Microsoft Real-time Compression Interface Specification
Version 1.00.09
April 30, 1993

Contents

1. What is MRCI? 2
2. What MRCI defines 2
3. How to implement MRCI 2
4. Operating System Support 3
5. Overview of MRCI Services 3
6. MRCI Interrupt 4
7. MRCI Definitions 4
8. MRCI System Client Example 12
9. MRCI Server Example 15

© 1992-1993, Microsoft Corporation

1. What is MRCI?

MRCI (pronounced "Merci") is the *Microsoft Real-time Compression Interface,* a software interface definition that allows a *MRCI client* to request compression services from a *MRCI server* that supports the *Microsoft Real-time Compression Format* (MRCF). A MRCI server may implement support for MRCF compression and decompression either fully in software, or with partial or complete hardware assistance. Today, MRCI servers exist totally in software (for example, the Microsoft DoubleSpace compressed file system includes a software MRCI server), but in the future hardware-based MRCI servers will be commonplace.

MRCI has two purposes:

1) To define standard, system-level, compress-decompress services that may be used by ISVs, independent of the implementation of those services.
2) To define a standard way for IHVs and PC manufacturers to implement dedicated compression hardware.

Note that MRCI defines a standard for *lossless* compression, which is different from the *lossy* compression used for pictorial and video images and defined in standards such as JPEG and MPEG. Lossless compression is useful for many operating system and application functions, especially where the type of data is unknown, and hence cannot benefit from data type-specific algorithms. Its benefits include:

- Increasing the effective storage capacity of media
- Maximizing the effective data bandwidth between two computers
- Maximizing the effective data bandwidth between a computer and a storage device.

MRCI is currently used by the Microsoft DoubleSpace compressed file system, the Microsoft Flash File System, and the Microsoft Backup program in MS-DOS 6.

Since MRCI defines an interface standard that allows for hardware implementations, it is important to note the advantages that compression-decompression hardware has over compression-decompression software:

- *Performance*: Tightly-integrated hardware (on the local bus, for example) will improve performance, but until there are actual implementations it is difficult to pinpoint the magnitude of improvement.
- *Multitasking performance*: When used by a multithreaded operating system such as Windows NT, the hardware is effectively a highly specialized second CPU that frees up the main CPU to execute other threads not requesting compression services.
- *Better compression*: By using more intensive matching algorithms, hardware can improve compression ratios 10%-15% over software, but without paying any performance penalty.

2. What MRCI defines

MRCI defines three principal standards:

- *Query API*: MRCI defines a rendezvous API that lets an application check for the presence of MRCI-compliant compression services (possibly implemented in hardware).
- *Compress and Decompress APIs*: If a MRCI server is present, an application can disable its own internal software compression-decompression routines. Using the address returned from the query, the application can then transfer data to and from the MRCI server that it needs compressed or decompressed.
- *Compression Format*: A standard compression format means that given a stream of uncompressed bytes as input, all MRCI-compliant hardware will output the exact same stream of compressed bytes, and vice-versa for decompression. By defining a format, which is a variant of Lempel-Ziv encoding, MRCI enables the exchange of compressed data between systems.

3. How to implement MRCI

ISVs wishing to take advantage of MRCI will need the following:

- *MRCI specification*: Available to the general public

- *MRCI software libraries for MS-DOS and Windows*: The libraries (MRCFDOS.LIB and MRCFWIN.LIB) provide very fast, real-time compression-decompression services in software and adhere to the MRCI compression format. The libraries check for the presence of a MRCI server and use it if present. If not present, the libraries use their own internal software routines. So by simply linking in a library, an ISV is guaranteed to have fast, real-time, MRCI-compliant compression-decompression services regardless of the hardware environment. There is no need for the ISV to learn about the details of MRCF, as these are hidden by the libraries.

The MRCI software libraries, which are the same libraries used in MS-DOS 6's DoubleSpace and MS-DOS 6's Backup program, are available for license to ISVs for no fee from Microsoft.

