Arg3: Maximal number of charachters to copy, excluding the ZTC.
Return:     eax = Number of copied BYTEs, including the ZTC.


Procedure:  StrCECatA
File:       \ObjAsm\Code\ObjMem\32\StrCECatA.asm
Purpose:    Concatenate 2 ANSI strings with length limitation and return the ending zero character
            address. The destination string buffer should have at least enough room for the
            maximum
            number of characters + 1.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
            Arg3: Maximal number of charachters that the destination string can hold including the
            ZTC.
```

```
Return:     eax → ZTC.


Procedure:  StrCECatW
File:       \ObjAsm\Code\ObjMem\32\StrCECatW.asm
Purpose:    Concatenate 2 WIDE strings with length limitation and return the ending zero character
            address. The destination string buffer should have at least enough room for the
            maximum number of characters + 1.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
            Arg3: Maximal number of charachters that the destination string can hold including the
            ZTC.
Return:     eax → ZTC.


Procedure:  StrCECopyA
File:       \ObjAsm\Code\ObjMem\32\StrCECopyA.asm
Purpose:    Copy the the source ANSI string with length limitation and return the ending zero
            character address.
            The destination buffer should hold the maximum number of characters + 1.
            Source and destination strings may overlap.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
            Arg3: Maximal number of characters not including the ZTC.
Return:     eax → ZTC.


Procedure:  StrCECopyW
File:       \ObjAsm\Code\ObjMem\32\StrCECopyW.asm
Purpose:    Copy the the source WIDE string with length limitation and return the last zero
            character address.
            The destination buffer should hold the maximum number of characters + 1.
            Source and destination strings may overlap.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
            Arg3: Maximal number of characters not including the ZTC.
Return:     eax → ZTC.


Procedure:  StrCICompA
File:       \ObjAsm\Code\ObjMem\32\StrCICompA.asm
Purpose:    Compare 2 ANSI strings without case sensitivity and length limitation.
Arguments:  Arg1: → ANSI string 1.
            Arg2: → ANSI string 2.
Return:     If string 1 < string 2, then eax < 0.
            If string 1 = string 2, then eax = 0.
            If string 1 > string 2, then eax > 0.


Procedure:  StrCICompW
File:       \ObjAsm\Code\ObjMem\32\StrCICompW.asm
Purpose:    Compare 2 WIDE strings without case sensitivity and length limitation.
Arguments:  Arg1: → Wide string 1.
            Arg2: → Wide string 2.
Return:     If string 1 < string 2, then eax < 0.
            If string 1 = string 2, then eax = 0.
            If string 1 > string 2, then eax > 0.


Procedure:  StrCLengthA
File:       \ObjAsm\Code\ObjMem\32\StrCLengthA.asm
Purpose:    Get the character count of the source ANSI string with length limitation.
Arguments:  Arg1: → Source ANSI string.
            Arg3: Maximal character count.
Return:     eax = Limited character count.


Procedure:  StrCLengthW
File:       \ObjAsm\Code\ObjMem\32\StrCLengthW.asm
Purpose:    Get the character count of the source WIDE string with length limitation.
Arguments:  Arg1: → Source WIDE string.
            Arg3: Maximal character count.
Return:     eax = Limited character count.


Procedure:  StrCNewA
File:       \ObjAsm\Code\ObjMem\32\StrCNewA.asm
Purpose:    Allocate a new copy of the source ANSI string with length limitation.
            If the pointer to the source string is NULL or points to an empty string, StrCNewA
            returns NULL and doesn't allocate any heap space. Otherwise, StrCNewA makes a
            duplicate of the source string. The maximal size of the new string is limited to the
```

```
            second parameter.
Arguments:  Arg1: → Source ANSI string.
            Arg2: Maximal character count.
Return:     eax → New ANSI string copy.


Procedure:  StrCNewA_UEFI
File:       \ObjAsm\Code\ObjMem\32\StrCNewA_UEFI.asm
Purpose:    Allocate a new copy of the source ANSI string with length limitation.
            If the pointer to the source string is NULL or points to an empty string, StrCNewA
            returns NULL and doesn't allocate any heap space. Otherwise, StrCNewA makes a
            duplicate of the source string. The maximal size of the new string is limited to the
            second parameter.
Arguments:  Arg1: → Source ANSI string.
            Arg2: Maximal character count.
Return:     eax → New ANSI string copy.


Procedure:  StrCNewW
File:       \ObjAsm\Code\ObjMem\32\StrCNewW.asm
Purpose:    Allocate a new copy of the source WIDE string with length limitation.
            If the pointer to the source string is NULL or points to an empty string, StrCNewW
            returns NULL and doesn't allocate any heap space. Otherwise, StrCNewW makes a
            duplicate of the source string. The maximal size of the new string is limited to the
            second parameter.
Arguments:  Arg1: → Source WIDE string.
            Arg2: Maximal character count.
Return:     eax → New WIDE string copy.


Procedure:  StrCNewW_UEFI
File:       \ObjAsm\Code\ObjMem\32\StrCNewW_UEFI.asm
Purpose:    Allocate a new copy of the source WIDE string with length limitation.
            If the pointer to the source string is NULL or points to an empty string, StrCNewW
            returns NULL and doesn't allocate any heap space. Otherwise, StrCNewW makes a
            duplicate of the source string. The maximal size of the new string is limited to the
            second parameter.
Arguments:  Arg1: → Source WIDE string.
            Arg2: Maximal character count.
Return:     eax → New WIDE string copy.


Procedure:  StrCompA
File:       \ObjAsm\Code\ObjMem\32\StrCompA.asm
Purpose:    Compare 2 ANSI strings with case sensitivity.
Arguments:  Arg1: → ANSI string 1.
            Arg2: → ANSI string 2.
Return:     If string 1 < string 2, then eax < 0.
            If string 1 = string 2, then eax = 0.
            If string 1 > string 2, then eax > 0.


Procedure:  StrCompW
File:       \ObjAsm\Code\ObjMem\32\StrCompW.asm
Purpose:    Compare 2 WIDE strings with case sensitivity.
Arguments:  Arg1: → WIDE string 1.
            Arg2: → WIDE string 2.
Return:     If string 1 < string 2, then eax < 0.
            If string 1 = string 2, then eax = 0.
            If string 1 > string 2, then eax > 0.


Procedure:  StrCopyA
File:       \ObjAsm\Code\ObjMem\32\StrCopyA.asm
Purpose:    Copy an ANSI string to a destination buffer.
Arguments:  Arg1: Destination ANSI string buffer.
            Arg2: Source ANSI string.
Return:     eax = Number of BYTEs copied, including the ZTC.


Procedure:  StrCopyW
File:       \ObjAsm\Code\ObjMem\32\StrCopyW.asm
Purpose:    Copy a WIDE string to a destination buffer.
Arguments:  Arg1: Destination WIDE string buffer.
Return:     eax = Number of BYTEs copied, including the ZTC.
Return:     Nothing.


Procedure:  StrCPosA
File:       \ObjAsm\Code\ObjMem\32\StrCPosA.asm
Purpose:    Scan for ANSI string2 into ANSI string1 with length limitation.
```

```
Arguments:  Arg1: → Source ANSI string.
            Arg2: → ANSI string to search for.
            Arg3: Maximal character count.
Return:     eax → String position or NULL if not found.


Procedure:  StrCPosW
File:       \ObjAsm\Code\ObjMem\32\StrCPosW.asm
Purpose:    Scan from the beginning of a WIDE string for a character.
Arguments:  Arg1: → Source WIDE string.
            Arg2: Character to search for.
Return:     eax → Character position or NULL if not found.


Procedure:  StrCScanA
File:       \ObjAsm\Code\ObjMem\32\StrCScanA.asm
Purpose:    Scan from the beginning of ANSI string for a character with length limitation.
Arguments:  Arg1: → Source ANSI string.
            Arg2: Maximal character count.
            Arg3: ANSI character to search for.
Return:     eax → Character address or NULL if not found.


Procedure:  StrCScanW
File:       \ObjAsm\Code\ObjMem\32\StrCScanW.asm
Purpose:    Scan from the beginning of a WIDE string for a character with length limitation.
Arguments:  Arg1: → Source WIDE string.
            Arg2: Maximal character count.
            Arg3: Wide character to search for.
Return:     eax → Character address or NULL if not found.


Procedure:  StrDispose
File:       \ObjAsm\Code\ObjMem\32\StrDispose.asm
Purpose:    Free the memory allocated for the string using StrNew, StrCNew, StrLENew or
            StrAlloc.
            If the pointer to the string is NULL, StrDispose does nothing.
Arguments:  Arg1: → String.
Return:     Nothing.


Procedure:  StrDispose_UEFI
File:       \ObjAsm\Code\ObjMem\32\StrDispose_UEFI.asm
Purpose:    Free the memory allocated for the string using StrNew_UEFI, StrCNew_UEFI,
            StrLENew_UEFI or StrAlloc_UEFI.
            If the pointer to the string is NULL, StrDispose_UEFI does nothing.
Arguments:  Arg1: → String.
Return:     Nothing.


Procedure:  StrECatA
File:       \ObjAsm\Code\ObjMem\32\StrECatA.asm
Purpose:    Append an ANSI string to another and return the address of the ending zero character.
            StrCatA does not perform any length checking. The destination buffer must have room
            for at least StrLengthA(Destination) + StrLengthA(Source) + 1 characters.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
Return:     eax → ZTC.


Procedure:  StrECatCharA
File:       \ObjAsm\Code\ObjMem\32\StrECatCharA.asm
Purpose:    Append a character to an ANSI string and return the address of the ending zero.
            StrECatCharA does not perform any length checking. The destination buffer must have
            enough room for at least StrLengthA(Destination) + 1 + 1 characters.
Arguments:  Arg1: → Destination ANSI string buffer.
            Arg2: → ANSI character.
Return:     eax → ZTC.


Procedure:  StrECatCharW
File:       \ObjAsm\Code\ObjMem\32\StrECatCharW.asm
Purpose:    Append a character to a WIDE string and return the address of the ending zero.
            StrECatCharW does not perform any length checking. The destination buffer must have
            enough room for at least StrLengthW(Destination) + 1 + 1 characters.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: → Wide character.
Return:     eax → ZTC.


Procedure:  StrECatW
```

```
File:       \ObjAsm\Code\ObjMem\32\StrECatW.asm
Purpose:    Append a WIDE string to another and return the address of the ending zero character.
            StrCatW does not perform any length checking. The destination buffer must have room
            for at least StrLengthW(Destination) + StrLengthW(Source) + 1 characters.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
Return:     eax → ZTC.


Procedure:  StrECopyA
File:       \ObjAsm\Code\ObjMem\32\StrECopyA.asm
Purpose:    Copy an ANSI to a buffer and return the address of the ending zero character.
            Source and destination strings may overlap.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
Return:     eax → ZTC.


Procedure:  StrECopyW
File:       \ObjAsm\Code\ObjMem\32\StrECopyW.asm
Purpose:    Copy a WIDE to a buffer and return the address of the ZTC.
            Source and destination strings may overlap.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
Return:     eax → ZTC.


Procedure:  StrEndA
File:       \ObjAsm\Code\ObjMem\32\StrEndA.asm
Purpose:    Get the address of the zero character that terminates the string.
Arguments:  Arg1: → Source ANSI string.
Return:     eax → ZTC.


Procedure:  StrEndsWithA
File:       \ObjAsm\Code\ObjMem\32\StrEndsWithA.asm
Purpose:    Compare the ending of a string.
Arguments:  Arg1: → Analized string.
            Arg2: → Suffix string.
Return:     eax = TRUE of the ending matches, otherwise FALSE.


Procedure:  StrEndsWithW
File:       \ObjAsm\Code\ObjMem\32\StrEndsWithW.asm
Purpose:    Compare the ending of a string.
Arguments:  Arg1: → Analized string.
            Arg2: → Suffix string.
Return:     eax = TRUE of the ending matches, otherwise FALSE.


Procedure:  StrEndW
File:       \ObjAsm\Code\ObjMem\32\StrEndW.asm
Purpose:    Get the address of the zero character that terminates the string.
Arguments:  Arg1: → Source WIDE string.
Return:     eax → ZTC.


Procedure:  StrFillChrA
File:       \ObjAsm\Code\ObjMem\32\StrFillChrA.asm
Purpose:    Fill a preallocated String with a character.
Arguments:  Arg1: → String.
            Arg2: Character.
            Arg3: Character Count.
Return:     Nothing.


Procedure:  StrFillChrW
File:       \ObjAsm\Code\ObjMem\32\StrFillChrW.asm
Purpose:    Fill a preallocated String with a character.
Arguments:  Arg1: → String.
            Arg2: Character.
            Arg3: Character Count.
Return:     Nothing.


Procedure:  StrFilterA
File:       \ObjAsm\Code\ObjMem\32\StrFilterA.asm
Purpose:    Perform a case sensitive string match test using wildcards (* and ?).
Arguments:  Arg1: → Source ANSI string.
            Arg2: → Filter ANSI string.
Return:     eax = TRUE if strings match, otherwise FALSE.
```

```
Procedure:  StrFilterW
File:       \ObjAsm\Code\ObjMem\32\StrFilterW.asm
Purpose:    Perform a case sensitive string match test using wildcards (* and ?).
Arguments:  Arg1: → Source WIDE string.
            Arg2: → Filter WIDE string.
Return:     eax = TRUE if strings match, otherwise FALSE.


Procedure:  StrICompA
File:       \ObjAsm\Code\ObjMem\32\StrICompA.asm
Purpose:    Compare 2 ANSI strings without case sensitivity.
Arguments:  Arg1: → ANSI string 1.
            Arg2: → ANSI string 2.
Return:     If string 1 < string 2, then eax < 0.
            If string 1 = string 2, then eax = 0.
            If string 1 > string 2, then eax > 0.


Procedure:  StrICompW
File:       \ObjAsm\Code\ObjMem\32\StrICompW.asm
Purpose:    Compare 2 WIDE strings without case sensitivity.
Arguments:  Arg1: → Wide string 1.
            Arg2: → Wide string 2.
Return:     If string 1 < string 2, then eax < 0.
            If string 1 = string 2, then eax = 0.
            If string 1 > string 2, then eax > 0.


Procedure:  StrIFilterA
File:       \ObjAsm\Code\ObjMem\32\StrIFilterA.asm
Purpose:    Perform a case insensitive string match test using wildcards (* and ?).
Arguments:  Arg1: → Source ANSI string.
            Arg2: → Filter ANSI string.
Return:     eax = TRUE if strings match, otherwise FALSE.


Procedure:  StrIFilterW
File:       \ObjAsm\Code\ObjMem\32\StrIFilterW.asm
Purpose:    Perform a case insensitive string match test using wildcards (* and ?).
Arguments:  Arg1: → Source WIDE string.
            Arg2: → Filter WIDE string.
Return:     eax = TRUE if strings match, otherwise FALSE.


Procedure:  StrLeftA
File:       \ObjAsm\Code\ObjMem\32\StrLeftA.asm
Purpose:    Extract the left n characters of the source ANSI string.
Arguments:  Arg1: → Destination character buffer.
            Arg2: → Source ANSI string.
Return:     eax = Number of copied characters, not including the ZTC.


Procedure:  StrLeftW
File:       \ObjAsm\Code\ObjMem\32\StrLeftW.asm
Purpose:    Extract the left n characters of the source WIDE string.
Arguments:  Arg1: → Destination buffer.
            Arg2: → Source WIDE string.
Return:     eax = Number of copied characters, not including the ZTC.


Procedure:  StrLengthA
File:       \ObjAsm\Code\ObjMem\32\StrLengthA.asm
Purpose:    Determine the length of an ANSI string not including the zero terminating character.
Arguments:  Arg1: → Source ANSI string.
Return:     eax = Length of the string in characters.


Procedure:  StrLengthW
File:       \ObjAsm\Code\ObjMem\32\StrLengthW.asm
Purpose:    Determine the length of a WIDE string not including the zero terminating character.
Arguments:  Arg1: → Wide string.
Return:     eax = Length of the string in characters.


Procedure:  StrLowerA
File:       \ObjAsm\Code\ObjMem\32\StrLowerA.asm
Purpose:    Convert all ANSI string characters into lowercase.
Arguments:  Arg1: → Source ANSI string.
Return:     eax → string.
```

```
Procedure:  StrLowerW
File:       \ObjAsm\Code\ObjMem\32\StrLowerW.asm
Purpose:    Convert all WIDE string characters into lowercase.
Arguments:  Arg1: → Source WIDE string.
Return:     eax → string.


Procedure:  StrLRTrimA
File:       \ObjAsm\Code\ObjMem\32\StrLRTrimA.asm
Purpose:    Trim blank characters from the beginning and the end of an ANSI string.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
Return:     Nothing.


Procedure:  StrLRTrimW
File:       \ObjAsm\Code\ObjMem\32\StrLRTrimW.asm
Purpose:    Trim blank characters from the beginning and the end of a WIDE string.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
Return:     Nothing.


Procedure:  StrLScanA
File:       \ObjAsm\Code\ObjMem\32\StrLScanA.asm
Purpose:    Scan for a character from the beginning of an ANSI string.
Arguments:  Arg1: → Source ANSI string.
            Arg2: Character to search.
Return:     eax → Character address or NULL if not found.


Procedure:  StrLScanW
File:       \ObjAsm\Code\ObjMem\32\StrLScanW.asm
Purpose:    Scan for a character from the beginning of a WIDE string.
Arguments:  Arg1: → Source WIDE string.
            Arg2: Character to search for.
Return:     eax → Character address or NULL if not found.


Procedure:  StrLTrimA
File:       \ObjAsm\Code\ObjMem\32\StrLTrimA.asm
Purpose:    Trim blank characters from the beginning of an ANSI string.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
Return:     Nothing.


Procedure:  StrLTrimW
File:       \ObjAsm\Code\ObjMem\32\StrLTrimW.asm
Purpose:    Trim blank characters from the beginning of a WIDE string.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
Return:     Nothing.


Procedure:  StrMidA
File:       \ObjAsm\Code\ObjMem\32\StrMidA.asm
Purpose:    Extract a substring from an ANSI source string.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
            Arg3: Start character index. Index ranges [1 .. String length].
            Arg3: Character count.
Return:     eax = Number of copied characters.


Procedure:  StrMidW
File:       \ObjAsm\Code\ObjMem\32\StrMidW.asm
Purpose:    Extract a substring from a WIDE source string.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
            Arg3: Start character index. Index ranges [1 .. String length].
            Arg3: Character count.
Return:     eax = Number of copied characters.


Procedure:  StrMoveW
File:       \ObjAsm\Code\ObjMem\32\StrMoveW.asm
Purpose:    Move part of a WIDE string. The ending zero charactrer is not appended automatically.
            Source and destination strings may overlap.
```

```
Arguments:  Arg1: → Destination buffer.
            Arg2: → Source WIDE string.
            Arg3: Character count.
Return:     Nothing.


Procedure:  StrNewA
File:       \ObjAsm\Code\ObjMem\32\StrNewA.asm
Purpose:    Allocate a new copy of the source string.
            If the pointer to the source string is NULL, StrNew returns NULL and doesn't allocate
            any memory space. Otherwise, StrNew makes a duplicate of the source string.
            The allocated memory space is Length(String) + ZTC.
Arguments:  Arg1: → Source WIDE string.
Return:     eax → New string copy.


Procedure:  StrNewA_UEFI
File:       \ObjAsm\Code\ObjMem\32\StrNewA_UEFI.asm
Purpose:    Allocate a new copy of the source string.
            If the pointer to the source string is NULL, StrNew returns NULL and doesn't allocate
            any memory space. Otherwise, StrNew makes a duplicate of the source string.
            The allocated memory space is Length(String) + ZTC.
Arguments:  Arg1: → Source WIDE string.
Return:     eax → New string copy.


Procedure:  StrNewW
File:       \ObjAsm\Code\ObjMem\32\StrNewW.asm
Purpose:    Allocate a new copy of the source string.
            If the pointer to the source string is NULL, StrNew returns NULL and doesn't allocate
            any memory space. Otherwise, StrNew makes a duplicate of the source string.
            The allocated memory space is Length(String) + ZTC.
Arguments:  Arg1: → Source WIDE string.
Return:     eax → New string copy.


Procedure:  StrNewW_UEFI
File:       \ObjAsm\Code\ObjMem\32\StrNewW_UEFI.asm
Purpose:    Allocate a new copy of the source string.
            If the pointer to the source string is NULL, StrNew returns NULL and doesn't allocate
            any memory space. Otherwise, StrNew makes a duplicate of the source string.
            The allocated memory space is Length(String) + ZTC.
Arguments:  Arg1: → Source WIDE string.
Return:     eax → New string copy.


Procedure:  StrPosA
File:       \ObjAsm\Code\ObjMem\32\StrPosA.asm
Purpose:    Find the occurence of string 2 into string1.
Arguments:  Arg1: → Source ANSI string.
            Arg2: → Searched ANSI string.
Return:     eax → string occurence or NULL if not found.


Procedure:  StrPosW
File:       \ObjAsm\Code\ObjMem\32\StrPosW.asm
Purpose:    Find the occurence of string 2 into string1.
Arguments:  Arg1: → Source WIDE string.
            Arg2: → Searched WIDE string.
Return:     eax → string occurence or NULL if not found.


Procedure:  StrRepChrA
File:       \ObjAsm\Code\ObjMem\32\StrRepChrA.asm
Purpose:    Create a new string filled with a given char.
Arguments:  Arg1: Used character.
            Arg2: Repetition count.
Return:     eax → New string or NULL if failed.


Procedure:  StrRepChrW
File:       \ObjAsm\Code\ObjMem\32\StrRepChrW.asm
Purpose:    Create a new string filled with a given char.
Arguments:  Arg1: Used character.
            Arg2: Repetition count.
Return:     eax → New string or NULL if failed.


Procedure:  StrRightA
File:       \ObjAsm\Code\ObjMem\32\StrRightA.asm
Purpose:    Copy the right n characters from the source string into the destination buffer.
```

```
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
            Arg3: Character count.
Return:     Nothing.


Procedure:  StrRightW
File:       \ObjAsm\Code\ObjMem\32\StrRightW.asm
Purpose:    Copy the right n characters from the source string into the destination buffer.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
            Arg3: Character count.
Return:     Nothing.


Procedure:  StrRScanA
File:       \ObjAsm\Code\ObjMem\32\StrRScanA.asm
Purpose:    Scan from the end of an ANSI string for a character.
Arguments:  Arg1: → Source ANSI string.
            Arg2: Character to search for.
Return:     eax → Character address or NULL if not found.


Procedure:  StrRScanW
File:       \ObjAsm\Code\ObjMem\32\StrRScanW.asm
Purpose:    Scan from the end of a WIDE string for a character.
Arguments:  Arg1: → Source WIDE string.
            Arg2: Character to search for.
Return:     eax → Character address or NULL if not found.


Procedure:  StrRTrimA
File:       \ObjAsm\Code\ObjMem\32\StrRTrimA.asm
Purpose:    Trim blank characters from the end of an ANSI string.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
Return:     Nothing.


Procedure:  StrRTrimW
File:       \ObjAsm\Code\ObjMem\32\StrRTrimW.asm
Purpose:    Trim blank characters from the end of a WIDE string.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
Return:     Nothing.


Procedure:  StrSizeA
File:       \ObjAsm\Code\ObjMem\32\StrSizeA.asm
Purpose:    Determine the size of an ANSI string including the zero terminating character (ZTC).
Arguments:  Arg1: → ANSI string.
Return:     eax = Size of the string in BYTEs.


Procedure:  StrSizeW
File:       \ObjAsm\Code\ObjMem\32\StrSizeW.asm
Purpose:    Determine the size of a WIDE string including the zero terminating character (ZTC).
Arguments:  Arg1: → Wide string.
Return:     eax = Size of the string in BYTEs.


Procedure:  StrStartsWithA
File:       \ObjAsm\Code\ObjMem\32\StrStartsWithA.asm
Purpose:    Compare the beginning of a string.
Arguments:  Arg1: → Analized string.
            Arg2: → Prefix string.
Return:     eax = TRUE of the beginning matches, otherwise FALSE.


Procedure:  StrStartsWithW
File:       \ObjAsm\Code\ObjMem\32\StrStartsWithW.asm
Purpose:    Compare the beginning of a string.
Arguments:  Arg1: → Analized string.
            Arg2: → Prefix string.
Return:     eax = TRUE of the beginning matches, otherwise FALSE.


Procedure:  StrToSt0A
File:       \ObjAsm\Code\ObjMem\32\StrToSt0A.asm
Purpose:    Load an ANSI string representation of a floating point number into the st(0)
            FPU register.
```

```
Arguments:  Arg1: → ANSI string floating point number.
Return:     eax = Result code f_OK or f_ERROR.
Note:       - Based on the work of Raymond Filiatreault (FpuLib).
            - Source string should not be greater than 19 chars + zero terminator.


Procedure:  StrToSt0W
File:       \ObjAsm\Code\ObjMem\32\StrToSt0W.asm
Purpose:    Load a WIDE string representation of a floating point number into the st(0)
            FPU register.
Arguments:  Arg1: → ANSI string floating point number.
Return:     eax = Result code f_OK or f_ERROR.
Note:       - Based on the work of Raymond Filiatreault (FpuLib).
            - Source string should not be greater than 19 chars + zero terminator.


Procedure:  StrUpperA
File:       \ObjAsm\Code\ObjMem\32\StrUpperA.asm
Purpose:    Convert all ANSI string characters into uppercase.
Arguments:  Arg1: → Source ANSI string.
Return:     eax → String.


Procedure:  StrUpperW
File:       \ObjAsm\Code\ObjMem\32\StrUpperW.asm
Purpose:    Convert all WIDE string characters into uppercase.
Arguments:  Arg1: → Source WIDE string.
Return:     eax → String.


Procedure:  StrW2StrA
File:       \ObjAsm\Code\ObjMem\32\StrW2StrA.asm
Purpose:    Convert a WIDE string into an ANSI string. Wide characters are converted to BYTEs by
            decimation of the high byte.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source WIDE string.
Return:     Nothing.


Procedure:  SysShutdown
File:       \ObjAsm\Code\ObjMem\32\SysShutdown.asm
Purpose:    Shut down the system.
Arguments:  Arg1: Shutdown type.
            Arg2: Shutdown reason (see System Shutdown Reason Codes).
Return:     Nothing.


Procedure:  SysStandby
File:       \ObjAsm\Code\ObjMem\32\SysStandby.asm
Purpose:    Set the system in standby modus.
Arguments:  None.
Return:     Nothing.


Procedure:  uCRC32C
File:       \ObjAsm\Code\ObjMem\32\uCRC32C.asm
Purpose:    Compute the CRC-32C (Castagnoli), using the polynomial 11EDC6F41h from an unaligned
            memory block.
Arguments:  Arg1: → Unaligned memory block.
            Arg2: Memory block size in BYTEs.
Return:     eax = CRC32C.


Procedure:  udword2decA
File:       \ObjAsm\Code\ObjMem\32\udword2decA.asm
Purpose:    Convert an unsigned DWORD to its decimal ANSI string representation.
Arguments:  Arg1: → Destination ANSI string buffer.
            Arg2: DWORD value.
Return:     eax = Number of BYTEs copied to the destination buffer, including the ZTC.
Note:       The destination buffer must be at least 11 BYTEs large to allocate the output string
            (10 ANSI characters + ZTC = 11 BYTEs).


Procedure:  udword2decW
File:       \ObjAsm\Code\ObjMem\32\udword2decW.asm
Purpose:    Convert an unsigned DWORD to its decimal WIDE string representation.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: DWORD value.
Return:     eax = Number of BYTEs copied to the destination buffer, including the ZTC.
Note:       The destination buffer must be at least 22 BYTEs large to allocate the output string
            (10 WIDE characters + ZTC = 22 BYTEs).
```

```
Procedure:  UefiGetErrStrA
File:       \ObjAsm\Code\ObjMem\32\UefiGetErrStrA.asm
Purpose:    Return a description ANSI string from an UEFI error code.
Arguments:  Arg1: UEFI error code.
Return:     eax → Error string.


Procedure:  UefiGetErrStrW
File:       \ObjAsm\Code\ObjMem\32\UefiGetErrStrW.asm
Purpose:    Return a description WIDE string from an UEFI error code.
Arguments:  Arg1: UEFI error code.
Return:     eax → Error string.


Procedure:  uqword2decA
File:       \ObjAsm\Code\ObjMem\32\uqword2decA.asm
Purpose:    Convert an unsigned QWORD to its decimal ANSI string representation.
Arguments:  Arg1: → Destination ANSI string buffer.
            Arg2: QWORD value.
Return:     eax = Number of BYTEs copied to the destination buffer, including the ZTC.
Note:       The destination buffer must be at least 21 BYTEs large to allocate the output string
            (20 ANSI characters + ZTC = 21 BYTEs).


Procedure:  uqword2decW
File:       \ObjAsm\Code\ObjMem\32\uqword2decW.asm
Purpose:    Convert an unsigned QWORD to its decimal WIDE string representation.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: QWORD value.
Return:     eax = Number of BYTEs copied to the destination buffer, including the ZTC.
Note:       The destination buffer must be at least 42 BYTEs large to allocate the output string
            (20 WIDE characters + ZTC = 42 BYTEs).


Procedure:  UTF8ToWide
File:       \ObjAsm\Code\ObjMem\32\UTF8ToWide.asm
Purpose:    Convert an UTF8 byte stream to a WIDE (UTF16) string.
Arguments:  Arg1: → Destination WIDE buffer.
            Arg2: → Source UTF8 BYTE stream. Must be zero terminated.
            Arg3: Destination buffer size in BYTEs.
Return:     eax = Number of BYTEs written.
            ecx = 0: succeeded
            1: buffer full
            2: conversion error
Notes:      - The destination WIDE string is always terminated with a ZTC
            (only if buffer size >= 2).


Procedure:  WaitForProcess
File:       \ObjAsm\Code\ObjMem\32\WaitForProcess.asm
Purpose:    Synchronisation procedure that waits until a process has finished.
Arguments:  Arg1: Process ID
            Arg2: Timeout value in ms.
Return:     eax = Wait result (WAIT_ABANDONED, WAIT_OBJECT_0 or WAIT_TIMEOUT).


Procedure:  WideToUTF8
File:       \ObjAsm\Code\ObjMem\32\WideToUTF8.asm
Purpose:    Convert an WIDE string to an UTF8 encoded stream.
Arguments:  Arg1: → Destination buffer.
            Arg2: → Source WIDE string.
            Arg3: Destination buffer size in BYTEs.
Return:     eax = Number of BYTEs written.
            ecx = 0: succeeded
            1: buffer full
Notes:      - The destination stream is always zero terminated.


Procedure:  WndFadeIn
File:       \ObjAsm\Code\ObjMem\32\WndFadeIn.asm
Purpose:    Fade in a window when WS_EX_LAYERED is set.
Arguments:  Arg1: Window HANDLE.
            Arg2: Transparency start value.
            Arg3: Transparency end value.
            Arg4: Transparency increment value.
            Arg5: Delay between steps.
Return:     Nothing.
```

```
Procedure:  WndFadeOut
File:       \ObjAsm\Code\ObjMem\32\WndFadeOut.asm
Purpose:    Fade out a window when WS_EX_LAYERED is set.
Arguments:  Arg1: Window HANDLE.
            Arg2: Transparency start value.
            Arg3: Transparency end value.
            Arg4: Transparency decrement value.
            Arg5: Delay between steps.
Return:     Nothing.


Procedure:  word2hexA
File:       \ObjAsm\Code\ObjMem\32\word2hexA.asm
Purpose:    Convert a WORD to its hexadecimal ANSI string representation.
Arguments:  Arg1: → Destination ANSI string buffer.
            Arg2: WORD value.
Return:     Nothing.
Notes:      The destination buffer must be at least 5 BYTEs large to allocate the output string
            (4 character BYTEs + ZTC = 5 BYTEs).


Procedure:  word2hexW
File:       \ObjAsm\Code\ObjMem\32\word2hexW.asm
Purpose:    Convert a WORD to its hexadecimal WIDE string representation.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: WORD value.
Return:     Nothing.
Notes:      The destination buffer must be at least 9 BYTEs large to allocate the output string
            (4 character WORDs + ZTC = 9 BYTEs).
```

8. 64 Bit Code

```
Procedure:  aCRC32C
File:       \ObjAsm\Code\ObjMem\64\aCRC32C.asm
Purpose:    Compute the CRC-32C (Castagnoli), using the polynomial 11EDC6F41h from an aligned
            memory block.
Arguments:  Arg1: → Aligned memory block.
            Arg2: Memory block size in BYTEs.
Return:     eax = CRC32C.


Procedure:  ActivatePrevInstanceA
File:       \ObjAsm\Code\ObjMem\64\ActivatePrevInstanceA.asm
Purpose:    Activate a previously existing instance of an application.
Arguments:  Arg1: → ANSI application name.
            Arg2: → ANSI class name.
Return:     rax = TRUE if activated, otherwise FALSE.


Procedure:  ActivatePrevInstanceW
File:       \ObjAsm\Code\ObjMem\64\ActivatePrevInstanceW.asm
Purpose:    Activate a previously existing instance of an application.
Arguments:  Arg1: → WIDE application name.
            Arg2: → WIDE class name.
Return:     rax = TRUE if activated, otherwise FALSE.


Procedure:  AreVisualStylesEnabled
File:       \ObjAsm\Code\ObjMem\64\AreVisualStylesEnabled.asm
Purpose:    Determine if there is an activated theme for the running application
Arguments:  None.
Return:     rax = TRUE if the application is themed, otherwise FALSE.


Procedure:  bin2dwordA
File:       \ObjAsm\Code\ObjMem\64\bin2dwordA.asm
Purpose:    Load an ANSI string binary representation of a DWORD.
Arguments:  Arg1: → ANSI binary string.
Return:     eax = DWORD.


Procedure:  bin2dwordW
File:       \ObjAsm\Code\ObjMem\64\bin2dwordW.asm
Purpose:    Load an WIDE string binary representation of a DWORD.
Arguments:  Arg1: → WIDE binary string.
Return:     eax = DWORD.


Procedure:  bin2qwordA
File:       \ObjAsm\Code\ObjMem\64\bin2qwordA.asm
Purpose:    Load an ANSI string binary representation of a QWORD.
Arguments:  Arg1: → ANSI binary string.
Return:     rax = QWORD.


Procedure:  bin2qwordW
File:       \ObjAsm\Code\ObjMem\64\bin2qwordW.asm
Purpose:    Load an WIDE string binary representation of a QWORD.
Arguments:  Arg1: → Wide binary string.
Return:     rax = QWORD.


Procedure:  Bmp2Rgn
File:       \ObjAsm\Code\ObjMem\64\Bmp2Rgn.asm
Purpose:    Create a GDI region based on a device dependant or independent bitmap (DDB or DIB).
            This region is defined by the non transparent area delimited by the transparent color.
Arguments:  Arg1: Bitmap HANDLE.
            Arg2: RGB transparet color.
Return:     rax = Region HANDLE or zero if failed.


Procedure:  BStrAlloc
File:       \ObjAsm\Code\ObjMem\64\BStrAlloc.asm
Purpose:    Allocate space for a BStr with n characters. The length field is set to zero.
Arguments:  Arg1: Character count.
Return:     rax → New allocated BStr or NULL if failed.
```

```
Procedure:  BStrCat
File:       \ObjAsm\Code\ObjMem\64\BStrCat.asm
Purpose:    Concatenate 2 BStrs.
Arguments:  Arg1: Destination BStr.
            Arg2: Source BStr.
Return:     Nothing.


Procedure:  BStrCatChar
File:       \ObjAsm\Code\ObjMem\64\BStrCatChar.asm
Purpose:    Append a character to the end of a BStr.
Arguments:  Arg1: Destination BStr.
            Arg2: WIDE character.
Return:     Nothing.


Procedure:  BStrCCatChar
File:       \ObjAsm\Code\ObjMem\64\BStrCCatChar.asm
Purpose:    Append a WIDE character to a BStr with length limitation.
Arguments:  Arg1: → Destination BStr.
            Arg2: → WIDE character.
Return:     rax → BStr or NULL if failed.


Procedure:  BStrCECat
File:       \ObjAsm\Code\ObjMem\64\BStrCECat.asm
Purpose:    Concatenate 2 BStrs with length limitation and return the the address of the ZTC.
            The destination string buffer should have at least enough room for the maximum number
            of characters + 1.
Arguments:  Arg1: → Destination BStr.
            Arg2: → Source BStr.
            Arg3: Maximal number of charachters the destination string can hold including the ZTC.
Return:     rax = NULL or → ZTC.


Procedure:  BStrCECopy
File:       \ObjAsm\Code\ObjMem\64\BStrCECopy.asm
Purpose:    Copy the the source BStr with length limitation and return the address of the ZTC.
            The destination buffer should hold the maximum number of characters + 1.
Arguments:  Arg1: → Destination BStr.
            Arg2: → Source BStr.
            Arg3: Maximal number of charachters the destination string can hold including the ZTC.
Return:     rax = NULL or → ZTC.


Procedure:  BStrCNew
File:       \ObjAsm\Code\ObjMem\64\BStrCNew.asm
Purpose:    Allocate a new copy of the source BStr with length limitation.
            If the pointer to the source string is NULL, BStrCNew returns NULL and doesn't
            allocate any space. Otherwise, StrCNew makes a duplicate of the source string.
            The maximal size of the new string is limited to the second parameter.
Arguments:  Arg1: → Source BStr.
            Arg2: Maximal character count.
Return:     rax → New BStr copy or NULL.