IHVs wishing to develop MRCI-compliant hardware will need the following:
- *MRCI specification*: Available to the general public.
- *MRCI software libraries for MS-DOS and Windows*: Same as above, and necessary to test hardware for MRCI compatibility.
- *MRCI algorithms*: C language reference code that implements the MRCI compression and decompression algorithms.
- *MRCI compression format*: A description of the MRCI compression format.

The MRCI software libraries, algorithms and compression format are available for license to IHVs for no fee from Microsoft.

4. Operating System Support

MRCI currently supports the MS-DOS and Microsoft Windows operating systems. MRCI for Windows NT will be defined at a later date, as an extension to the Hardware Abstraction Layer (HAL).

5. Overview of MRCI Services

The MRCI services are briefly described in the following table.

Service	Description
MRCQuery	Return MRCI server information
MRCCompress	Compress a data buffer using StandardCompression
MRCDecompress	Decompress a data buffer
MRCIncrementalDecompress	Decompress a portion of a data buffer (optimization for DoubleSpace)

MRCQuery enables a client or server to establish communication with an existing MRCI server. A software interrupt is used to make this call, the interrupt number being chosen to permit the server to be supplied in any manner (as code in ROM, or as a DOS device driver or TSR, or as a Windows VxD). A new MRCI server can install itself in the system, partially or fully replacing a previously installed MRCI server. Only the last MRCI server installed is accessible via **MRCQuery**.

The compression and decompression services, **MRCCompress, MRCDecompress**, and **MRCIncrementalDecompress**, permit a client to call a server to compress and decompress data in the MRC Format (MRCF). These services are invoked via a direct call to the server (the address having been obtained by **MRCQuery**), for maximum performance.

NOTE: A client must enter the Windows Disk Critical Section **before** calling the direct call entry point of a MRCI server. This protects the MRCI server from being reentered. Failure to do so will cause data corruption and loss in a multi-tasking environment. See the sample code below for details.

6. MRCI Interrupt

MRCI defines a new interrupt 2Fh call (AX=4A12h) to allow an MRCI client to detect an MRCI server, and a new interrupt 1Ah call (AX=B001h) to allow the first MRCI server or client to detect a ROM BIOS (presumably hardware-based) MRCI server. MRCI clients and servers first check for the presence of an existing MRCI server by issuing the INT 2Fh call. If that fails, then they issue the INT 1Ah call. If that fails, then there is no MRCI server present in the system.

These two *rendezvous* interrupts are provided to ensure compatibility with existing MS-DOS software. If a software MRCI server intends to install itself over an existing INT 1Ah MRCI server, or a MRCI client is going to remain resident (either as an MS-DOS device driver or terminate-and-stay-resident (TSR) program), then it must hook INT 2Fh and create a RAM copy of the MRCINFO structure (see below). The procedure allows subsequent programs to supersede the MRCI server without cutting the initial resident program out of the loop.

7. MRCI Definitions

The following ASM headers document MRCI. These are specified in ASM because that is the highest-performance interface. These may be called from C with suitable C "wrapper" functions.

```
;***    MRCI.INC - Microsoft Real-Time Compression Interface definitions
;
;       MRCI version 1.00.08  02-Apr-1993

intMRCI         equ       2Fh       ; MRCI interrupt number
mrciDETECT      equ    04A12h       ; intMRCI AX for detecting MRCI server

intMRCIROM      equ       1Ah       ; ROM MRCI interrupt number
mrciDETECTROM   equ    0B001h       ; intMRCIROM AX for detecting MRCI server


;***    mcXXXX - flag values passed to MRCI operations
;
;       MRCCompress and MRCDecompress take a flag to indicate whether the
;       client is a *system* component (and hence may call with InDOS
;       set), or an *application*.
;
;       mcSYSTEM clients must ensure the following is true before calling
;       MRCI:
;           1)  The Windows Disk Critical Section is owned
;           2)  The InDOS flag is *set*
;
;       mcAPPLICATION clients must ensure the following is true before calling
;       MRCI:
;           1)  The Windows Disk Critical Section is owned
;           2)  The InDOS flag is *clear*
;
;       FAILING TO FOLLOW THE ABOVE RULES WILL LIKELY RESULT IN A SYSTEM
;       HANG AND LOSS OF USER DATA.
;
mcAPPLICATION   equ     0   ; Client is an application
mcSYSTEM        equ     1   ; Client is a file system driver


;***    micapXXXXX - bit flags for MRCINFO.mi_flCapabilities
;
;       These define both the capabilities of the Server, and also double
;       as *operation* codes passed to the mi_pfnOperate entry point in
;       the server.
;                           111111
;                           5432109876543210
;                           ----------------
micapNONE           equ     0000000000000000b ; No capabilities
micapSTANDARD       equ     0000000000000001b ; Standard compress
micapDECOMPRESS     equ     0000000000000010b ; Standard compress
```

```
micapRESERVED1      equ    0000000000000100b ; RESERVED for future use
micapRESERVEDM      equ    0000000000001000b ; RESERVED for future use
micapRESERVED2      equ    0000000000010000b ; RESERVED for future use
micapINCDECOMP      equ    0000000000100000b ; Incremental Decompress
   ;
   ; Remaining bits (6..14) are RESERVED and must be 0
   ;
micapREADONLY       equ    1000000000000000b ; MRCINFO structure is read-only
micapDEINSTALL      equ    1111111111111111b ; Server deinstall service


;***    MRCINFO - MRC Information data structure
;
;       A pointer to an MRCINFO structure is returned from MRCQuery, and sent
;       on MRCNotifyLoad.  This structure contains information on the MRCI
;       server and its capabilities.

MRCINFO struc

mi_lVendor          dd  ?       ; A 4-byte vendor ID.
                                ; Microsoft's vendor ID is "MSFT".

mi_wVendorVersion   dw  ?       ; Version number of the MRC server.
                                ; High byte is major number, low byte is minor.
                                ; EXAMPLES: v3.20 = 0314h, v10.01 = 0A01h

mi_wMRCIVersion     dw  ?       ; Version number of the MRCI supported by
                                ; this server.

mi_pfnOperate       dd  ?       ; Far pointer of the server compression entry
                                ; point.
                                ; NOTE: Caller must ensure that the Windows
                                ;       critical section is held *before*
                                ;       calling this entry point!

mi_flCapability     dw  ?       ; Bit field of server capabilities
                                ; See micapXXX for bit definitions

mi_flHWAssist       dw  ?       ; Bit field of hardware assisted
                                ; server capabilities.  One-to-one
                                ; correspondance with mi_flCapability
                                ; bits.  A bit set in this field
                                ; indicates the corresponding
                                ; capability is hardware assisted.

mi_cbMax            dw  ?       ; Maximum number of bytes that the compression
                                ; services provider can compress or decompress.
                                ; Requests to compress or decompress buffers in
                                ; excess of this length will fail.
                                ; All MRCI servers are to support at least