Procedure:  BStrCopy
File:       \ObjAsm\Code\ObjMem\64\BStrCopy.asm
Purpose:    Copy a BStr to a destination buffer.
Arguments:  Arg1: Destination BStr buffer.
            Arg2: Source BStr.
Return:     Nothing.


Procedure:  BStrCScan
File:       \ObjAsm\Code\ObjMem\64\BStrCScan.asm
Purpose:    Scan from the beginning of a BStr for a character with length limitation.
Arguments:  Arg1: → Source WIDE string.
            Arg2: Maximal character count.
            Arg3: WIDE character to search for.
Return:     rax → Character address or NULL if not found.


Procedure:  BStrDispose
File:       \ObjAsm\Code\ObjMem\64\BStrDispose.asm
Purpose:    Free the memory allocated for the string using BStrNew, BStrCNew, BStrLENew or
            BStrAlloc.
            If the pointer to the string is NULL, BStrDispose does nothing.
Arguments:  Arg1: → BStr.
Return:     Nothing.
```

```
Procedure:  BStrECat
File:       \ObjAsm\Code\ObjMem\64\BStrECat.asm
Purpose:    Append a BStr to another and return the address of the ZTC.
            BStrCat does not perform any length checking. The destination buffer must have room
            for at least BStrLength(Destination) + BStrLength(Source) + 1 characters.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → Added BStr.
Return:     rax → ZTC.


Procedure:  BStrECatChar
File:       \ObjAsm\Code\ObjMem\64\BStrECatChar.asm
Purpose:    Append a WIDE character to a BStr and return the address of the ZTC.
            BStrECatChar does not perform any length checking. The destination buffer must have
            enough room for at least BStrLength(Destination) + 1 + 1 characters.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → WIDE character.
Return:     rax → ZTC.


Procedure:  BStrECopy
File:       \ObjAsm\Code\ObjMem\64\BStrECopy.asm
Purpose:    Copy a BStr to a buffer and return the address of the ZTC.
            Source and destination strings may overlap.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → Source BStr buffer.
Return:     rax → ZTC.


Procedure:  BStrEnd
File:       \ObjAsm\Code\ObjMem\64\BStrEnd.asm
Purpose:    Get the address of the ZTC.
Arguments:  Arg1: → Source BStr.
Return:     rax → ZTC.


Procedure:  BStrEndsWith
File:       \ObjAsm\Code\ObjMem\64\BStrEndsWith.asm
Purpose:    Compare the ending of a BSTR.
Arguments:  Arg1: → Analized BSTR.
            Arg2: → Suffix BSTR.
Return:     eax = TRUE of the ending matches, otherwise FALSE.


Procedure:  BStrFillChr
File:       \ObjAsm\Code\ObjMem\64\BStrFillChr.asm
Purpose:    Fill a preallocated BSTR with a character.
Arguments:  Arg1: → String.
            Arg2: Character.
            Arg3: Character Count.
Return:     Nothing.


Procedure:  BStrLeft
File:       \ObjAsm\Code\ObjMem\64\BStrLeft.asm
Purpose:    Extract the left n characters of the source BStr.
Arguments:  Arg1: → Destination BStr.
            Arg2: → Source BStr.
Return:     rax = Number of copied characters, not including the ZTC.


Procedure:  BStrLength
File:       \ObjAsm\Code\ObjMem\64\BStrLength.asm
Purpose:    Determine the length of a BStr not including the ZTC.
Arguments:  Arg1: → Source BStr.
Return:     rax = Length of the string in characters.


Procedure:  BStrLRTrim
File:       \ObjAsm\Code\ObjMem\64\BStrLRTrim.asm
Purpose:    Trim blank characters from the beginning and the end of a BStr.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → Source BStr.
Return:     Nothing.


Procedure:  BStrLTrim
File:       \ObjAsm\Code\ObjMem\64\BStrLTrim.asm
Purpose:    Trim blank characters from the beginning of a BStr.
```

```
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → Source BStr.
Return:     Nothing.


Procedure:  BStrMid
File:       \ObjAsm\Code\ObjMem\64\BStrMid.asm
Purpose:    Extract a substring from a BStr string.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → Source Bstr.
            Arg3: Start character index. Index ranges [0 .. length-1].
            Arg3: Character count.
Return:     eax = String length.


Procedure:  BStrMove
File:       \ObjAsm\Code\ObjMem\64\BStrMove.asm
Purpose:    Move part of a BStr. The ZTC is not appended automatically.
            Source and destination strings may overlap.
Arguments:  Arg1: → Destination BStr.
            Arg2: → Source BStr.
            Arg3: Character count.
Return:     Nothing.


Procedure:  BStrNew
File:       \ObjAsm\Code\ObjMem\64\BStrNew.asm
Purpose:    Allocate an new copy of the source BStr.
            If the pointer to the source string is NULL or points to an empty string, BStrNew
            returns NULL and doesn't allocate any heap space. Otherwise, BStrNew makes a duplicate
            of the source string.
            The allocated space is Length(String) + 1 character.
Arguments:  Arg1: → Source BStr.
Return:     rax → New BStr copy or NULL.


Procedure:  BStrRepChr
File:       \ObjAsm\Code\ObjMem\64\BStrRepChr.asm
Purpose:    Create a new BSTR filled with a given char.
Arguments:  Arg1: Used character.
            Arg2: Repetition count.
Return:     xax → New BSTR or NULL if failed.


Procedure:  BStrRight
File:       \ObjAsm\Code\ObjMem\64\BStrRight.asm
Purpose:    Copy the right n characters from the source string into the destination BStr, that
            must
            have enought room for the new BStr.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → Source BStr.
            Arg3: Character count.
Return:     rax = Copied characters.


Procedure:  BStrRTrim
File:       \ObjAsm\Code\ObjMem\64\BStrRTrim.asm
Purpose:    Trim blank characters from the end of a BStr.
Arguments:  Arg1: → Destination BStr buffer.
            Arg2: → Source BStr.
Return:     Nothing.


Procedure:  BStrSize
File:       \ObjAsm\Code\ObjMem\64\BStrSize.asm
Purpose:    Determine the size of a BStr including the ZTC + leading DWORD.
Arguments:  Arg1: → Source BStr.
Return:     rax = String size including the length field and ZTC in BYTEs.


Procedure:  BStrStartsWith
File:       \ObjAsm\Code\ObjMem\64\BStrStartsWith.asm
Purpose:    Compare the beginning of a BSTR.
Arguments:  Arg1: → Analized BSTR.
            Arg2: → Prefix BSTR.
Return:     eax = TRUE of the beginning matches, otherwise FALSE.


Procedure:  byte2hexA
File:       \ObjAsm\Code\ObjMem\64\byte2hexA.asm
Purpose:    Convert a BYTE to its hexadecimal ANSI string representation.
```

```
Arguments:  Arg1: → Destination ANSI string buffer.
            Arg2: BYTE value.
Return:     Nothing.
Notes:      The destination buffer must be at least 3 BYTEs large to allocate the output string
            (2 character BYTEs + ZTC = 3 BYTEs).


Procedure:  byte2hexW
File:       \ObjAsm\Code\ObjMem\64\byte2hexW.asm
Purpose:    Convert a BYTE to its hexadecimal WIDE string representation.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: BYTE value.
Return:     Nothing.
Notes:      The destination buffer must be at least 5 BYTEs large to allocate the output string
            (2 character WORDs + ZTC = 5 BYTEs).


Procedure:  CalcVarianceDW
File:       \ObjAsm\Code\ObjMem\64\CalcVarianceDW.asm
Purpose:    Calculate the MSE of an array of DWORDs.
Arguments:  Arg1: → Array of DWORDs.
            Arg2: QWORD Array count.
            Arg3: → Variance.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:
               https://www.mun.ca/biology/scarr/Simplified_calculation_of_variance.html#:~:text=A%
            20more%20straightforward%20calculation%20recognizes,i2%20)%20%2F%20n%20%2D%202
            https://www.mun.ca/biology/scarr/Mean_&_Variance.html#:~:text=easily%20calculated%20as
Formulas:   Var = Y2/N-(Y/N)^2  or  (Y2*N-Y^2)/N^2
            where Y: Sum(y), Y2: Sum(y^2), N:Population count = Array size.


Procedure:  CalcVarianceQW
File:       \ObjAsm\Code\ObjMem\64\CalcVarianceQW.asm
Purpose:    Calculate the MSE of an array of QWORDs.
Arguments:  Arg1: → Array of QWORDs.
            Arg2: QWORD Array count.
            Arg3: → Variance.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:
               https://www.mun.ca/biology/scarr/Simplified_calculation_of_variance.html#:~:text=A%
            20more%20straightforward%20calculation%20recognizes,i2%20)%20%2F%20n%20%2D%202
            https://www.mun.ca/biology/scarr/Mean_&_Variance.html#:~:text=easily%20calculated%20as
Formulas:   Var = Y2/N-(Y/N)^2  or  (Y2*N-Y^2)/N^2
            where Y: Sum(y), Y2: Sum(y^2), N:Population count = Array size.


Procedure:  CalcVarianceR4
File:       \ObjAsm\Code\ObjMem\64\CalcVarianceR4.asm
Purpose:    Calculate the MSE of an array of REAL4s.
Arguments:  Arg1: → Array of REAL4s.
            Arg2: QWORD Array count.
            Arg3: → Variance.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:
               https://www.mun.ca/biology/scarr/Simplified_calculation_of_variance.html#:~:text=A%
            20more%20straightforward%20calculation%20recognizes,i2%20)%20%2F%20n%20%2D%202
            https://www.mun.ca/biology/scarr/Mean_&_Variance.html#:~:text=easily%20calculated%20as
Formulas:   Var = Y2/N-(Y/N)^2  or  (Y2*N-Y^2)/N^2
            where Y: Sum(y), Y2: Sum(y^2), N:Population count = Array size.


Procedure:  CalcVarianceR8
File:       \ObjAsm\Code\ObjMem\64\CalcVarianceR8.asm
Purpose:    Calculate the MSE of an array of REAL8s.
Arguments:  Arg1: → Array of REAL8s.
            Arg2: QWORD Array count.
            Arg3: → Variance.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:
               https://www.mun.ca/biology/scarr/Simplified_calculation_of_variance.html#:~:text=A%
            20more%20straightforward%20calculation%20recognizes,i2%20)%20%2F%20n%20%2D%202
            https://www.mun.ca/biology/scarr/Mean_&_Variance.html#:~:text=easily%20calculated%20as
Formulas:   Var = Y2/N-(Y/N)^2  or  (Y2*N-Y^2)/N^2
            where Y: Sum(y), Y2: Sum(y^2), N:Population count = Array size.


Procedure:  CenterForm
File:       \ObjAsm\Code\ObjMem\64\CenterForm.asm
Purpose:    Calculate the starting coordinate of a window based on the screen and the window size.
Arguments:  Arg1: Window size in pixel.
```

```
            Arg2: Screen size in pixel.
Return:     eax = Starting point in pixel.


Procedure:  ComEventsAdvice
File:       \ObjAsm\Code\ObjMem\64\ComEventsAdvice.asm
Purpose:    Notificate the Event source that pISink will recieve Events.
Arguments:  Arg1: → Any Source Object Interface.
            Arg2: → Sink IUnknown Interface.
            Arg3: → IID of the outgoing interface whose connection point object is being
            requested (defined by the Source to communicate and implemented by the Sink).
            Arg4: → ConnectionPoint interface pointer.
            Arg5: → DWORD Cookie.
Return:     eax = HRESULT.


Procedure:  ComEventsUnadvice
File:       \ObjAsm\Code\ObjMem\64\ComEventsUnadvice.asm
Purpose:    Notificate the Event source that pISource will NOT recieve Events any more.
Arguments:  Arg1: → Previous ConnectionPoint interface.
            Arg2: DWORD Cookie received from previous ComEventsAdvice call.
Return:     eax = HRESULT.


Procedure:  ComGetErrStrA
File:       \ObjAsm\Code\ObjMem\64\ComGetErrStrA.asm
Purpose:    Return a description ANSI string from a COM error code.
Arguments:  Arg1: COM error code.
Return:     rax → Error string.


Procedure:  ComGetErrStrW
File:       \ObjAsm\Code\ObjMem\64\ComGetErrStrW.asm
Purpose:    Return a description WIDE string from a COM error code.
Arguments:  Arg1: COM error code.
Return:     rax → Error string.


Procedure:  ComPtrAssign
File:       \ObjAsm\Code\ObjMem\64\ComPtrAssign.asm
Purpose:    First increments the reference count of the new interface and then releases any
            existing interface pointer.
Arguments:  Arg1: → Old Interface pointer.
            Arg2: New Interface pointer.


Procedure:  CreatePathA
File:       \ObjAsm\Code\ObjMem\64\CreatePathA.asm
Purpose:    Create a path on the destination drive.
Arguments:  Arg1: → ANSI path string.
Return:     Nothing.


Procedure:  CreatePathW
File:       \ObjAsm\Code\ObjMem\64\CreatePathW.asm
Purpose:    Create a path on the destination drive.
Arguments:  Arg1: → WIDE path string.
Return:     Nothing.


Procedure:  DbgClose
File:       \ObjAsm\Code\ObjMem\64\DbgClose.asm
Purpose:    Close the connection to the output device.
Arguments:  None.
Return:     Nothing.


Procedure:  DbgConOpen
File:       \ObjAsm\Code\ObjMem\64\DbgConOpen.asm
Purpose:    Open a new console for the calling process.
Arguments:  None.
Return:     rax = TRUE if it was opened, otherwise FALSE.


Procedure:  DbgLogOpen
File:       \ObjAsm\Code\ObjMem\64\DbgLogOpen.asm
Purpose:    Open a Log-File.
Arguments:  None.
Return:     rax = TRUE if it was opened, otherwise FALSE.
```

```
Procedure:  DbgOutApiErr
File:       \ObjAsm\Code\ObjMem\64\DbgOutApiErr.asm
Purpose:    Identify a API error with a string.
Arguments:  Arg1: Api error code obtained with GetLastError.
            Arg2: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgOutBitmap
File:       \ObjAsm\Code\ObjMem\64\DbgOutBitmap.asm
Purpose:    Send a bitmap to the Debug Center Window.
Arguments:  Arg1: Bitamp HANDLE.
            Arg2: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgOutCmd
File:       \ObjAsm\Code\ObjMem\64\DbgOutCmd.asm
Purpose:    Send a command to a specific Debug window.
Arguments:  Arg1: Command ID [BYTE].
            Arg2: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgOutComErr
File:       \ObjAsm\Code\ObjMem\64\DbgOutComErr.asm
Purpose:    Identify a COM error with a string.
Arguments:  Arg1: COM error ID.
            Arg2: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgOutComponent
File:       \ObjAsm\Code\ObjMem\64\DbgOutComponent.asm
Purpose:    Identify a COM-Component.
Arguments:  Arg1: → CSLID.
            Arg2: Foreground color.
            Arg2: → Destination Window WIDE name.


Procedure:  DbgOutComponentName
File:       \ObjAsm\Code\ObjMem\64\DbgOutComponentName.asm
Purpose:    Identify a COM-Component.
Arguments:  Arg1: → CSLID.
            Arg2: Foreground color.
            Arg2: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgOutFPU
File:       \ObjAsm\Code\ObjMem\64\DbgOutFPU.asm
Purpose:    Display the content of the FPU.
Arguments:  Arg1: → Destination Window WIDE name.
            Arg2: Text RGB color.
Return:     Nothing.


Procedure:  DbgOutFPU_UEFI
File:       \ObjAsm\Code\ObjMem\64\DbgOutFPU_UEFI.asm
Purpose:    Display the content of the FPU.
Arguments:  Arg1: → Destination Window WIDE name.
            Arg2: Text RGB color.
Return:     Nothing.


Procedure:  DbgOutInterface
File:       \ObjAsm\Code\ObjMem\64\DbgOutInterface.asm
Purpose:    Identify a COM-Interface.
Arguments:  Arg1: → CSLID.
            Arg2: Foreground color.
            Arg2: → Destination Window WIDE name.


Procedure:  DbgOutInterfaceName
File:       \ObjAsm\Code\ObjMem\64\DbgOutInterfaceName.asm
Purpose:    Identify a COM-Interface.
Arguments:  Arg1: → CSLID.
            Arg2: Foreground color.
            Arg2: → Destination window WIDE name.
Return:     Nothing.
```

```
Procedure:  DbgOutMem
File:       \ObjAsm\Code\ObjMem\64\DbgOutMem.asm
Purpose:    Output the content of a memory block.
Arguments:  Arg1: → Memory block.
            Arg2: Memory block size.
            Arg3: Representation format.
            Arg4: Memory output color.
            Arg5: Representation output color.
            Arg6: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgOutMem_UEFI
File:       \ObjAsm\Code\ObjMem\64\DbgOutMem_UEFI.asm
Purpose:    Output the content of a memory block.
Arguments:  Arg1: → Memory block.
            Arg2: Memory block size.
            Arg3: Representation format.
            Arg4: Memory output color.
            Arg5: Representation output color.
            Arg6: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgOutMsg
File:       \ObjAsm\Code\ObjMem\64\DbgOutMsg.asm
Purpose:    Identify a windows message with a string.
Arguments:  Arg1: Windows message ID.
            Arg2: Foreground color.
            Arg3: → Destination window WIDE name.
Return:     Nothing.


Procedure:  DbgOutTextA
File:       \ObjAsm\Code\ObjMem\64\DbgOutTextA.asm
Purpose:    Send an ANSI string to the debug output device.
Arguments:  Arg1: → Zero terminated ANSI string.
            Arg2: Color value.
            Arg3: Effect value (DBG_EFFECT_XXX).
            Arg4: → Destination window WIDE name.
Return:     Nothing.


Procedure:  DbgOutTextA_UEFI
File:       \ObjAsm\Code\ObjMem\64\DbgOutTextA_UEFI.asm
Purpose:    Send an ANSI string to the debug output device.
Arguments:  Arg1: → Zero terminated ANSI string.
            Arg2: Color value.
            Arg3: Effect value (DBG_EFFECT_XXX).
            Arg4: → Destination window WIDE name.
Return:     Nothing.