                                ; 8192 byte (8Kb) blocks.
MRCINFO ends


;***    MRCREQUEST - MRC compress/decompress Request packet
;
;       This structure is used to pass parameters to the server for
;       compress/decompress operations.
;
;       General Notes
;       -------------
;       (1) <mr_pbSrc,mr_cbSrc> and <mr_pbDst,mr_cbDst> MUST NOT
;           OVERLAP!
;
;       (2) The safest practice is for mr_cbSrc and mr_cbDst to be
;           identical.
;
```

```
;       Details on Structure Members
;       ----------------------------
;       mr_pbSrc
;           This points to the *source* buffer.
;
;           On a *compress* operation, the contents of this buffer are
;           *uncompressed* data.
;
;           On a *decompress* operation, the contents of this buffer are
;           *compressed* data.
;
;           On a *incremental decompress* operation, this field points to
;           the next section of compressed data to be uncompressed.  The
;           server updates the offset portion of this address after each
;           incremental decompress call and the application should not modify
;           this address between incremental decompress calls on the same block
;           of compressed data.
;
;       mr_cbSrc
;           This is the size of the *source* buffer.
;
;           On a *compress* operation, this is the amount of data to
;           be compressed.
;
;           For a *decompress* operation, this value is ignored.  The amount
;           of data to be decompressed is specified by the mr_cbDst parameter,
;           described below.
;
;       mr_RESERVED
;           RESERVED for future use.  Should be 0.
;
;       mr_pbDst
;           This points to the *destination* buffer.
;
;           On a *compress* operation, this buffer receives the *compressed*
;           result of the operation.
;
;           On a *decompress* operation, this buffer receives the
;           *uncompressed* result of the operation.
;
;           On a *incremental decompress* operation, this field points to
;           the next location in the destination buffer where uncompressed
;           data is to be stored.  The server updates the offset portion of
;           this address after each incremental decompress call, and the
;           application should not modify this address between incremental
;           decompress calls on the same block of compressed data.
;
;       mr_cbDst
;           On INPUT, for a *compress* operation, this is the size of the
;           *destination* buffer. If the compressed data would overflow
;           this buffer length, then the operation fails and the server
;           returns the error MRCI_ERROR_BUFFER_OVERFLOW.
;
;           On INPUT, for a *decompress* operation, this must be EXACTLY the
;           number of bytes that will be decompressed, as the MRCI server
;           uses this information to determine when to stop decompressing.
;
;           On INPUT, for an *incremental decompress* operation, this is
;           the number of bytes that should uncompressed at this time.  This
;           will typically be less than the original uncompressed size of the
;           compressed block.  A single compressed block can be uncompressed in
;           steps by making multiple incremental decompress calls for smaller
;           sized blocks.  However, to incrementally decompress an entire
;           compressed block requires that the sum of the individual mr_cbDst
;           counts be EXACTLY the number of bytes that were originally
;           compressed so the MRCI server can determine when to stop
;           decompressing.
```

```
;
;
;           On OUTPUT, the Server updates this field with the actual size
;           of the resulting compressed/uncompressed data.
;
;       mr_cbChunk
;           This is information that the Server compress routines can use to
;           "early out" of the compression as early as possible.
;
;           Valid values are 1 (client is interested in savings as small as
;           1 byte) to 32767.  DblSpace passes 512, and Flash File System
;           passes 1.
;
;           This field is most easily explained by giving an example:
;
;           Example:
;               DblSpace does space allocation in chunks of 512 bytes (the
;               common sector size on a disk).
;
;               The compression server can use this information for two