Procedure:  DbgOutTextCA
File:       \ObjAsm\Code\ObjMem\64\DbgOutTextCA.asm
Purpose:    Send a counted ANSI string to the debug output device.
Arguments:  Arg1: → Null terminated WIDE string.
            Arg2: Maximal character count.
            Arg3: Color value.
            Arg4: Effect value (DBG_EFFECT_XXX).
            Arg5: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgOutTextCW
File:       \ObjAsm\Code\ObjMem\64\DbgOutTextCW.asm
Purpose:    Send a counted WIDE string to the debug output device.
Arguments:  Arg1: → Null terminated WIDE string.
            Arg2: Maximal character count.
            Arg3: Color value.
            Arg4: Effect value (DBG_EFFECT_XXX).
            Arg5: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgOutTextW
File:       \ObjAsm\Code\ObjMem\64\DbgOutTextW.asm
Purpose:    Send a WIDE string to the debug output device.
Arguments:  Arg1: → Zero terminated WIDE string.
            Arg2: Color value.
```

```
            Arg3: Effect value (DBG_EFFECT_XXX)
            Arg4: → Destination window WIDE name.
Return:     Nothing.


Procedure:  DbgOutTextW_UEFI
File:       \ObjAsm\Code\ObjMem\64\DbgOutTextW_UEFI.asm
Purpose:    Send a WIDE string to the debug output device.
Arguments:  Arg1: → Zero terminated WIDE string.
            Arg2: Color value.
            Arg3: Effect value (DBG_EFFECT_XXX).
            Arg4: → Destination window WIDE name.
Return:     Nothing.


Procedure:  DbgShowObjectHeader
File:       \ObjAsm\Code\ObjMem\64\DbgShowObjectHeader.asm
Purpose:    Output heading object information.
Arguments:  Arg1: → Object Name.
            Arg2: → Instance.
            Arg3: Text RGB color.
            Arg3: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgShowObjectHeader_UEFI
File:       \ObjAsm\Code\ObjMem\64\DbgShowObjectHeader_UEFI.asm
Purpose:    Output heading object information.
Arguments:  Arg1: → Object Name.
            Arg2: → Instance.
            Arg3: Text RGB color.
            Arg3: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgShowTraceMethod
File:       \ObjAsm\Code\ObjMem\64\DbgShowTraceMethod.asm
Purpose:    Output trace information about a method.
Arguments:  Arg1: → Method Name.
            Arg2: Method count.
            Arg3: Method ticks.
            Arg4: → Destination Window WIDE name.
Return:     Nothing.


Procedure:  DbgWndOpen
File:       \ObjAsm\Code\ObjMem\64\DbgWndOpen.asm
Purpose:    Open a "Debug Center" instance.
Arguments:  None.
Return:     eax = TRUE if it was opened, otherwise FALSE.


Procedure:  dec2dwordA
File:       \ObjAsm\Code\ObjMem\64\dec2dwordA.asm
Purpose:    Convert a decimal ANSI string to a DWORD.
Arguments:  Arg1: → Source ANSI string. Possible leading characters are " ", tab, "+" and "-",
            followed by a sequence of chars between "0".."9" and finalized by a ZTC.
            Other characters terminate the convertion returning zero.
Return:     eax = Converted DWORD.
            rcx = Conversion result. Zero if succeeded, otherwise failed.


Procedure:  dec2dwordW
File:       \ObjAsm\Code\ObjMem\64\dec2dwordW.asm
Purpose:    Convert a decimal WIDE string to a DWORD.
Arguments:  Arg1: → Source WIDE string. Possible leading characters are " ", tab, "+" and "-",
            followed by a sequence of chars between "0".."9" and finalized by a ZTC.
            Other characters terminate the convertion returning zero.
Return:     rax = Converted DWORD.
            rcx = Conversion result. Zero if succeeded, otherwise failed.


Procedure:  DisableCPUSerialNumber
File:       \ObjAsm\Code\ObjMem\64\DisableCPUSerialNumber.asm
Purpose:    Disable the reading of the CPU serial number.
Arguments:  None.
Return:     Nothing.


Procedure:  DllErr2StrA
File:       \ObjAsm\Code\ObjMem\64\DllErr2StrA.asm
```

```
Purpose:    Translate an error code to an ANSI string stored in a DLL.
Arguments:  Arg1: Error code.
            Arg2: → preallocated ANSI character buffer.
            Arg3: Buffer size in characters, inclusive ZTC.
            Arg4: → DLL ANSI name.
Return:     Nothing.


Procedure:  DllErr2StrW
File:       \ObjAsm\Code\ObjMem\64\DllErr2StrW.asm
Purpose:    Translate an error code to a WIDE string stored in a DLL.
Arguments:  Arg1: Error code.
            Arg2: → WIDE character buffer.
            Arg3: Buffer size in characters, inclusive ending terminator.
            Arg4: → DLL WIDE name.
Return:     Nothing.


Procedure:  DrawTransparentBitmap
File:       \ObjAsm\Code\ObjMem\64\DrawTransparentBitmap.asm
Purpose:    Draw a bitmap with transparency on a device context.
Arguments:  Arg1: DC HANDLE.
            Arg2: Bitmap HANDLE to draw.
            Arg3; X start position on DC.
            Arg4: Y start position on DC.
            Arg5: RGB transparent color. Use TBM_FIRSTPIXEL to indicate that the pixel in the
            upper left corner contains the transparent color.
Return:     Nothing.
Notes:      Original source by Microsoft.
            "HOWTO: Drawing Transparent Bitmaps (Q79212)"
            (http://support.microsoft.com/default.aspx?scid=kb;EN-US;q79212)
            Transcribed by Ernest Murphy.


Procedure:  dword2binA
File:       \ObjAsm\Code\ObjMem\64\dword2binA.asm
Purpose:    Convert a DWORD to its binary ANSI string representation.
Arguments:  Arg1: → Destination buffer.
            Arg2: DWORD value.
Return:     Nothing.
Notes:      The destination buffer must be at least 33 BYTEs large to allocate the output string
            (32 character BYTEs + ZTC = 33 BYTEs).


Procedure:  dword2binW
File:       \ObjAsm\Code\ObjMem\64\dword2binW.asm
Purpose:    Convert a DWORD to its binary WIDE string representation.
Arguments:  Arg1: → Destination buffer.
            Arg2: DWORD value.
Return:     Nothing.
Notes:      The destination buffer must be at least 66 BYTEs large to allocate the output string
            (32 character WORDs + ZTC = 66 BYTEs).


Procedure:  dword2hexA
File:       \ObjAsm\Code\ObjMem\64\dword2hexA.asm
Purpose:    Convert a DWORD to its hexadecimal ANSI string representation.
Arguments:  Arg1: → Destination ANSI string buffer.
            Arg2: DWORD value.
Return:     Nothing.
Notes:      The destination buffer must be at least 9 BYTEs large to allocate the output string
            (8 character BYTEs + ZTC = 9 BYTEs).


Procedure:  dword2hexW
File:       \ObjAsm\Code\ObjMem\64\dword2hexW.asm
Purpose:    Convert a DWORD to its hexadecimal WIDE string representation.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: DWORD value.
Return:     Nothing.
Notes:      The destination buffer must be at least 18 BYTEs large to allocate the output string
            (8 character WORDs + ZTC = 18 BYTEs).


Procedure:  EHandler
File:       \ObjAsm\Code\ObjMem\64\EHandler.asm
Purpose:    ASM exception handler
Arguments:  Arg1: → Exception Record.
            Arg2: → Establisher Frame.
            Arg3: → ContextRecord
            Arg4: → DispatcherContext
```

```
Link:       https://docs.microsoft.com/en-us/cpp/build/language-specific-handler
            http://www.nynaeve.net/?p=113
Return:     rax = ExceptionContinueSearch.


Procedure:  Err2StrA
File:       \ObjAsm\Code\ObjMem\64\Err2StrA.asm
Purpose:    Translate a system error code to an ANSI string.
Arguments:  Arg1: Error code.
            Arg2: → ANSI string buffer.
            Arg3: Buffer size in characters, inclusive ZTC.
Return:     Nothing.


Procedure:  Err2StrW
File:       \ObjAsm\Code\ObjMem\64\Err2StrW.asm
Purpose:    Translate a system error code to a WIDE string.
Arguments:  Arg1: Error code.
            Arg2: → WIDE string buffer.
            Arg3: Buffer size in characters, inclusive ZTC.
Return:     Nothing.


Procedure:  ErrorMessageBoxA
File:       \ObjAsm\Code\ObjMem\64\ErrorMessageBoxA.asm
Purpose:    Show a messagebox containing an error string in the locale lenguage and an user
            string.
Arguments:  Arg1: Messagebox parent window HANDLE.
            Arg2: → User ANSI string.
            Arg3: Locale ID.
            Arg4: API error code.
Return:     Nothing.


Procedure:  ErrorMessageBoxW
File:       \ObjAsm\Code\ObjMem\64\ErrorMessageBoxW.asm
Purpose:    Show a messagebox containing an error string in the locale lenguage and an user
            string.
Arguments:  Arg1: Messagebox parent window HANDLE.
            Arg2: → User WIDE string.
            Arg3: Locale ID.
            Arg4: API error code.
Return:     Nothing.


Procedure:  FileExistA
File:       \ObjAsm\Code\ObjMem\64\FileExistA.asm
Purpose:    Check the existence of a file.
Arguments:  Arg1: → ANSI file name.
Return:     rax = TRUE if the file exists, otherwise FALSE.


Procedure:  FileExistW
File:       \ObjAsm\Code\ObjMem\64\FileExistW.asm
Purpose:    Check the existence of a file.
Arguments:  Arg1: → WIDE file name.
Return:     rax = TRUE if the file exists, otherwise FALSE.


Procedure:  FindFileA
File:       \ObjAsm\Code\ObjMem\64\FindFileA.asm
Purpose:    Search for a file in a list of paths.
Arguments:  Arg1: → File name.
            Arg2: → List of path strings. The end of the list is indicated with a ZTC.
            Arg3: → Buffer containing the full path and file name in which the file was found.
            Buffer length = MAX_PATH.
Return:     eax = Number of chars copied to the destination buffer. 0 if the file was not found.
            Example: invoke FindFile, $OfsCStr("free.inc"), $OfsCStr("\Here*",0), addr cBuf
            Search free.inc in all \Here and suddirectories.


Procedure:  FindFileW
File:       \ObjAsm\Code\ObjMem\64\FindFileW.asm
Purpose:    Search for a file in a list of paths.
Arguments:  Arg1: → File name.
            Arg2: → List of path strings. The end of the list is indicated with a ZTC.
            Arg3: → Buffer containing the full path and file name in which the file was found.
            Buffer length = MAX_PATH.
Return:     eax = Number of chars copied to the destination buffer. 0 if the file was not found.
            Example: invoke FindFile, $OfsCStr("free.inc"), $OfsCStr("\Here*",0), addr cBuf
            Search free.inc in all \Here and suddirectories.
```

```
Procedure:  FindModuleByAddrA
File:       \ObjAsm\Code\ObjMem\64\FindModuleByAddrA.asm
Purpose:    Find the module name from an address on a WinNT system.
Arguments:  Arg1: Address.
            Arg2: → ANSI module name buffer.
Return:     eax = Number of characters copied into the buffer.


Procedure:  FindModuleByAddrW
File:       \ObjAsm\Code\ObjMem\64\FindModuleByAddrW.asm
Purpose:    Find the module name from an address on a WinNT system.
Arguments:  Arg1: Address.
            Arg2: → WIDE module name buffer.
Return:     eax = Number of characters copied into the buffer.


Procedure:  GetAncestorID
File:       \ObjAsm\Code\ObjMem\64\GetAncestorID.asm
Purpose:    Retrieve the ancestor type ID of an object type ID.
Arguments:  Arg1: → Object class ID.
Return:     eax = Ancestor type ID or zero if not found.


Procedure:  GetBottomWindow
File:       \ObjAsm\Code\ObjMem\64\GetBottomWindow.asm
Purpose:    Get the Z order bottom child window HANDLE.
Arguments:  Arg1: Parent HWND.
Return:     eax =  Z order bottom child window HANDLE.


Procedure:  GetDlgBaseUnits
File:       \ObjAsm\Code\ObjMem\64\GetDlgBaseUnits.asm
Purpose:    Return the Dialog Base Units.
Arguments:  Arg1: Dialog DC.
Return:     eax = X DBU.
            ecx = Y DBU.


Procedure:  GetExceptionStrA
File:       \ObjAsm\Code\ObjMem\64\GetExceptionStrA.asm
Purpose:    Translate an exception code to an ANSI string.
Arguments:  Arg1: Exception code.
Return:     rax → ANSI string.


Procedure:  GetExceptionStrW
File:       \ObjAsm\Code\ObjMem\64\GetExceptionStrW.asm
Purpose:    Translate an exception code to a WIDE string.
Arguments:  Arg1: Exception code.
Return:     rax → WIDE string.


Procedure:  GetFileHashA
File:       \ObjAsm\Code\ObjMem\64\GetFileHashA.asm
Purpose:    Compute the hash value from the content of a file.
Arguments:  Arg1: → Hash return value
            Arg2: → ANSI file name.
            Arg3: Hash type.
Return:     eax = 0 if succeeded.
Links:      http://www.masm32.com/board/index.php?topic=4322.msg32297#msg32297
Notes:      Original translation from MSDN library by Edgar Hansen
            It requires a fully qualified path to a file to generate a hash for and a pointer
            to a WIDE string buffer to hold the resulting hash in HEX (16 BYTEs for MDx, 20 BYTEs
            for SHAx) and an algorithm ID, for MD5 set dHashType to GFH_MD5.
            See ObjMem.inc GFH_xxx.


Procedure:  GetFileHashW
File:       \ObjAsm\Code\ObjMem\64\GetFileHashW.asm
Purpose:    Compute the hash value from the content of a file.
Arguments:  Arg1: → Hash return value
            Arg2: → WIDE file name.
            Arg3: Hash type.
Return:     eax = 0 if succeeded.
Links:      http://www.masm32.com/board/index.php?topic=4322.msg32297#msg32297
Notes:      Original translation from MSDN library by Edgar Hansen
            It requires a fully qualified path to a file to generate a hash for and a pointer
            to a WIDE string buffer to hold the resulting hash in HEX (16 BYTEs for MDx, 20 BYTEs
            for SHAx) and an algorithm ID, for MD5 set dHashType to GFH_MD5.
```

```
            See ObjMem.inc GFH_xxx.


Procedure:  GetFileLinesA
File:       \ObjAsm\Code\ObjMem\64\GetFileLinesA.asm
Purpose:    Return an array of line ending offsets of an ANSI text file.
Arguments:  Arg1: File HANDLE.
Return:     eax = Number of lines.
            rcx → Mem block containing an array of DWORD offsets.
            The user must dispose it using MemFree.

Notes:      - Lines must be terminated with the ANSI char sequence 13, 10 (CRLF).
            - The last line may not terminate with a CRLF.


Procedure:  GetLogProcCount
File:       \ObjAsm\Code\ObjMem\64\GetLogProcCount.asm
Purpose:    Return the number of logical CPUs on the current system.
Arguments:  None
Return:     eax = Number of logical processors.


Procedure:  GetObjectID
File:       \ObjAsm\Code\ObjMem\64\GetObjectID.asm
Purpose:    Retrieve the type ID of an object instance.
Arguments:  Arg1: → Object instance.
Return:     eax = Object class ID.


Procedure:  GetObjectTemplate
File:       \ObjAsm\Code\ObjMem\64\GetObjectTemplate.asm
Purpose:    Get the template address of an object type ID.
Arguments:  Arg1: Object type ID.
Return:     rax → Object template or NULL if not found.
            ecx = Object template size or zero if not found.


Procedure:  GetPrevInstanceA
File:       \ObjAsm\Code\ObjMem\64\GetPrevInstanceA.asm
Purpose:    Return a HANDLE to a previously running instance of an application.
Arguments:  Arg1: → ANSI application name.
            Arg2: → ANSI class name.
Return:     rax = Window HANDLE of the application instance or zero if failed.


Procedure:  GetPrevInstanceW
File:       \ObjAsm\Code\ObjMem\64\GetPrevInstanceW.asm
Purpose:    Return a handle to a previously running instance of an application.
Arguments:  Arg1: → WIDE application name.
            Arg2: → WIDE class name.
Return:     rax = Window HANDLE of the application instance or zero if failed.


Procedure:  GetRawClientRect
File:       \ObjAsm\Code\ObjMem\64\GetRawClientRect.asm
Purpose:    Calculate the window client RECT including scrollbars, but without the room needed
            for the menubar
Arguments:  Arg1: Window HANDLE
            Arg2: → RECT.
Return:     Nothing.


Procedure:  GUID2BStr
File:       \ObjAsm\Code\ObjMem\64\GUID2BStr.asm
Purpose:    Convert a GUID to a BStr.
Arguments:  Arg1: → Destination BStr Buffer. It must hold at least
            36 characters plus a terminating zero.
            Arg2: → GUID.
Return:     Nothing.


Procedure:  GUID2StrA
File:       \ObjAsm\Code\ObjMem\64\GUID2StrA.asm
Purpose:    Convert a GUID to an ANSI string.
Arguments:  Arg1: → Destination ANSI string buffer.
            It must hold at least 36 characters plus a ZTC (= 37 BYTEs).
            Arg2: → GUID.
Return:     Nothing.


Procedure:  GUID2StrW
```

```
File:       \ObjAsm\Code\ObjMem\64\GUID2StrW.asm
Purpose:    Convert a GUID to a WIDE string.
Arguments:  Arg1: → Destination WIDE string Buffer.
            It must hold at least 36 characters plus a ZTC (= 74 BYTEs).
            Arg2: → GUID.
Return:     Nothing.


Procedure:  hex2dwordA
File:       \ObjAsm\Code\ObjMem\64\hex2dwordA.asm
Purpose:    Load an ANSI string hexadecimal representation of a DWORD.
Arguments:  Arg1: → ANSI hexadecimal string with 8 characters.
Return:     eax = DWORD.


Procedure:  hex2dwordW
File:       \ObjAsm\Code\ObjMem\64\hex2dwordW.asm
Purpose:    Load a WIDE string hexadecimal representation of a DWORD.
Arguments:  Arg1: → WIDE hex string with 8 characters.
Return:     eax = DWORD.


Procedure:  hex2qwordA
File:       \ObjAsm\Code\ObjMem\64\hex2qwordA.asm
Purpose:    Load an ANSI string hexadecimal representation of a QWORD.
Arguments:  Arg1: → ANSI hexadecimal string with 16 characters.
Return:     rax = QWORD.


Procedure:  hex2qwordW
File:       \ObjAsm\Code\ObjMem\64\hex2qwordW.asm
Purpose:    Load a WIDE string hexadecimal representation of a QWORD.
Arguments:  Arg1: → WIDE hexadecimal string with 16 characters.
Return:     rax = QWORD.


Procedure:  IsAdmin
File:       \ObjAsm\Code\ObjMem\64\IsAdmin.asm
Purpose:    Check if the current user has administrator rights.
Arguments:  None.
Return:     rax = TRUE or FALSE.


Procedure:  IsGUIDEqual
File:       \ObjAsm\Code\ObjMem\64\IsGUIDEqual.asm
Purpose:    Compare 2 GUIDs.
Arguments:  Arg1: → GUID1
            Arg2: → GUID2.
Return:     rax = TRUE if they are equal, otherwise FALSE.


Procedure:  IsHardwareFeaturePresent
File:       \ObjAsm\Code\ObjMem\64\IsHardwareFeaturePresent.asm
Purpose:    Check if a CPU hardware feature is present on the system.
Arguments:  Arg1: CPUID feature ID.
Return:     rax = TRUE or FALSE.


Procedure:  IsPntInRect
File:       \ObjAsm\Code\ObjMem\64\IsPntInRect.asm
Purpose:    Check if a point is within a rect.
Arguments:  Arg1: → POINT.
            Arg2: → RECT
Return:     rax = TRUE or FALSE.


Procedure:  IsProcessElevated
File:       \ObjAsm\Code\ObjMem\64\IsProcessElevated.asm
Purpose:    Check if the current process has elevated privileges.
Arguments:  Arg: Process HANDLE.
Return:     eax = TRUE or FALSE.
            Example: invoke GetCurrentProcess
            invoke IsProcessElevated, xax


Procedure:  IsScrollBarVisible
File:       \ObjAsm\Code\ObjMem\64\IsScrollBarVisible.asm
Purpose:    Determine if a Scrollbar is currently visible.
Arguments:  Arg1: Main window HANDLE that the scrollbar belongs to.
            Arg2: Scrollbar type [SB_HORZ or SB_VERT].
Return:     eax = TRUE if the scrollbar is visible, otherwise FALSE.
```

```
Procedure:  IsWinNT
File:       \ObjAsm\Code\ObjMem\64\IsWinNT.asm
Purpose:    Detect if the OS is Windows NT based.
Arguments:  None.
Return:     rax = TRUE if OS is Windows NT based, otherwise FALSE.


Procedure:  LoadCommonControls
File:       \ObjAsm\Code\ObjMem\64\LoadCommonControls.asm
Purpose:    Invoke InitCommonControls with a correctly filled input structure.
Arguments:  Arg1: ICC_COOL_CLASSES, ICC_BAR_CLASSES, ICC_LISTVIEW_CLASSES, ICC_TAB_CLASSES,
            ICC_USEREX_CLASSES, etc.
Return:     Nothing.


Procedure:  Mem2HexA
File:       \ObjAsm\Code\ObjMem\64\Mem2HexA.asm
Purpose:    Convert the memory content into a hex ANSI string representation.
Arguments:  Arg1: → ANSI character buffer.
            Arg2: → Source memory.
            Arg3: Byte count.
Return:     Nothing.


Procedure:  Mem2HexW
File:       \ObjAsm\Code\ObjMem\64\Mem2HexW.asm
Purpose:    Convert the memory content into a hex WIDE string representation.
Arguments:  Arg1: → WIDE character buffer.
            Arg2: → Source memory.
            Arg3: Byte count.
Return:     Nothing.


Procedure:  MemAlloc_UEFI
File:       \ObjAsm\Code\ObjMem\64\MemAlloc_UEFI.asm
Purpose:    Allocate a memory block.
Arguments:  Arg1: Memory block attributes [0, MEM_INIT_ZERO].
            Arg2: Memory block size in BYTEs.
Return:     rax → Memory block or NULL if failed.


Procedure:  MemClone
File:       \ObjAsm\Code\ObjMem\64\MemClone.asm
Purpose:    Copy a memory block from a source to a destination buffer.
            Source and destination must NOT overlap.
            Destination buffer must be at least as large as number of BYTEs to copy, otherwise a
            fault may be triggered.
Arguments:  Arg1: → Destination buffer.
            Arg2: → Source buffer.
            Arg3: Number of BYTEs to copy.
Return:     eax = Number of copied BYTEs.


Procedure:  MemComp
File:       \ObjAsm\Code\ObjMem\64\MemComp.asm
Purpose:    Compare 2 memory blocks.
            Both memory blocks must be at least as large as the maximal number of BYTEs to
            compare,
            otherwise a fault may be triggered.
Arguments:  Arg1: → Memory block 1.
            Arg2: → Memory block 2.
            Arg3: Maximal number of BYTEs to compare.
Return:     If MemBlock1 = MemBlock2, then eax <> 0.
            If MemBlock1 == MemBlock2, then eax = 0.


Procedure:  MemFillB
File:       \ObjAsm\Code\ObjMem\64\MemFillB.asm
Purpose:    Fill a memory block with a given byte value.
            Destination buffer must be at least as large as number of BYTEs to fill, otherwise a
            fault may be triggered.
Arguments:  Arg1: → Destination memory block.
            Arg2: Memory block size in BYTEs.
            Arg3: Byte value to fill.
Return:     Nothing.


Procedure:  MemFillW
File:       \ObjAsm\Code\ObjMem\64\MemFillW.asm
```

```
Purpose:    Fill a memory block with a given word value.
            Destination buffer must be at least as large as number of BYTEs to fill, otherwise a
            fault may be triggered.
Arguments:  Arg1: → Destination memory block.
            Arg2: Memory block size in BYTEs.
            Arg3: Word value to fill with.
Return:     Nothing.


Procedure:  MemFree_UEFI
File:       \ObjAsm\Code\ObjMem\64\MemFree_UEFI.asm
Purpose:    Dispose a memory block.
Arguments:  Arg1: → Memory block.
Return:     rax = EFI_SUCCESS or an UEFI error code.


Procedure:  MemReAlloc_UEFI
File:       \ObjAsm\Code\ObjMem\64\MemReAlloc_UEFI.asm
Purpose:    Shrink or expand a memory block.
Arguments:  Arg1: → Memory block
            Arg2: Memory block size in BYTEs.
            Arg3: New memory block size in BYTEs.
            Arg4: Memory block attributes [0, MEM_INIT_ZERO].
Return:     rax → New memory block.


Procedure:  MemShift
File:       \ObjAsm\Code\ObjMem\64\MemShift.asm
Purpose:    Copy a memory block from a source to a destination buffer.
            Source and destination may overlap.
            Destination buffer must be at least as large as number of BYTEs to shift, otherwise a
            fault may be triggered.
Arguments:  Arg1: → Destination buffer.
            Arg2: → Source buffer.
            Arg3: Number of BYTEs to shift.
Return:     rax = Number of BYTEs shifted.


Procedure:  MemSwap
File:       \ObjAsm\Code\ObjMem\64\MemSwap.asm
Purpose:    Exchange the memory content from a memory buffer to another.
            They must NOT overlap.
            Both buffers must be at least as large as number of BYTEs to exchange, otherwise a
            fault may be triggered.
Arguments:  Arg1: → Memory buffer 1.
            Arg2: → Memory buffer 2.
            Arg3: Number of BYTEs to exchange.
Return:     Nothing.


Procedure:  MemZero
File:       \ObjAsm\Code\ObjMem\64\MemZero.asm
Purpose:    Fill a memory block with zeros. A bit faster than MemFillB.
            The memory buffer must be at least as large as number of BYTEs to zero, otherwise a
            fault may be triggered.
Arguments:  Arg1: → Memory buffer.
            Arg2: Number of BYTEs to zero.
Return:     Nothing.


Procedure:  MoveWindowVisible
File:       \ObjAsm\Code\ObjMem\64\MoveWindowVisible.asm
Purpose:    On a multimonitor system, move a window but remains always in the visible region.
Arguments:  Arg1: HANDLE of the Window to move.
            Arg2: Target X position in pixel.
            Arg3: Target Y position in pixel.
Return:     Nothing.


Procedure:  MsgBoxA
File:       \ObjAsm\Code\ObjMem\64\MsgBoxA.asm
Purpose:    Show a customized MessageBox.
Arguments:  Arg1: Parent HANDLE.
            Arg2: → Markup text.
            Arg3: → Caption text.
            Arg4: Flags.
Return:     eax = Zero if failed, otherwise pressed button ID.
Note:       Caption, text etc. are transferred via a caption string which contains a header and
            the address of a MsgBoxInfo structure in text form.
```

```
Procedure:  MsgBoxW
File:       \ObjAsm\Code\ObjMem\64\MsgBoxW.asm
Purpose:    Show a customized MessageBox.
Arguments:  Arg1: Parent HANDLE.
            Arg2: → Markup text.
            Arg3: → Caption text.
            Arg4: Flags.
Return:     eax = Zero if failed, otherwise pressed button ID.
Note:       Caption, text etc. are transferred via a caption string which contains a header and
            the address of a MsgBoxInfo structure in text form.


Procedure:  NetErr2StrA
File:       \ObjAsm\Code\ObjMem\64\NetErr2StrA.asm
Purpose:    Translate a network error code to an ANSI string.
Arguments:  Arg1: Error code.
            Arg2: → ANSI character buffer.
            Arg3: Buffer size in characters, inclusive ZTC.
Return:     eax = Number CHRA stored in the output buffer, excluding the ZTC.


Procedure:  NetErr2StrW
File:       \ObjAsm\Code\ObjMem\64\NetErr2StrW.asm
Purpose:    Translate a network error code to a WIDE string.
Arguments:  Arg1: Error code.
            Arg2: → WIDE string buffer.
            Arg3: Buffer size in characters, inclusive ZTC.
Return:     eax = Number CHRW stored in the output buffer, excluding the ZTC.


Procedure:  NewObjInst
File:       \ObjAsm\Code\ObjMem\64\NewObjInst.asm
Purpose:    Create an object instance from an object ID.
Arguments:  Arg1: Object ID.
Return:     rax → New object instance or NULL if failed.


Procedure:  NewObjInst_UEFI
File:       \ObjAsm\Code\ObjMem\64\NewObjInst_UEFI.asm
Purpose:    Create an object instance from an object ID.
Arguments:  Arg1: Object ID.
Return:     rax → New object instance or NULL if failed.


Procedure:  ParseA
File:       \ObjAsm\Code\ObjMem\64\ParseA.asm
Purpose:    Extract a comma separated substring from a source string.
Arguments:  Arg1: → Destination buffer. Must be large enough to hold the ANSI substring.
            Arg2: → Source ANSI string.
            Arg3: Zero based index of the requested ANSI substring.
Return:     eax = 1: success.
            2: insufficient number of components.
            3: non matching quotation marks.
            4: empty quote.


Procedure:  ParseW
File:       \ObjAsm\Code\ObjMem\64\ParseW.asm
Purpose:    Extract a comma separated substring from a source string.
Arguments:  Arg1: → Destination buffer. Must be large enough to hold the WIDE substring.
            Arg2: → Source WIDE string.
            Arg3: Zero based index of the requested WIDE substring.
Return:     eax = 1: success.
            2: insufficient number of components.
            3: non matching quotation marks.
            4: empty quote.


Procedure:  PdfViewA
File:       \ObjAsm\Code\ObjMem\64\PdfViewA.asm
Purpose:    Display a PDF document on a named destination.
Arguments:  Arg1: Parent HANDLE.
            Arg2: → PDF document.
            Arg3: → Destination.
Return:     rax = HINSTANCE. See ShellExecute return values.
            A value greater than 32 indicates success.


Procedure:  PdfViewW
File:       \ObjAsm\Code\ObjMem\64\PdfViewW.asm
Purpose:    Display a PDF document on a named destination.
```

```
Arguments:  Arg1: Parent HANDLE.
            Arg2: → PDF document.
            Arg3: → Destination.
Return:     rax = HINSTANCE. See ShellExecute return values.
            A value greater than 32 indicates success.


Procedure:  qword2binA
File:       \ObjAsm\Code\ObjMem\64\qword2binA.asm
Purpose:    Convert a QWORD to its binary ANSI string representation.
Arguments:  Arg1: → Destination buffer.
            Arg2: QWORD value.
Return:     Nothing.
Notes:      The destination buffer must be at least 65 BYTEs large to allocate the output string
            (64 character BYTEs + ZTC = 65 BYTEs).


Procedure:  qword2binW
File:       \ObjAsm\Code\ObjMem\64\qword2binW.asm
Purpose:    Convert a QWORD to its binary WIDE string representation.
Arguments:  Arg1: → Destination buffer.
            Arg2: QWORD value.
Return:     Nothing.
Notes:      The destination buffer must be at least 130 BYTEs large to allocate the output string
            (64 character WORDs + ZTC = 130 BYTEs).


Procedure:  qword2hexA
File:       \ObjAsm\Code\ObjMem\64\qword2hexA.asm
Purpose:    Convert a QWORD to its hexadecimal ANSI string representation.
Arguments:  Arg1: → Destination ANSI string buffer.
            Arg2: QWORD value.
Return:     Nothing.
Note:       The destination buffer must be at least 17 BYTEs large to allocate the output string
            (16 character BYTEs + ZTC = 17 BYTEs).


Procedure:  qword2hexW
File:       \ObjAsm\Code\ObjMem\64\qword2hexW.asm
Purpose:    Convert a QWORD to its hexadecimal WIDE string representation.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: QWORD value.
Return:     Nothing.
Notes:      The destination buffer must be at least 34 BYTEs large to allocate the output string
            (16 character WORDs + ZTC = 34 BYTEs).


Procedure:  RadixSortF32
File:       \ObjAsm\Code\ObjMem\64\RadixSortF32.asm
Purpose:    Ascending sort of an array of single precision floats (REAL4) using a modified
            "4 passes radix sort" algorithm.
Arguments:  Arg1: → Array of single precision floats (REAL4).
            Arg2: Number of single precision floats contained in the array.
            Arg3: → Memory used for the sorting process or NULL. The buffer size must be at least
            the size of the input array. If NULL, a memory chunk is allocated automatically.
Return:     eax = TRUE if succeeded, otherwise FALSE.
Notes:      - For short arrays, the shadow array can be placed onto the stack, saving the
            expensive memory allocation/deallocation API calls. To achieve this, the proc
            must be modified and stack probing must be included.
Links:      - http://www.codercorner.com/RadixSortRevisited.htm
            - http://en.wikipedia.org/wiki/Radix_sort


Procedure:  RadixSortF64
File:       \ObjAsm\Code\ObjMem\64\RadixSortF64.asm
Purpose:    Ascending sort of an array of double precision floats (REAL8) using a modified
            "8 passes radix sort" algorithm.
Arguments:  Arg1: → Array of double precision floats (REAL8).
            Arg2: Number of double precision floats contained in the array.
            Arg3: → Memory used for the sorting process or NULL. The buffer size must be at least
            the size of the input array. If NULL, a memory chunk is allocated automatically.
Return:     eax = TRUE if succeeded, otherwise FALSE.
Notes:      - For short arrays, the shadow array can be placed onto the stack, saving the
            expensive memory allocation/deallocation API calls. To achieve this, the proc
            must be modified and stack probing must be included.
Links:      - http://www.codercorner.com/RadixSortRevisited.htm
            - http://en.wikipedia.org/wiki/Radix_sort


Procedure:  RadixSortI32
File:       \ObjAsm\Code\ObjMem\64\RadixSortI32.asm
```

```
Purpose:    Ascending sort of an array of SDWORDs using a modified "4 passes radix sort"
            algorithm.
Arguments:  Arg1: → Array of SDWORDs.
            Arg2: Number of SDWORDs contained in the array.
            Arg3: → Memory used for the sorting process or NULL. The buffer size must be at least
            the size of the input array. If NULL, a memory chunk is allocated automatically.
Return:     eax = TRUE if succeeded, otherwise FALSE.
Notes:      - Original code from r22.
            http://www.asmcommunity.net/board/index.php?topic=24563.0
            - For short arrays, the shadow array can be placed onto the stack, saving the
            expensive memory allocation/deallocation API calls. To achieve this, the proc
            must be modified and stack probing must be included.
Links:      - http://www.codercorner.com/RadixSortRevisited.htm
            - http://en.wikipedia.org/wiki/Radix_sort


Procedure:  RadixSortPtrF32
File:       \ObjAsm\Code\ObjMem\64\RadixSortPtrF32.asm
Purpose:    Ascending sort of an array of POINTERs to structures containing a
            single precision float (REAL4) key using a modified "4 passes radix sort" algorithm.
Arguments:  Arg1: → Array of POINTERs.
            Arg2: Number of POINTERs contained in the array.
            Arg3: offset of the REAL4 key within the hosting structure.
            Arg4: → Memory used for the sorting process or NULL. The buffer size must be at least
            the size of the input array. If NULL, a memory chunk is allocated automatically.
Return:     eax = TRUE if succeeded, otherwise FALSE.
Notes:      - For short arrays, the shadow array can be placed onto the stack, saving the
            expensive memory allocation/deallocation API calls. To achieve this, the proc
            must be modified and stack probing must be included.
Links:      - http://www.codercorner.com/RadixSortRevisited.htm
            - http://en.wikipedia.org/wiki/Radix_sort


Procedure:  RadixSortPtrF64
File:       \ObjAsm\Code\ObjMem\64\RadixSortPtrF64.asm
Purpose:    Ascending sort of an array of POINTERs to structures containing a
            double precision float (REAL8) key using a modified "8 passes radix sort" algorithm.
Arguments:  Arg1: → Array of POINTERs.
            Arg2: Number of POINTERs contained in the array.
            Arg3: offset of the REAL8 key within the hosting structure.
            Arg4: → Memory used for the sorting process or NULL. The buffer size must be at least
            the size of the input array. If NULL, a memory chunk is allocated automatically.
Return:     eax = TRUE if succeeded, otherwise FALSE.
Notes:      - For short arrays, the shadow array can be placed onto the stack, saving the
            expensive memory allocation/deallocation API calls. To achieve this, the proc
            must be modified and stack probing must be included.
Links:      - http://www.codercorner.com/RadixSortRevisited.htm
            - http://en.wikipedia.org/wiki/Radix_sort


Procedure:  RadixSortPtrI32
File:       \ObjAsm\Code\ObjMem\64\RadixSortPtrI32.asm
Purpose:    Ascending sort of an array of POINTERs to structures containing a SDWORD key using a
            modified "4 passes radix sort" algorithm.
Arguments:  Arg1: → Array of POINTERs.
            Arg2: Number of POINTERs contained in the array.
            Arg3: offset of the SDWORD key within the hosting structure.
            Arg4: → Memory used for the sorting process or NULL. The buffer size must be at least
            the size of the input array. If NULL, a memory chunk is allocated automatically.
Return:     eax = TRUE if succeeded, otherwise FALSE.
Notes:      - Original code from r22.
            http://www.asmcommunity.net/board/index.php?topic=24563.0
            - For short arrays, the shadow array can be placed onto the stack, saving the
            expensive memory allocation/deallocation API calls. To achieve this, the proc
            must be modified and stack probing must be included.
Links:      - http://www.codercorner.com/RadixSortRevisited.htm
            - http://en.wikipedia.org/wiki/Radix_sort


Procedure:  RadixSortPtrUI32
File:       \ObjAsm\Code\ObjMem\64\RadixSortPtrUI32.asm
Purpose:    Ascending sort of a POINTER array to structures containing a DWORD key using the
            "4 passes radix sort" algorithm.
Arguments:  Arg1: → Array of POINTERs.
            Arg2: Number of POINTERs contained in the array.
            Arg3: offset of the DWORD key within the hosting structure.
            Arg4: → Memory used for the sorting process or NULL. The buffer size must be at least
            the size of the input array. If NULL, a memory chunk is allocated automatically.
Return:     eax = TRUE if succeeded, otherwise FALSE.
Notes:      - Original code from r22.
            http://www.asmcommunity.net/board/index.php?topic=24563.0
```

```
            - For short arrays, the shadow array can be placed onto the stack, saving the
            expensive memory allocation/deallocation API calls. To achieve this, the proc
            must be modified and stack probing must be included.
Links:      - http://www.codercorner.com/RadixSortRevisited.htm
            - http://en.wikipedia.org/wiki/Radix_sort


Procedure:  RadixSortUI32
File:       \ObjAsm\Code\ObjMem\64\RadixSortUI32.asm
Purpose:    Ascending sort of an array of DWORDs using the "4 passes radix sort" algorithm.
Arguments:  Arg1: → Array of DWORDs.
            Arg2: Number of DWORDs contained in the array.
            Arg3: → Memory used for the sorting process or NULL. The buffer size must be at least
            the size of the input array. If NULL, a memory chunk is allocated automatically.
Return:     eax = TRUE if succeeded, otherwise FALSE.
Notes:      - Original code from r22.
            http://www.asmcommunity.net/board/index.php?topic=24563.0
            - For short arrays, the shadow array can be placed onto the stack, saving the
            expensive memory allocation/deallocation API calls. To achieve this, the proc
            must be modified and stack probing must be included.
Links:      - http://www.codercorner.com/RadixSortRevisited.htm
            - http://en.wikipedia.org/wiki/Radix_sort


Procedure:  Random32
File:       \ObjAsm\Code\ObjMem\64\Random32.asm
Purpose:    Generate a random 32 bit number in a given range [0..Limit-1].
            Park Miller random number algorithm. Written by Jaymeson Trudgen (NaN) and optimized
            by Rickey Bowers Jr. (bitRAKE).
Arguments:  Arg1: Range limit (max. = 07FFFFFFFh).
Return:     eax = Random number in the range [0..Limit-1].