;               optimizations:
;
;               (1) If the Server cannot compress the uncompressed data
;                   enough to save at least 512 bytes, then the data is
;                   *incompressible* as far as DblSpace is concerned, even
;                   if it could be compressed to save fewer than 512 bytes.
;
;               (2) While compressing, if the Server gets to a point where
;                   the remaining uncompressed data is of such a length that
;                   it can be encoded simply (without table lookups, etc.)
;                   and not cross a 512 byte boundary, then the Server can
;                   do the simple encoding.
;
;           It is likely that these optimizations will be hard to perform
;           quickly in software, but it is possible that hardware can do
;           these optimizations without any performance loss.
;
;       mr_dwIncDecomp
;           NOTE: This is used for Incremental Decompression only.
;
;           For the first *incremental decompression* call on a compressed
;           block, this value must be set to zero.  Upon return, the field
;           will contain state information for use on the next incremental
;           decompress call.  This value must not be modified between
;           subsequent incremental decompress calls on the same compressed
;           block.

MRCREQUEST  struc
mr_pbSrc        dd  ?   ; Pointer to source buffer
mr_cbSrc        dw  ?   ; Size of source buffer, in bytes
mr_RESERVED     dw  ?   ; RESERVED for future use.
mr_pbDst        dd  ?   ; Pointer to destination buffer
mr_cbDst        dw  ?   ; Size of destination buffer, in bytes
mr_cbChunk      dw  ?   ; Client compressed data storage chunk size (see above!)
mr_dwIncDecomp  dd  ?   ; Incremental Decompression state
MRCREQUEST  ends


;***    MRCI_ERROR_XXX definitions
;
;       Error codes returned from MRCIOperate
;
MRCI_ERROR_NONE             equ     0   ; No error
MRCI_ERROR_NOT_SUPPORTED    equ     1   ; Unsupported operation requested
MRCI_ERROR_BUSY             equ     2   ; Server is busy
MRCI_ERROR_BUFFER_OVERFLOW  equ     3   ; Destination buffer too small
MRCI_ERROR_NOT_COMPRESSIBLE equ     4   ; Data could not be compressed
```

```
MRCI_ERROR_BAD_MRC_FORMAT   equ     5   ; Compressed data format is bad

;***    DefineMRCQuery - Macro to generate MRCQuery function
;
;       Put this macro somewhere in your code segment.  It will define the
;       MRCQuery routine, which you can then call far.  See the MRCQuery
;       header below for documentation on its behavior.

DefineMRCQuery  macro


;***    sigOLD_CX, sigOLD_DX, sigNEW_CX, sigNEW_DX - MRCI Server detection
;
;       These values are used to verify that the response from issuing
;       intMRCI is coming from an MRCI server, and not some other piece
;       of code.
;
;       The *old* values are passed on the mrciQUERY call, and the server
;       must change CX/DX to the *new* values, so that the caller can
;       trust that the MRCI server was responding, and not some other
;       interrupt hook.
;
;       The Server uses this code sequence to transform CX/DX:
;
;           ;------------- entry: cx='ab'  dx='cd'
;
;           xchg    ch,cl       ; cx='ba'  dx='cd'
;           xchg    dh,dl       ; cx='ba'  dx='dc'
;           xchg    dx,cx       ; cx='dc'  dx='ba'
;
sigOLD_CX   equ 'MR'
sigOLD_DX   equ 'CI'

sigNEW_CX   equ 'IC'
sigNEW_DX   equ 'RM'


;***    MRCQuery - Detect presence of MRCI server, return MRCINFO
;
;       Detect presence of MRCI server safely, and if present return
;       pointer to the server's MRCINFO structure.  NOTE that we check
;       first for a RAM-based server, and then for a ROM-based server.
;
;       Entry
;           none
;
;       Exit-Success
;           ax = 0, MRCI server is present.
;           es:di -> MRCINFO structure
;
;
;       Exit-Failure
;           ax = 1, NO MRCI server is present.
;
;       Uses
;           ax,di,es,flags

MRCQuery  proc near
;*  Save caller's registers
        SaveReg <ax,bx,cx,dx,si,bp,ds>

;*  Check intMRCI vector before we issue the interrupt.
        xor     ax,ax                   ; Segment of interrupt vector table
        mov     ds,ax
        lds     si,ds:[intMRCI*4]       ; ds:si -> MRCI server

;*  Test if vector is plausible
```

```
        mov     ax,ds
        or      ax,ax                   ; Vector hooked?
        jz      mdr                     ;   NO, go make another check

;*  Call the server
        mov     ax,mrciDETECT           ; Function
        mov     cx,sigOLD_CX            ; Signatures for validation
        mov     dx,sigOLD_DX
        int     intMRCI                 ; Call server

        cmp     cx,sigNEW_CX            ; Signature match?
        jne     mdr                     ;   NO, go make another check

        cmp     dx,sigNEW_DX            ; Signature match?
        je      mdp                     ;   YES, have server

;*  Server not present, check for ROM based server
mdr:
        xor     ax,ax
        mov     ds,ax
        lds     si,ds:[intMRCIROM*4]    ; ds:si -> ROM MRCI server

;*  Test if vector is plausible
        mov     ax,ds
        or      ax,ax                   ; Vector hooked?
        jz      mde                     ;   NO, fail

;*  Call the server
        mov     ax,mrciDETECTROM        ; Function
        mov     cx,sigOLD_CX            ; Signatures for validation
        mov     dx,sigOLD_DX
        int     intMRCIROM              ; Call ROM server

        cmp     cx,sigNEW_CX            ; Signature match?
        jne     mde                     ;   NO, go make another check

        cmp     dx,sigNEW_DX            ; Signature match?
        jne     mde                     ;   No, fail

;*  Server is present
mdp:
        xor     ax,ax                   ; Indicate success
        jmp     short mdx               ; Go exit

;*  Set error
mde:    mov     ax,1                    ; Indicate failure

;*  Restore caller's registers and exit
mdx:    RestoreReg <ds,bp,si,dx,cx,bx,ax>
        ret
MRCQuery  endp

endm ;; DefineMRCQuery


;***    DefineMRCCompress - Macro to generate MRCCompress function
;
;       Put this macro somewhere in your code segment.  It will define the
;       MRCCompress routine, which you can then call far.  See the MRCCompress
;       header below for documentation on its behavior.

DefineMRCCompress   macro

;***    MRCCompress - Compress an uncompressed data buffer
;
;       Entry
;           ax = operation to perform:
```

```
;                   micapSTANDARD
;           cx = type of client:
;                   mcAPPLICATION - application
;                   mcSYSTEM      - file system client
;
;           ds:si -> MRCREQUEST structure
;               mr_pbSrc     - Pointer to uncompressed data buffer
;               mr_cbSrc     - Length of uncompressed data
;               mr_pbDst     - Pointer to compressed data buffer
;               mr_cbDst     - Length of compressed data buffer
;               mr_cbChunk   - Granularity of compressed data storage
;
;           es:bx -> MRCINFO structure returned by MRCQuery
;
;       Exit-Success
;           ax = 0, compress operation completed
;           ds:[si].mr_cbDst has length of compressed data in mr_pbDst.
;
;       Exit-Failure
;           Contents of mr_pbDst buffer and value of mr_cbDst are
;           undefined.
;           ax = non-zero error code:
;           MRCI_ERROR_NOT_SUPPORTED
;               Server does not support this operation.
;
;           MRCI_ERROR_BUSY
;               Server is busy with another operation.  Try again later.
;               NOTE: The most common case where this could occur is if an
;                     application calls the server while a disk cache
;                     (like SmartDrive) is writing its lazy-write queue at
;                     interrupt time to a compressed drive.  The application
;                     should try the operation again
;
;           MRCI_ERROR_BUFFER_OVERFLOW
;               The destination buffer size (mr_cbDst) was not large enough
;               to hold the compressed data.
;
;           MRCI_ERROR_NOT_COMPRESSIBLE
;               The data was not compressible, i.e., the size of the compressed
;               data would have been greater than (mr_cbDst - mr_cbChunk).
;
;       Uses
;           ax,flags

MRCCompress proc near
;*  Save caller's registers
        SaveReg <bx,cx,dx,bp,si,di,ds>

;*  Enter a Windows disk critical section.  This must be done to prevent
;   the MCRI server from being reentered if multiple VMs are making MRCI
;   calls under 386 Enhanced mode Windows.  The MRCI server sets the
;   InDOS flag, but that is not enough, as InDOS is a per-VM variable.

        push    ax                      ; NOTE: Use this exact sequence of
        mov     ax,8001h                ;       instructions because Windows