Procedure:  Real4ToHalf
File:       \ObjAsm\Code\ObjMem\64\Real4ToHalf.asm
Purpose:    Convert a REAL4 to an HALF.
Arguments:  Arg1: REAL4 value.
Return:     ax = HALF.
Note:       alternative code using VCVTPS2PH:
            movss xmm0, r4Value
            VCVTPS2PH xmm1, xmm0, 0
            movd eax, xmm1


Procedure:  RGB24To16ColorIndex
File:       \ObjAsm\Code\ObjMem\64\RGB24To16ColorIndex.asm
Purpose:    Map a 24 bit RGB color to a 16 color palette index.
Arguments:  Arg1: RGB color.
Return:     eax = Palette index.


Procedure:  sdword2decA
File:       \ObjAsm\Code\ObjMem\64\sdword2decA.asm
Purpose:    Convert a signed DWORD to its decimal ANSI string representation.
Arguments:  Arg1: → Destination ANSI string buffer.
            Arg2: SDWORD value.
Return:     eax = Number of BYTEs copied to the destination buffer, including the ZTC.
Note:       The destination buffer must be at least 12 BYTEs large to allocate the output string
            (Sign + 10 ANSI characters + ZTC = 12 BYTEs).


Procedure:  sdword2decW
File:       \ObjAsm\Code\ObjMem\64\sdword2decW.asm
Purpose:    Convert a signed DWORD to its decimal WIDE string representation.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: SDWORD value.
Return:     eax = Number of BYTEs copied to the destination buffer, including the ZTC.
Note:       The destination buffer must be at least 24 BYTEs large to allocate the output string
            (Sign + 10 WIDE characters + ZTC = 24 BYTEs).


Procedure:  SendChildrenMessage
File:       \ObjAsm\Code\ObjMem\64\SendChildrenMessage.asm
Purpose:    Callback procedure for EnumChildWindows that sends a message to a child window.
Arguments:  Arg1: Child window HANDLE.
            Arg2: → CHILD_MSG structure.
Return:     eax = always TRUE (continue the enumeration).


Procedure:  SetClientSize
File:       \ObjAsm\Code\ObjMem\64\SetClientSize.asm
```

```
Purpose:    Set the client window size.
Arguments:  Arg1: Target window handle.
            Arg2: Client area width in pixel.
            Arg3: Client area height in pixel.
Return:     Nothing.


Procedure:  SetPrivilegeTokenA
File:       \ObjAsm\Code\ObjMem\64\SetPrivilegeTokenA.asm
Purpose:    Enable privilege tokens.
Arguments:  Arg1: Process handle.
            Arg2: → Privilege name (ANSI string).
            Arg3: Eanble = TRUE, disable = FALSE.
Return:     eax = Zero if failed.


Procedure:  SetPrivilegeTokenW
File:       \ObjAsm\Code\ObjMem\64\SetPrivilegeTokenW.asm
Purpose:    Enable privilege tokens.
Arguments:  Arg1: Process handle.
            Arg2: → Privilege name (ANSI string).
            Arg3: Eanble = TRUE, disable = FALSE
Return:     eax = Zero if failed.


Procedure:  SetShellAssociationA
File:       \ObjAsm\Code\ObjMem\64\SetShellAssociationA.asm
Purpose:    Set association for a file extension.
Arguments:  Arg1: TRUE = system wide association, FALSE = user account only.
            Arg2: → File extension (without dot).
            Arg3: → Verb ("open", "print", "play", "edit", etc.). This verb is displayed
            in the explorer context menu of a file with this extension.
            Arg4: → Application to associate with (full path).
            Arg5: → Application arguments, usually $OfsCStr("%1").
Return:     eax = HRESULT.
Note:       dGlobal = TRUE requires adminitrative rights.


Procedure:  SetShellAssociationW
File:       \ObjAsm\Code\ObjMem\64\SetShellAssociationW.asm
Purpose:    Set association for a file extension.
Arguments:  Arg1: TRUE = system wide association, FALSE = user account only.
            Arg2: → File extension (without dot).
            Arg3: → Verb ("open", "print", "play", "edit", etc.). This verb is displayed
            in the explorer context menu of a file with this extension.
            Arg4: → Application to associate with (full path).
            Arg5: → Application arguments, usually $OfsCStr("%1").
Return:     eax = HRESULT.
Note:       dGlobal = TRUE requires adminitrative rights.


Procedure:  SetShellPerceivedTypeA
File:       \ObjAsm\Code\ObjMem\64\SetShellPerceivedTypeA.asm
Purpose:    Set shell perception of a file type.
Arguments:  Arg1: TRUE = system wide perseption, FALSE = user account only.
            Arg2: → File extension (without dot).
            Arg3: → Type (Folder, Text, Image, Audio, Video, Compressed, Document, System,
            Application, Gamemedia, Contacts)
Return:     eax = HRESULT.
Note:       To retrieve the perceived type use the AssocGetPerceivedType API.
            dGlobal = TRUE requires adminitrative rights.


Procedure:  SetShellPerceivedTypeW
File:       \ObjAsm\Code\ObjMem\64\SetShellPerceivedTypeW.asm
Purpose:    Set shell perception of a file type.
Arguments:  Arg1: TRUE = system wide perseption, FALSE = user account only.
            Arg2: → File extension (without dot).
            Arg3: → Type (Folder, Text, Image, Audio, Video, Compressed, Document, System,
            Application, Gamemedia, Contacts)
Return:     eax = HRESULT.
Note:       To retrieve the perceived type use the AssocGetPerceivedType API.
            dGlobal = TRUE requires adminitrative rights.


Procedure:  ShortToLongPathNameA
File:       \ObjAsm\Code\ObjMem\64\ShortToLongPathNameA.asm
Purpose:    Allocate a new ANSI string containing the long path of a short path string.
Arguments:  Arg1: → Short path ANSI string.
Return:     rax → Long path ANSI string or NULL if failed.
```

```
Procedure:  SLR_Calc_AB_DW
File:       \ObjAsm\Code\ObjMem\64\SLR_Calc_AB_DW.asm
Purpose:    Calculate the Slope (A) and Intercept (B) values of the linear equation y = A*x + B
            that minimize mean squared error (MSE) of a DWORD array.
Arguments:  Arg1: → SLR_DATA structure.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
            If an FPU exception occurs, the results are NaN.
Formulas:   A = (XY*N-X*Y)/Q
            B = (Y-A*X)/N


Procedure:  SLR_Calc_AB_MSE_DW
File:       \ObjAsm\Code\ObjMem\64\SLR_Calc_AB_MSE_DW.asm
Purpose:    Calculate the Slope (A) and Intercept (B) values of the linear equation y = A*x + B
            that minimize mean squared error (MSE) and the MSE value of a DWORD array.
Arguments:  Arg1: → SLR_DATA structure.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
            If an FPU exception occurs, the results are NaN.
Formulas:   A = (XY*N-X*Y)/Q
            B = (Y-A*X)/N
            MSE = (Y2 - 2*A*XY - 2*B*Y + A^2*X2 + 2*A*B*X)/N + B^2


Procedure:  SLR_Calc_AB_MSE_QW
File:       \ObjAsm\Code\ObjMem\64\SLR_Calc_AB_MSE_QW.asm
Purpose:    Calculate the Slope (A) and Intercept (B) values of the linear equation y = A*x + B
            that minimize mean squared error (MSE) and the MSE value of a QWORD array.
Arguments:  Arg1: → SLR_DATA structure.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
            If an FPU exception occurs, the results are NaN.
Formulas:   A = (XY*N-X*Y)/Q
            B = (Y-A*X)/N
            MSE = (Y2 - 2*A*XY - 2*B*Y + A^2*X2 + 2*A*B*X)/N + B^2


Procedure:  SLR_Calc_AB_MSE_R4
File:       \ObjAsm\Code\ObjMem\64\SLR_Calc_AB_MSE_R4.asm
Purpose:    Calculate the Slope (A) and Intercept (B) values of the linear equation y = A*x + B
            that minimize mean squared error (MSE) and the MSE value of a REAL4 array.
Arguments:  Arg1: → SLR_DATA structure.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
            If an FPU exception occurs, the results are NaN.
Formulas:   A = (XY*N-X*Y)/Q
            B = (Y-A*X)/N
            MSE = (Y2 - 2*A*XY - 2*B*Y + A^2*X2 + 2*A*B*X)/N + B^2


Procedure:  SLR_Calc_AB_MSE_R8
File:       \ObjAsm\Code\ObjMem\64\SLR_Calc_AB_MSE_R8.asm
Purpose:    Calculate the Slope (A) and Intercept (B) values of the linear equation y = A*x + B
            that minimize mean squared error (MSE) and the MSE value of a REAL8 array.
Arguments:  Arg1: → SLR_DATA structure.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
```

```
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
            If an FPU exception occurs, the results are NaN.
Formulas:   A = (XY*N-X*Y)/Q
            B = (Y-A*X)/N
            MSE = (Y2 - 2*A*XY - 2*B*Y + A^2*X2 + 2*A*B*X)/N + B^2


Procedure:  SLR_Calc_AB_QW
File:       \ObjAsm\Code\ObjMem\64\SLR_Calc_AB_QW.asm
Purpose:    Calculate the Slope (A) and Intercept (B) values of the linear equation y = A*x + B
            that minimize mean squared error (MSE) of a QWORD array.
Arguments:  Arg1: → SLR_DATA structure.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
            If an FPU exception occurs, the results are NaN.
Formulas:   A = (XY*N-X*Y)/Q
            B = (Y-A*X)/N


Procedure:  SLR_Calc_AB_R4
File:       \ObjAsm\Code\ObjMem\64\SLR_Calc_AB_R4.asm
Purpose:    Calculate the Slope (A) and Intercept (B) values of the linear equation y = A*x + B
            that minimize mean squared error (MSE) of a REAL4 array.
Arguments:  Arg1: → SLR_DATA structure.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
            If an FPU exception occurs, the results are NaN.
Formulas:   A = (XY*N-X*Y)/Q
            B = (Y-A*X)/N


Procedure:  SLR_Calc_AB_R8
File:       \ObjAsm\Code\ObjMem\64\SLR_Calc_AB_R8.asm
Purpose:    Calculate the Slope (A) and Intercept (B) values of the linear equation y = A*x + B
            that minimize mean squared error (MSE) of a REAL8 array.
Arguments:  Arg1: → SLR_DATA structure.
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
            If an FPU exception occurs, the results are NaN.
Formulas:   A = (XY*N-X*Y)/Q
            B = (Y-A*X)/N


Procedure:  SLR_Init
File:       \ObjAsm\Code\ObjMem\64\SLR_Init.asm
Purpose:    Calculate in advance the invariant coefficients of a Simple Linear Regression (X, X2,
            Q)
Arguments:  Arg1: → SLR_DATA structure
Return:     eax = TRUE is succeeded, otherwise FALSE.
Links:      https://en.wikipedia.org/wiki/1_%2B_2_%2B_3_%2B_4_%2B_%E2%8B%AF
            https://mathschallenge.net/library/number/sum_of_squares
            https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-
            c7dde9a26b93/
Note:       Since X ranges from [0..N-1], the known formulas have to be adjusted accordingly by
            replacing N with N-1.
Formulas:   X = N*(N-1)/2
            X2 = X*(2*N - 1)/3
            Q = N^2*(N^2-1)/12


Procedure:  sqword2decA
File:       \ObjAsm\Code\ObjMem\64\sqword2decA.asm
Purpose:    Convert a signed QWORD to its decimal ANSI string representation.
```

```
Arguments:  Arg1: → Destination ANSI string buffer.
            Arg2: SQWORD value.
Return:     eax = Number of BYTEs copied to the destination buffer, including the ZTC.
Note:       The destination buffer must be at least 21 BYTEs large to allocate the output string
            (Sign + 19 ANSI characters + ZTC = 21 BYTEs).


Procedure:  sqword2decW
File:       \ObjAsm\Code\ObjMem\64\sqword2decW.asm
Purpose:    Convert a signed SQWORD to its decimal WIDE string representation.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: SQWORD value.
Return:     eax = Number of BYTEs copied to the destination buffer, including the ZTC.
Note:       The destination buffer must be at least 42 BYTEs large to allocate the output string
            (Sign + 19 WIDE characters + ZTC = 42 BYTEs).


Procedure:  St0ToStrA
File:       \ObjAsm\Code\ObjMem\64\St0ToStrA.asm
Purpose:    Create an ANSI string representation of the content of the st(0) FPU register.
Arguments:  Arg1: → Destination buffer.
            Arg2: Minimal number of places from the start of string up to the decimal point.
            (f_NOR only)
            Arg3: Number of decimal places after the decimal point.
            Arg4: Format flag (f_NOR, f_SCI, f_TRIM, f_ALIGNED) defined in fMath.inc
Return:     eax = Result code f_OK, f_ERROR, f_NAN, ...
Notes:      - Based on the work of Raymond Filiatreault (FpuLib).
            - st4, st5, st6 and st7 must be empty.
            - f_NOR: regular output format
            - f_SCI: Scientific output format
            - f_TRIM: Trim zeros on the right
            - f_ALIGN: Add a heading space to align the output with other negative numbers
            - f_PLUS: like f_ALIGN, but using a + character.


Procedure:  St0ToStrW
File:       \ObjAsm\Code\ObjMem\64\St0ToStrW.asm
Purpose:    Create a WIDE string representation of the content of the st(0) FPU register.
Arguments:  Arg1: → Destination buffer.
            Arg2: Minimal number of places from the start of string up to the decimal point.
            (f_NOR only)
            Arg3: Number of decimal places after the decimal point.
            Arg4: Format flag (f_NOR, f_SCI, f_TRIM, f_ALIGNED) defined in fMath.inc
Return:     eax = Result code f_OK, f_ERROR, f_NAN, ...
Notes:      - Based on the work of Raymond Filiatreault (FpuLib).
            - st4, st5, st6 and st7 must be empty.
            - f_NOR: regular output format
            - f_SCI: Scientific output format
            - f_TRIM: Trim zeros on the right
            - f_ALIGN: Add a heading space to align the output with other negative numbers
            - f_PLUS: like f_ALIGN, but using a + character.


Procedure:  StkGrdCallback
File:       \ObjAsm\Code\ObjMem\64\StkGrdCallback.asm
Purpose:    StackGuard notification callback procedure.
            It is called when StackGuard is active and a stack overrun was detected.
            It displays a MessageBox asking to abort. If yes, then Exitprocess is called
            immediately.
Arguments:  None.
Return:     Nothing.


Procedure:  Str2BStrA
File:       \ObjAsm\Code\ObjMem\64\Str2BStrA.asm
Purpose:    Convert a ANSI string into a BStr.
Arguments:  Arg1: → Destination BStr buffer = Buffer address + sizeof(DWORD).
            Arg2: → Source ANSI string.
Return:     eax = String length.


Procedure:  Str2BStrW
File:       \ObjAsm\Code\ObjMem\64\Str2BStrW.asm
Purpose:    Convert a WIDE string into a BStr.
Arguments:  Arg1: → Destination BStr buffer = Buffer address + sizeof(DWORD).
            Arg2: → Source WIDE string.
Return:     Nothing.


Procedure:  StrA2StrW
File:       \ObjAsm\Code\ObjMem\64\StrA2StrW.asm
```

```
Purpose:    Convert a ANSI string into a WIDE string.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: → Source ANSI string.
Return:     Nothing.


Procedure:  StrAllocA
File:       \ObjAsm\Code\ObjMem\64\StrAllocA.asm
Purpose:    Allocate space for a string with n characters.
Arguments:  Arg1: Character count without the ZTC.
Return:     rax → New allocated string or NULL if failed.


Procedure:  StrAllocA_UEFI
File:       \ObjAsm\Code\ObjMem\64\StrAllocA_UEFI.asm
Purpose:    Allocate space for a string with n characters.
Arguments:  Arg1: Character count without the ZTC.
Return:     rax → New allocated string or NULL if failed.


Procedure:  StrAllocW
File:       \ObjAsm\Code\ObjMem\64\StrAllocW.asm
Purpose:    Allocate space for a string with n characters.
Arguments:  Arg1: Character count without the ZTC.
Return:     rax → New allocated string or NULL if failed.


Procedure:  StrAllocW_UEFI
File:       \ObjAsm\Code\ObjMem\64\StrAllocW_UEFI.asm
Purpose:    Allocate space for a string with n characters.
Arguments:  Arg1: Character count without the ZTC.
Return:     rax → New allocated string or NULL if failed.


Procedure:  StrCatA
File:       \ObjAsm\Code\ObjMem\64\StrCatA.asm
Purpose:    Concatenate 2 ANSI strings.
Arguments:  Arg1: Destination ANSI buffer.
            Arg2: Source ANSI string.
Return:     Nothing.


Procedure:  StrCatCharA
File:       \ObjAsm\Code\ObjMem\64\StrCatCharA.asm
Purpose:    Append a character to the end of an ANSI string.
Arguments:  Arg1: Destination ANSI buffer.
            Arg2: ANSI character.
Return:     Nothing.


Procedure:  StrCatCharW
File:       \ObjAsm\Code\ObjMem\64\StrCatCharW.asm
Purpose:    Append a character to the end of an WIDE string.
Arguments:  Arg1: Destination ANSI buffer.
            Arg2: WIDE character.
Return:     Nothing.


Procedure:  StrCatW
File:       \ObjAsm\Code\ObjMem\64\StrCatW.asm
Purpose:    Concatenate 2 WIDE strings.
Arguments:  Arg1: Destination WIDE string.
            Arg2: Source WIDE string.
Return:     Nothing.


Procedure:  StrCCatA
File:       \ObjAsm\Code\ObjMem\64\StrCCatA.asm
Purpose:    Concatenate 2 ANSI strings with length limitation.
            The destination string buffer should have at least enough room for the maximum number
            of characters + 1.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
            Arg3: Maximal number of charachters that the destination string can hold including the
            ZTC.
Return:     rax = Number of added BYTEs.


Procedure:  StrCCatCharA
File:       \ObjAsm\Code\ObjMem\64\StrCCatCharA.asm
Purpose:    Append a character to the end of an ANSI string with length limitation.
```

```
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → ANSI character.
            Arg3: Maximal number of characters that fit into the destination buffer.
Return:     Nothing.


Procedure:  StrCCatCharW
File:       \ObjAsm\Code\ObjMem\64\StrCCatCharW.asm
Purpose:    Append a character to the end of a WIDE string with length limitation.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → WIDE character.
            Arg3: Maximal number of characters that fit into the destination buffer.
Return:     Nothing.


Procedure:  StrCCatW
File:       \ObjAsm\Code\ObjMem\64\StrCCatW.asm
Purpose:    Concatenate 2 WIDE strings with length limitation.
            The destination string buffer should have at least enough room for the maximum number
            of characters + 1.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
            Arg3: Maximal number of charachters that the destination string can hold including the
            ZTC.
Return:     rax = Number of added BYTEs.


Procedure:  StrCCompA
File:       \ObjAsm\Code\ObjMem\64\StrCCompA.asm
Purpose:    Compare 2 ANSI strings with case sensitivity up to a maximal number of characters.
Arguments:  Arg1: → ANSI string 1.
            Arg2: → ANSI string 2.
            Arg3: Maximal number of characters to compare.
Return:     If string 1 < string 2, then eax < 0.
            If string 1 = string 2, then eax = 0.
            If string 1 > string 2, then eax > 0.


Procedure:  StrCCompW
File:       \ObjAsm\Code\ObjMem\64\StrCCompW.asm
Purpose:    Compare 2 WIDE strings with case sensitivity up to a maximal number of characters.
Arguments:  Arg1: → WIDE string 1.
            Arg2: → WIDE string 2.
            Arg3: Maximal number of characters to compare.
Return:     If string 1 < string 2, then eax < 0.
            If string 1 = string 2, then eax = 0.
            If string 1 > string 2, then eax > 0.


Procedure:  StrCCopyA
File:       \ObjAsm\Code\ObjMem\64\StrCCopyA.asm
Purpose:    Copy the the source ANSI string with length limitation.
            The destination buffer should be large enough to hold the maximum number of
            characters + 1.
Arguments:  Arg1: → Destination buffer.
            Arg2: → Source ANSI string.
            Arg3: Maximal number of charachters to copy, excluding the ZTC.
Return:     rax = Number of copied BYTEs, including the ZTC.


Procedure:  StrCCopyW
File:       \ObjAsm\Code\ObjMem\64\StrCCopyW.asm
Purpose:    Copy the the source WIDE string with length limitation.
            The destination buffer should be big enough to hold the maximum number of
            characters + 1.
Arguments:  Arg1: → Destination buffer.
            Arg2: → Source WIDE string.
            Arg3: Maximal number of charachters to copy, excluding the ZTC.
Return:     rax = Number of copied BYTEs, including the ZTC.


Procedure:  StrCECatA
File:       \ObjAsm\Code\ObjMem\64\StrCECatA.asm
Purpose:    Concatenate 2 ANSI strings with length limitation and return the ZTC address.
            The destination string buffer should have at least enough room for the maximum
            number of characters + 1.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
            Arg3: Maximal number of charachters that the destination string can hold including the
            ZTC.
Return:     rax → ZTC.
```

```
Procedure:  StrCECatW
File:       \ObjAsm\Code\ObjMem\64\StrCECatW.asm
Purpose:    Concatenate 2 WIDE strings with length limitation and return the ZTC address.
            The destination string buffer should have at least enough room for the maximum
            number of characters + 1.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
            Arg3: Maximal number of charachters that the destination string can hold including the
            ZTC.
Return:     rax → ZTC.


Procedure:  StrCECopyA
File:       \ObjAsm\Code\ObjMem\64\StrCECopyA.asm
Purpose:    Copy the the source ANSI string with length limitation and return the ZTC address.
            The destination buffer should hold the maximum number of characters + 1.
            Source and destination strings may overlap.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
            Arg3: Maximal number of characters not including the ZTC.
Return:     rax → ZTC.


Procedure:  StrCECopyW
File:       \ObjAsm\Code\ObjMem\64\StrCECopyW.asm
Purpose:    Copy the the source WIDE string with length limitation and return the ZTC address.
            The destination buffer should hold the maximum number of characters + 1.
            Source and destination strings may overlap.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
            Arg3: Maximal number of characters not including the ZTC.
Return:     rax → ZTC.


Procedure:  StrCICompA
File:       \ObjAsm\Code\ObjMem\64\StrCICompA.asm
Purpose:    Compare 2 ANSI strings without case sensitivity and length limitation.
Arguments:  Arg1: → ANSI string 1.
            Arg2: → ANSI string 2.
Return:     If string 1 < string 2, then eax < 0.
            If string 1 = string 2, then eax = 0.
            If string 1 > string 2, then eax > 0.


Procedure:  StrCICompW
File:       \ObjAsm\Code\ObjMem\64\StrCICompW.asm
Purpose:    Compare 2 WIDE strings without case sensitivity and length limitation.
Arguments:  Arg1: → WIDE string 1.
            Arg2: → WIDE string 2.
Return:     If string 1 < string 2, then eax < 0.
            If string 1 = string 2, then eax = 0.
            If string 1 > string 2, then eax > 0.


Procedure:  StrCLengthA
File:       \ObjAsm\Code\ObjMem\64\StrCLengthA.asm
Purpose:    Get the character count of the source ANSI string with length limitation.
Arguments:  Arg1: → Source ANSI string.
            Arg3: Maximal character count.
Return:     eax = Limited character count.


Procedure:  StrCLengthW
File:       \ObjAsm\Code\ObjMem\64\StrCLengthW.asm
Purpose:    Get the character count of the source WIDE string with length limitation.
Arguments:  Arg1: → Source WIDE string.
            Arg3: Maximal character count.
Return:     eax = Limited character count.


Procedure:  StrCNewA
File:       \ObjAsm\Code\ObjMem\64\StrCNewA.asm
Purpose:    Allocate a new copy of the source ANSI string with length limitation.
            If the pointer to the source string is NULL or points to an empty string, StrCNewA
            returns NULL and doesn't allocate any heap space. Otherwise, StrCNewA makes a
            duplicate of the source string. The maximal size of the new string is limited to the
            second parameter.
Arguments:  Arg1: → Source ANSI string.
            Arg2: Maximal character count.
```

```
Return:     rax → New ANSI string copy.


Procedure:  StrCNewA_UEFI
File:       \ObjAsm\Code\ObjMem\64\StrCNewA_UEFI.asm
Purpose:    Allocate a new copy of the source ANSI string with length limitation.
            If the pointer to the source string is NULL or points to an empty string, StrCNewA
            returns NULL and doesn't allocate any heap space. Otherwise, StrCNewA makes a
            duplicate of the source string. The maximal size of the new string is limited to the
            second parameter.
Arguments:  Arg1: → Source ANSI string.
            Arg2: Maximal character count.
Return:     rax → New ANSI string copy.


Procedure:  StrCNewW
File:       \ObjAsm\Code\ObjMem\64\StrCNewW.asm
Purpose:    Allocate a new copy of the source WIDE string with length limitation.
            If the pointer to the source string is NULL or points to an empty string, StrCNewW
            returns NULL and doesn't allocate any heap space. Otherwise, StrCNewW makes a
            duplicate of the source string. The maximal size of the new string is limited to the
            second parameter.
Arguments:  Arg1: → Source WIDE string.
            Arg2: Maximal character count.
Return:     rax → New WIDE string copy.


Procedure:  StrCNewW_UEFI
File:       \ObjAsm\Code\ObjMem\64\StrCNewW_UEFI.asm
Purpose:    Allocate a new copy of the source WIDE string with length limitation.
            If the pointer to the source string is NULL or points to an empty string, StrCNewW
            returns NULL and doesn't allocate any heap space. Otherwise, StrCNewW makes a
            duplicate of the source string. The maximal size of the new string is limited to the
            second parameter.
Arguments:  Arg1: → Source WIDE string.
            Arg2: Maximal character count.
Return:     rax → New WIDE string copy.


Procedure:  StrCompA
File:       \ObjAsm\Code\ObjMem\64\StrCompA.asm
Purpose:    Compare 2 ANSI strings with case sensitivity.
Arguments:  Arg1: → ANSI string 1.
            Arg2: → ANSI string 2.
Return:     If string 1 < string 2, then eax < 0.
            If string 1 = string 2, then eax = 0.
            If string 1 > string 2, then eax > 0.


Procedure:  StrCompW
File:       \ObjAsm\Code\ObjMem\64\StrCompW.asm
Purpose:    Compare 2 WIDE strings with case sensitivity.
Arguments:  Arg1: → WIDE string 1.
            Arg2: → WIDE string 2.
Return:     If string 1 < string 2, then eax < 0.
            If string 1 = string 2, then eax = 0.
            If string 1 > string 2, then eax > 0.


Procedure:  StrCopyA
File:       \ObjAsm\Code\ObjMem\64\StrCopyA.asm
Purpose:    Copy an ANSI string to a destination buffer.
Arguments:  Arg1: Destination ANSI string.
            Arg2: Source ANSI string.
Return:     rax = Number of BYTEs copied, including the ZTC.


Procedure:  StrCopyW
File:       \ObjAsm\Code\ObjMem\64\StrCopyW.asm
Purpose:    Copy a WIDE string to a destination buffer.
Arguments:  Arg1: Destination WIDE string buffer.
            Arg2: Source WIDE string.
Return:     eax = Number of BYTEs copied, including the ZTC.


Procedure:  StrCPosA
File:       \ObjAsm\Code\ObjMem\64\StrCPosA.asm
Purpose:    Scan for ANSI string2 into ANSI string1 with length limitation.
Arguments:  Arg1: → Source ANSI string.
            Arg2: → ANSI string to search for.
            Arg3: Maximal character count.
```

```
Return:     rax → String position or NULL if not found.


Procedure:  StrCPosW
File:       \ObjAsm\Code\ObjMem\64\StrCPosW.asm
Purpose:    Scan for WIDE string2 into WIDE string1 with length limitation.
Arguments:  Arg1: → Source WIDE string.
            Arg2: → WIDE string to search for.
            Arg3: Maximal character count.
Return:     rax → String position or NULL if not found.


Procedure:  StrCScanA
File:       \ObjAsm\Code\ObjMem\64\StrCScanA.asm
Purpose:    Scan from the beginning of ANSI string for a character with length limitation.
Arguments:  Arg1: → Source ANSI string.
            Arg2: Maximal character count.
            Arg3: ANSI character to search for.
Return:     rax → Character address or NULL if not found.


Procedure:  StrCScanW
File:       \ObjAsm\Code\ObjMem\64\StrCScanW.asm
Purpose:    Scan from the beginning of a WIDE string for a character with length limitation.
Arguments:  Arg1: → Source WIDE string.
            Arg2: Maximal character count.
            Arg3: WIDE character to search for.
Return:     rax → Character address or NULL if not found.


Procedure:  StrDispose
File:       \ObjAsm\Code\ObjMem\64\StrDispose.asm
Purpose:    Free the memory allocated for the string using StrNew, StrCNew, StrLENew or
            StrAlloc.
            If the pointer to the string is NULL, StrDispose does nothing.
Arguments:  Arg1: → String.
Return:     Nothing.


Procedure:  StrDispose_UEFI
File:       \ObjAsm\Code\ObjMem\64\StrDispose_UEFI.asm
Purpose:    Free the memory allocated for the string using StrNew_UEFI, StrCNew_UEFI,
            StrLENew_UEFI or StrAlloc_UEFI.
            If the pointer to the string is NULL, StrDispose_UEFI does nothing.
Arguments:  Arg1: → String.
Return:     Nothing.


Procedure:  StrECatA
File:       \ObjAsm\Code\ObjMem\64\StrECatA.asm
Purpose:    Append an ANSI string to another and return the address of the ending zero character.
            StrCatA does not perform any length checking. The destination buffer must have room
            for at least StrLengthA(Destination) + StrLengthA(Source) + 1 characters.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
Return:     rax → ZTC.


Procedure:  StrECatCharA
File:       \ObjAsm\Code\ObjMem\64\StrECatCharA.asm
Purpose:    Append a character to an ANSI string and return the address of the ending zero.
            StrECatCharA does not perform any length checking. The destination buffer must have
            enough room for at least StrLengthA(Destination) + 1 + 1 characters.
Arguments:  Arg1: → Destination ANSI string buffer.
            Arg2: → ANSI character.
Return:     rax → ZTC.


Procedure:  StrECatCharW
File:       \ObjAsm\Code\ObjMem\64\StrECatCharW.asm
Purpose:    Append a character to a WIDE string and return the address of the ending zero.
            StrECatCharW does not perform any length checking. The destination buffer must have
            enough room for at least StrLengthW(Destination) + 1 + 1 characters.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: → WIDE character.
Return:     rax → ZTC.


Procedure:  StrECatW
File:       \ObjAsm\Code\ObjMem\64\StrECatW.asm
Purpose:    Append a WIDE string to another and return the address of the ending zero character.
```

```
            StrCatW does not perform any length checking. The destination buffer must have room
            for at least StrLengthW(Destination) + StrLengthW(Source) + 1 characters.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
Return:     rax → ZTC.


Procedure:  StrECopyA
File:       \ObjAsm\Code\ObjMem\64\StrECopyA.asm
Purpose:    Copy an ANSI string to a buffer and return the address of the ZTC.
            Source and destination strings may overlap.
Arguments:  Arg1: → Destination ANSI character string.
            Arg2: → Source ANSI string.
Return:     rax → ZTC.


Procedure:  StrECopyW
File:       \ObjAsm\Code\ObjMem\64\StrECopyW.asm
Purpose:    Copy a WIDE string to a buffer and return the address of the ZTC.
            Source and destination strings may overlap.
Arguments:  Arg1: → Destination WIDE character string.
            Arg2: → Source WIDE string.
Return:     rax → ZTC.


Procedure:  StrEndA
File:       \ObjAsm\Code\ObjMem\64\StrEndA.asm
Purpose:    Get the address of the zero character that terminates the string.
Arguments:  Arg1: → Source ANSI string.
Return:     rax → ZTC.


Procedure:  StrEndsWithA
File:       \ObjAsm\Code\ObjMem\64\StrEndsWithA.asm
Purpose:    Compare the ending of a string.
Arguments:  Arg1: → Analized string.
            Arg2: → Suffix string.
Return:     eax = TRUE of the ending matches, otherwise FALSE.


Procedure:  StrEndsWithW
File:       \ObjAsm\Code\ObjMem\64\StrEndsWithW.asm
Purpose:    Compare the ending of a string.
Arguments:  Arg1: → Analized string.
            Arg2: → Suffix string.
Return:     eax = TRUE of the ending matches, otherwise FALSE.


Procedure:  StrEndW
File:       \ObjAsm\Code\ObjMem\64\StrEndW.asm
Purpose:    Get the address of the zero character that terminates the string.
Arguments:  Arg1: → Source WIDE string.
Return:     rax → ZTC.


Procedure:  StrFillChrA
File:       \ObjAsm\Code\ObjMem\64\StrFillChrA.asm
Purpose:    Fill a preallocated String with a character.
Arguments:  Arg1: → String.
            Arg2: Character.
            Arg3: Character Count.
Return:     Nothing.


Procedure:  StrFillChrW
File:       \ObjAsm\Code\ObjMem\64\StrFillChrW.asm
Purpose:    Fill a preallocated String with a character.
Arguments:  Arg1: → String.
            Arg2: Character.
            Arg3: Character Count.
Return:     Nothing.


Procedure:  StrFilterA
File:       \ObjAsm\Code\ObjMem\64\StrFilterA.asm
Purpose:    Perform a case sensitive string match test using wildcards (* and ?).
Arguments:  Arg1: → Source ANSI string.
            Arg2: → Filter ANSI string.
Return:     eax = TRUE if strings match, otherwise FALSE.
```

```
Procedure:  StrFilterW
File:       \ObjAsm\Code\ObjMem\64\StrFilterW.asm
Purpose:    Perform a case sensitive string match test using wildcards (* and ?).
Arguments:  Arg1: → Source WIDE string.
            Arg2: → Filter WIDE string.
Return:     eax = TRUE if strings match, otherwise FALSE.


Procedure:  StrICompA
File:       \ObjAsm\Code\ObjMem\64\StrICompA.asm
Purpose:    Compare 2 ANSI strings without case sensitivity.
Arguments:  Arg1: → ANSI string 1.
            Arg2: → ANSI string 2.
Return:     If string 1 < string 2, then eax < 0.
            If string 1 = string 2, then eax = 0.
            If string 1 > string 2, then eax > 0.


Procedure:  StrICompW
File:       \ObjAsm\Code\ObjMem\64\StrICompW.asm
Purpose:    Compare 2 WIDE strings without case sensitivity.
Arguments:  Arg1: → WIDE string 1.
            Arg2: → WIDE string 2.
Return:     If string 1 < string 2, then eax < 0.
            If string 1 = string 2, then eax = 0.
            If string 1 > string 2, then eax > 0.


Procedure:  StrIFilterA
File:       \ObjAsm\Code\ObjMem\64\StrIFilterA.asm
Purpose:    Perform a case insensitive string match test using wildcards (* and ?).
Arguments:  Arg1: → Source ANSI string.
            Arg2: → Filter ANSI string.
Return:     eax = TRUE if strings match, otherwise FALSE.