        int     2ah                     ;       expects it and will patch
        pop     ax                      ;       other code here

        call    dword ptr es:[bx].mi_pfnOperate ; Call Server

;*  Release the disk critical section.  The MRCI server is now available.

        push    ax                      ; NOTE: Use this exact sequence of
        mov     ax,8101h                ;       instructions because Windows
        int     2ah                     ;       expects it and will patch
        pop     ax                      ;       other code here
```

```
;*  Restore caller's registers and exit
        RestoreReg <ds,di,si,bp,dx,cx,bx>
        ret
MRCCompress endp

endm ;; DefineMRCCompress


;***    DefineMRCDecompress - Macro to generate MRCDeCompress function
;
;       Put this macro somewhere in your code segment.  It will define the
;       MRCDecompress routine, which you can then call far.
;       See the MRCDecompress header below for documentation on its behavior.

DefineMRCDecompress   macro


;***    MRCDecompress - Decompress an compressed data buffer
;
;       Entry
;           cx = type of client:
;                   mcAPPLICATION - application
;                   mcSYSTEM      - file system client
;
;           ds:si -> MRCREQUEST structure
;               mr_pbSrc   - Pointer to compressed data buffer
;               mr_cbSrc   - Length of compressed data
;               mr_pbDst   - Pointer to uncompressed data buffer
;               mr_cbDst   - Length of uncompressed data buffer
;
;           es:bx -> MRCINFO structure returned by MRCQuery
;
;       Exit-Success
;           ax = 0, decompress operation completed
;           ds:[si].mr_cbDst has length of uncompressed data in mr_pbDst.
;
;       Exit-Failure
;           Contents of mr_pbDst buffer and value of mr_cbDst are
;           undefined.
;           ax = non-zero error code:
;           MRCI_ERROR_NOT_SUPPORTED
;               Server does not support this operation.
;
;           MRCI_ERROR_BUSY
;               Server is busy with another operation.  Try again later.
;               NOTE: The most common case where this could occur is if an
;                     application calls the server while a disk cache
;                     (like SmartDrive) is reading ahead at interrupt time to
;                     a compressed drive.  The application should try the
;                     operation again.
;
;           MRCI_ERROR_BUFFER_OVERFLOW
;               The destination buffer size (mr_cbDst) was not large enough
;               to hold the uncompressed data.
;
;           MRCI_ERROR_BAD_MRC_FORMAT
;               The compressed data format was invalid (generally only
;               detectable as an overrun of the source buffer length, because
;               the MRC Format has no redundancy).
;               NOTE: Most software implementations will not generate this
;                     error, since it is to expensive (in time) to check for
;                     buffer overrun.  Hardware implementations may be able
;                     to check for this without performance penalty, however.
;
;       Uses
;           ax,flags
```

```
MRCDecompress proc near
;*  Save caller's registers
        SaveReg <bx,cx,dx,bp,si,di,ds>

;*  Enter a Windows disk critical section.  This must be done to prevent
;   the MCRI server from being reentered if multiple VMs are making MRCI
;   calls under 386 Enhanced mode Windows.  The MRCI server sets the
;   InDOS flag, but that is not enough, as InDOS is a per-VM variable.

        push    ax                      ; NOTE: Use this exact sequence of
        mov     ax,8001h                ;       instructions because Windows
        int     2ah                     ;       expects it and will patch
        pop     ax                      ;       other code here

        mov     ax,micapDECOMPRESS
        call    dword ptr es:[bx].mi_pfnOperate ; Call Server

;*  Release the disk critical section.  The MRCI server is now available.

        push    ax                      ; NOTE: Use this exact sequence of
        mov     ax,8101h                ;       instructions because Windows
        int     2ah                     ;       expects it and will patch
        pop     ax                      ;       other code here

;*  Restore caller's registers and exit
        RestoreReg <ds,di,si,bp,dx,cx,bx>
        ret
MRCDecompress endp

endm ;; DefineMRCDecompress
```