Procedure:  StrIFilterW
File:       \ObjAsm\Code\ObjMem\64\StrIFilterW.asm
Purpose:    Perform a case insensitive string match test using wildcards (* and ?).
Arguments:  Arg1: → Source WIDE string.
            Arg2: → Filter WIDE string.
Return:     eax = TRUE if strings match, otherwise FALSE.


Procedure:  StrLeftA
File:       \ObjAsm\Code\ObjMem\64\StrLeftA.asm
Purpose:    Extract the left n characters of the source ANSI string.
Arguments:  Arg1: → Destination character buffer.
            Arg2: → Source ANSI string.
Return:     eax = Number of copied characters, not including the ZTC.


Procedure:  StrLeftW
File:       \ObjAsm\Code\ObjMem\64\StrLeftW.asm
Purpose:    Extract the left n characters of the source WIDE string.
Arguments:  Arg1: → Destination buffer.
            Arg2: → Source WIDE string.
Return:     eax = Number of copied characters, not including the ZTC.


Procedure:  StrLengthA
File:       \ObjAsm\Code\ObjMem\64\StrLengthA.asm
Purpose:    Determine the length of an ANSI string not including the zero terminating character.
Arguments:  Arg1: → Source ANSI string.
Return:     eax = Length of the string in characters.


Procedure:  StrLengthW
File:       \ObjAsm\Code\ObjMem\64\StrLengthW.asm
Purpose:    Determine the length of a WIDE string not including the ZTC.
Arguments:  Arg1: → WIDE string.
Return:     rax = Length of the string in characters.


Procedure:  StrLowerA
File:       \ObjAsm\Code\ObjMem\64\StrLowerA.asm
Purpose:    Convert all ANSI string characters into lowercase.
Arguments:  Arg1: → Source ANSI string.
Return:     rax → String.
```

```
Procedure:  StrLowerW
File:       \ObjAsm\Code\ObjMem\64\StrLowerW.asm
Purpose:    Convert all WIDE string characters into lowercase.
Arguments:  Arg1: → Source WIDE string.
Return:     rax → String.


Procedure:  StrLRTrimA
File:       \ObjAsm\Code\ObjMem\64\StrLRTrimA.asm
Purpose:    Trim blank and tab characters from the beginning and the end of an ANSI string.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
Return:     Nothing.
Note:       Source and Destination may overlap.


Procedure:  StrLRTrimW
File:       \ObjAsm\Code\ObjMem\64\StrLRTrimW.asm
Purpose:    Trim blank and tab characters from the beginning and the end of a WIDE string.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
Return:     Nothing.
Note:       Source and Destination may overlap.


Procedure:  StrLScanA
File:       \ObjAsm\Code\ObjMem\64\StrLScanA.asm
Purpose:    Scan for a character from the beginning of an ANSI string.
Arguments:  Arg1: → Source ANSI string.
            Arg2: Character to search.
Return:     rax → Character address or NULL if not found.


Procedure:  StrLScanW
File:       \ObjAsm\Code\ObjMem\64\StrLScanW.asm
Purpose:    Scan for a character from the beginning of a WIDE string.
Arguments:  Arg1: → Source WIDE string.
            Arg2: Character to search for.
Return:     rax → Character address or NULL if not found.


Procedure:  StrLTrimA
File:       \ObjAsm\Code\ObjMem\64\StrLTrimA.asm
Purpose:    Trim blank characters from the beginning of an ANSI string.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
Return:     Nothing.


Procedure:  StrLTrimW
File:       \ObjAsm\Code\ObjMem\64\StrLTrimW.asm
Purpose:    Trim blank characters from the beginning of a WIDE string.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
Return:     Nothing.


Procedure:  StrMidA
File:       \ObjAsm\Code\ObjMem\64\StrMidA.asm
Purpose:    Extract a substring from an ANSI source string.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
            Arg3: Start character index. Index ranges [0 .. length-1].
            Arg3: Character count.
Return:     eax = Number of copied characters.


Procedure:  StrMidW
File:       \ObjAsm\Code\ObjMem\64\StrMidW.asm
Purpose:    Extract a substring from a WIDE source string.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
            Arg3: Start character index. Index ranges [0 .. length-1].
            Arg3: Character count.
Return:     eax = Number of copied characters.


Procedure:  StrMoveW
File:       \ObjAsm\Code\ObjMem\64\StrMoveW.asm
Purpose:    Move part of a WIDE string. The ZTC is NOT appended automatically.
            Source and destination strings may overlap.
```

```
Arguments:  Arg1: → Destination buffer.
            Arg2: → Source WIDE string.
            Arg3: Character count.
Return:     Nothing.


Procedure:  StrNewA
File:       \ObjAsm\Code\ObjMem\64\StrNewA.asm
Purpose:    Allocate a new copy of the source string.
            If the pointer to the source string is NULL, StrNew returns NULL and doesn't allocate
            any memory space. Otherwise, StrNew makes a duplicate of the source string.
            The allocated memory space is Length(String) + ZTC.
Arguments:  Arg1: → Source WIDE string.
Return:     rax → New string copy.


Procedure:  StrNewA_UEFI
File:       \ObjAsm\Code\ObjMem\64\StrNewA_UEFI.asm
Purpose:    Allocate a new copy of the source string.
            If the pointer to the source string is NULL, StrNew returns NULL and doesn't allocate
            any memory space. Otherwise, StrNew makes a duplicate of the source string.
            The allocated memory space is Length(String) + ZTC.
Arguments:  Arg1: → Source WIDE string.
Return:     rax → New string copy.


Procedure:  StrNewW
File:       \ObjAsm\Code\ObjMem\64\StrNewW.asm
Purpose:    Allocate a new copy of the source string.
            If the pointer to the source string is NULL, StrNew returns NULL and doesn't allocate
            any memory space. Otherwise, StrNew makes a duplicate of the source string.
            The allocated memory space is Length(String) + ZTC.
Arguments:  Arg1: → Source WIDE string.
Return:     rax → New string copy.


Procedure:  StrNewW_UEFI
File:       \ObjAsm\Code\ObjMem\64\StrNewW_UEFI.asm
Purpose:    Allocate a new copy of the source string.
            If the pointer to the source string is NULL, StrNew returns NULL and doesn't allocate
            any memory space. Otherwise, StrNew makes a duplicate of the source string.
            The allocated memory space is Length(String) + ZTC.
Arguments:  Arg1: → Source WIDE string.
Return:     rax → New string copy.


Procedure:  StrPosA
File:       \ObjAsm\Code\ObjMem\64\StrPosA.asm
Purpose:    Find the occurence of string 2 into string1.
Arguments:  Arg1: → Source ANSI string.
            Arg2: → Searched ANSI string.
Return:     rax → String occurence or NULL if not found.


Procedure:  StrPosW
File:       \ObjAsm\Code\ObjMem\64\StrPosW.asm
Purpose:    Find the occurence of string 2 into string1.
Arguments:  Arg1: → Source WIDE string.
            Arg2: → Searched WIDE string.
Return:     rax → String occurence or NULL if not found.


Procedure:  StrRepChrA
File:       \ObjAsm\Code\ObjMem\64\StrRepChrA.asm
Purpose:    Create a new string filled with a given char.
Arguments:  Arg1: Used character.
            Arg2: Repetition count.
Return:     rax → New string or NULL if failed.


Procedure:  StrRepChrW
File:       \ObjAsm\Code\ObjMem\64\StrRepChrW.asm
Purpose:    Create a new string filled with a given char.
Arguments:  Arg1: Used character.
            Arg2: Repetition count.
Return:     rax → New string or NULL if failed.


Procedure:  StrRightA
File:       \ObjAsm\Code\ObjMem\64\StrRightA.asm
Purpose:    Copy the right n characters from the source string into the destination buffer.
```

```
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
            Arg3: Character count.
Return:     rax = Number of characters.


Procedure:  StrRightW
File:       \ObjAsm\Code\ObjMem\64\StrRightW.asm
Purpose:    Copy the right n characters from the source string into the destination buffer.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
            Arg3: Character count.
Return:     rax = Number of characters.


Procedure:  StrRScanA
File:       \ObjAsm\Code\ObjMem\64\StrRScanA.asm
Purpose:    Scan from the end of an ANSI string for a character.
Arguments:  Arg1: → Source ANSI string.
            Arg2: Character to search for.
Return:     rax → Character address or NULL if not found.


Procedure:  StrRScanW
File:       \ObjAsm\Code\ObjMem\64\StrRScanW.asm
Purpose:    Scan from the end of a WIDE string for a character.
Arguments:  Arg1: → Source WIDE string.
            Arg2: Character to search for.
Return:     rax → Character address or NULL if not found.


Procedure:  StrRTrimA
File:       \ObjAsm\Code\ObjMem\64\StrRTrimA.asm
Purpose:    Trim blank characters from the end of an ANSI string.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source ANSI string.
Return:     Nothing.


Procedure:  StrRTrimW
File:       \ObjAsm\Code\ObjMem\64\StrRTrimW.asm
Purpose:    Trim blank characters from the end of a WIDE string.
Arguments:  Arg1: → Destination WIDE character buffer.
            Arg2: → Source WIDE string.
Return:     eax = Number of characters in destination buffer.


Procedure:  StrSizeA
File:       \ObjAsm\Code\ObjMem\64\StrSizeA.asm
Purpose:    Determine the size of an ANSI string including the ZTC.
Arguments:  Arg1: → ANSI string.
Return:     eax = Size of the string in BYTEs.


Procedure:  StrSizeW
File:       \ObjAsm\Code\ObjMem\64\StrSizeW.asm
Purpose:    Determine the size of a WIDE string including the ZTC.
Arguments:  Arg1: → WIDE string.
Return:     rax = Size of the string in BYTEs.


Procedure:  StrStartsWithA
File:       \ObjAsm\Code\ObjMem\64\StrStartsWithA.asm
Purpose:    Compare the beginning of a string.
Arguments:  Arg1: → Analized string.
            Arg2: → Prefix string.
Return:     eax = TRUE of the beginning matches, otherwise FALSE.


Procedure:  StrStartsWithW
File:       \ObjAsm\Code\ObjMem\64\StrStartsWithW.asm
Purpose:    Compare the beginning of a string.
Arguments:  Arg1: → Analized string.
            Arg2: → Prefix string.
Return:     eax = TRUE of the beginning matches, otherwise FALSE.


Procedure:  StrToSt0A
File:       \ObjAsm\Code\ObjMem\64\StrToSt0A.asm
Purpose:    Load an ANSI string representation of a floating point number into the st(0)
            FPU register.
```

```
Arguments:  Arg1: → ANSI string floating point number.
Return:     eax = Result code f_OK or f_ERROR.
Notes:      - Based on the work of Raymond Filiatreault (FpuLib).
            - Source string should not be greater than 19 chars + zero terminator.


Procedure:  StrToSt0W
File:       \ObjAsm\Code\ObjMem\64\StrToSt0W.asm
Purpose:    Load a WIDE string representation of a floating point number into the st(0)
            FPU register.
Arguments:  Arg1: → WIDE string floating point number.
Return:     eax = Result code f_OK or f_ERROR.
Note:       - Based on the work of Raymond Filiatreault (FpuLib).
            - Source string should not be greater than 19 chars + zero terminator.


Procedure:  StrUpperA
File:       \ObjAsm\Code\ObjMem\64\StrUpperA.asm
Purpose:    Convert all ANSI string characters into uppercase.
Arguments:  Arg1: → Source ANSI string.
Return:     rax → String.


Procedure:  StrUpperW
File:       \ObjAsm\Code\ObjMem\64\StrUpperW.asm
Purpose:    Convert all WIDE string characters into uppercase.
Arguments:  Arg1: → Source WIDE string.
Return:     rax → String.


Procedure:  StrW2StrA
File:       \ObjAsm\Code\ObjMem\64\StrW2StrA.asm
Purpose:    Convert a WIDE string into an ANSI string. WIDE characters are converted to BYTEs by
            decimation of the high byte.
Arguments:  Arg1: → Destination ANSI character buffer.
            Arg2: → Source WIDE string.
Return:     rax = Number of characters.


Procedure:  SysShutdown
File:       \ObjAsm\Code\ObjMem\64\SysShutdown.asm
Purpose:    Shut down the system.
Arguments:  Arg1: Shutdown type.
            Arg2:
Return:     Nothing.


Procedure:  SysStandby
File:       \ObjAsm\Code\ObjMem\64\SysStandby.asm
Purpose:    Set the system in standby modus.
Arguments:  None.
Return:     Nothing.


Procedure:  uCRC32C
File:       \ObjAsm\Code\ObjMem\64\uCRC32C.asm
Purpose:    Compute the CRC-32C (Castagnoli), using the polynomial 11EDC6F41h from an unaligned
            memory block.
Arguments:  Arg1: → Unaligned memory block.
            Arg2: Memory block size in BYTEs.
Return:     eax = CRC32C.


Procedure:  udword2decA
File:       \ObjAsm\Code\ObjMem\64\udword2decA.asm
Purpose:    Convert a unsigned DWORD to its decimal ANSI string representation.
Arguments:  Arg1: → Destination ANSI string buffer.
            Arg2: DWORD value.
Return:     eax = Number of BYTEs copied to the destination buffer, including the ZTC.
Note:       The destination buffer must be at least 11 BYTEs large to allocate the output string
            (10 ANSI characters + ZTC = 11 BYTEs).


Procedure:  udword2decW
File:       \ObjAsm\Code\ObjMem\64\udword2decW.asm
Purpose:    Convert an unsigned DWORD to its decimal WIDE string representation.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: DWORD value.
Return:     eax = Number of BYTEs copied to the destination buffer, including the ZTC.
Note:       The destination buffer must be at least 22 BYTEs large to allocate the output string
            (10 WIDE characters + ZTC = 22 BYTEs).
```

```
Procedure:  UefiGetErrStrA
File:       \ObjAsm\Code\ObjMem\64\UefiGetErrStrA.asm
Purpose:    Return a description ANSI string from an UEFI error code.
Arguments:  Arg1: UEFI error code.
Return:     rax → Error string.


Procedure:  UefiGetErrStrW
File:       \ObjAsm\Code\ObjMem\64\UefiGetErrStrW.asm
Purpose:    Return a description WIDE string from an UEFI error code.
Arguments:  Arg1: UEFI error code.
Return:     rax → Error string.


Procedure:  uqword2decA
File:       \ObjAsm\Code\ObjMem\64\uqword2decA.asm
Purpose:    Convert an unsigned QWORD into its decimal ANSI string representation.
Arguments:  Arg1: → Destination ANSI string buffer.
            Arg2: QWORD value.
Return:     eax = Number of BYTEs copied to the destination buffer, including the ZTC.
Note:       The destination buffer must be at least 21 BYTEs large to allocate the output string
            (20 ANSI characters + ZTC = 21 BYTEs).


Procedure:  uqword2decW
File:       \ObjAsm\Code\ObjMem\64\uqword2decW.asm
Purpose:    Convert an unsigned QWORD into its decimal WIDE string representation.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: QWORD value.
Return:     eax = Number of BYTEs copied to the destination buffer, including the ZTC.
Note:       The destination buffer must be at least 42 BYTEs large to allocate the output string
            (20 WIDE characters + ZTC = 42 BYTEs).


Procedure:  UrlEscDecode
File:       \ObjAsm\Code\ObjMem\64\URL.asm
Purpose:    Translate a wide string containig URL escape sequences to a plain wide string.
Arguments:  Arg1: → Input wide string.
            Arg2: → Output Buffer.
            Arg3: Output Buffer size in BYTEs.
Return:     eax = Number of chars written, including the ZTC.
Procedure:  UrlEscEncode
Purpose:    Translate a plain wide string to a wide string containig URL escape sequences.
Arguments:  Arg1: → Input wide string.
            Arg2: → Output Buffer.
            Arg3: Output Buffer size in BYTEs.
Return:     eax = Number of chars written.


Procedure:  UTF8ToWide
File:       \ObjAsm\Code\ObjMem\64\UTF8ToWide.asm
Purpose:    Convert an UTF8 byte stream to a WIDE (UTF16) string.
Arguments:  Arg1: → Destination WIDE buffer.
            Arg2: → Source UTF8 BYTE stream. Must be zero terminated.
            Arg3: Destination buffer size in BYTEs.
Return:     eax = Number of BYTEs written.
            ecx = 0: succeeded
            1: buffer full
            2: conversion error
Notes:      - The destination WIDE string is always terminated with a ZTC
            (only if buffer size >= 2).


Procedure:  WaitForProcess
File:       \ObjAsm\Code\ObjMem\64\WaitForProcess.asm
Purpose:    Synchronisation procedure that waits until a process has finished.
Arguments:  Arg1: Process ID.
            Arg2: Timeout value in ms.
Return:     eax = Wait result (WAIT_ABANDONED, WAIT_OBJECT_0 or WAIT_TIMEOUT) or -1 if failed.


Procedure:  WideToUTF8
File:       \ObjAsm\Code\ObjMem\64\WideToUTF8.asm
Purpose:    Convert an WIDE string to an UTF8 encoded stream.
Arguments:  Arg1: → Destination buffer.
            Arg2: → Source WIDE string.
            Arg3: Destination buffer size in BYTEs.
Return:     eax = Number of BYTEs written.
            ecx = 0: succeeded
```

```
            1: buffer full
Notes:      - The destination stream is always zero terminated.


Procedure:  WndFadeIn
File:       \ObjAsm\Code\ObjMem\64\WndFadeIn.asm
Purpose:    Fade in a window when WS_EX_LAYERED is set.
Arguments:  Arg1: Window HANDLE.
            Arg2: Transparency start value.
            Arg3: Transparency end value.
            Arg4: Transparency increment value.
            Arg5: Delay between steps.
Return:     Nothing.


Procedure:  WndFadeOut
File:       \ObjAsm\Code\ObjMem\64\WndFadeOut.asm
Purpose:    Fade out a window when WS_EX_LAYERED is set.
Arguments:  Arg1: Window HANDLE.
            Arg2: Transparency start value.
            Arg3: Transparency end value.
            Arg4: Transparency decrement value.
            Arg5: Delay between steps.
Return:     Nothing.


Procedure:  word2hexA
File:       \ObjAsm\Code\ObjMem\64\word2hexA.asm
Purpose:    Convert a DORD to its hexadecimal ANSI string representation.
Arguments:  Arg1: → Destination ANSI string buffer.
            Arg2: WORD value.
Return:     Nothing.
Notes:      The destination buffer must be at least 5 BYTEs large to allocate the output string
            (4 character BYTEs + ZTC = 5 BYTEs).


Procedure:  word2hexW
File:       \ObjAsm\Code\ObjMem\64\word2hexW.asm
Purpose:    Convert a WORD to its hexadecimal WIDE string representation.
Arguments:  Arg1: → Destination WIDE string buffer.
            Arg2: WORD value.
Return:     Nothing.
Notes:      The destination buffer must be at least 9 BYTEs large to allocate the output string
            (4 character WORDs + ZTC = 9 BYTEs).
```