8. MRCI System Client Example

```
;***    MRCSYSCL.ASM - MRCI System Client EXAMPLE
;
;       MRCI version 1.00.08  02-Apr-1993
;
;       NOTE: This example is implemented assuming the code segment is
;             writeable, to simplify the exposition.
;
;       This is an example "system client" of the Microsoft Real-time
;       Compression API.  A "system client" is a device driver or TSR
;       that loads and stays resident for the lifetime of the system.
;       DoubleSpace, Flash File System 2, and network drivers are examples
;       of system clients.
;
;       By contrast, an "application client" is a program like MSBackup,
;       which loads, uses MRCI, and then terminates.
;
;       This example demonstrates the proper technique for:
;
;       1) Calling the MRCI server to get its capabilities
;       2) Calling the MRCI server to perform Standard Compress
;       3) Calling the MRCI server to perform Decompress


include utility.inc
include mrci.inc


;***********************
;* Private Definitions *
;***********************

cbBUFFER    equ     8192                ; Our compression block size
```

```
;****************
;* CODE segment *
;****************

code    segment
    assume  cs:code


;*************
;* DATA AREA *
;*************

pmi     dd  0                           ; Pointer to MRCINFO structure

buffer1 db  cbBUFFER dup (?)            ; One data buffer
buffer2 db  cbBUFFER dup (?)            ; Another data buffer

mr      MRCREQUEST    <>                ; MRCI request block


;*********************
;* USE MRCI Services *
;*********************


main    proc
        call    InitMRC                 ; Get MRCI server information
        or      ax,ax                   ; MRCI server present
        jnz     max                     ;   NO, exit

;*  Compress some data
;
;       Assume buffer1 has uncompressed data
;
        mov     ax,cs                   ; Get CS in AX, for buffer pointers
        mov     ds,ax
        mov     si,OFFSET mr            ; ds:si -> our MRCREQUEST packet

        ;* Set source pointer, length
        mov     ds:[si].mr_pbSrc.offst,OFFSET buffer1
        mov     ds:[si].mr_pbSrc.segmt,ax
        mov     ds:[si].mr_cbSrc,cbBUFFER

        ;* Set destination pointer, length
        mov     ds:[si].mr_pbDst.offst,OFFSET buffer2
        mov     ds:[si].mr_pbDst.segmt,ax
        mov     ds:[si].mr_cbDst,cbBUFFER

        ;* Set storage chunk granularity
        mov     ds:[si].mr_cbChunk,1    ; We store at byte granularity

        ;* Get MRCINFO pointer for call
        les     bx,cs:pmi

        ;* Set operation and client type
        mov     ax,micapSTANDARD
        mov     cx,mcSYSTEM

        call    MRCCompress             ; Compress!
        or      ax,ax                   ; Did compress work?
        jnz     max                     ;   NO, go exit

        ;*  buffer2 now has compressed data, and
        ;   ds:[si].mr_cbDst has the size of the compressed data.
```

```
;*  Decompress some data
;
;       Assume buffer1 has compressed data
;
        mov     ax,cs                   ; Get CS in AX, for buffer pointers
        mov     ds,ax
        mov     si,OFFSET mr            ; ds:si -> our MRCREQUEST packet

        ;* Set source pointer, length
        mov     ds:[si].mr_pbSrc.offst,OFFSET buffer1
        mov     ds:[si].mr_pbSrc.segmt,ax
        mov     ds:[si].mr_cbSrc,cbBUFFER

        ;* Set destination pointer, length
        mov     ds:[si].mr_pbDst.offst,OFFSET buffer2
        mov     ds:[si].mr_pbDst.segmt,ax
        mov     ds:[si].mr_cbDst,cbBUFFER

        ;* Get MRCINFO pointer
        les     bx,cs:pmi

        ;* Set client type
        mov     cx,mcSYSTEM

        call    MRCDecompress           ; Compress!
        or      ax,ax                   ; Did compress work?
        jnz     max                     ;   NO, go exit

        ;*  buffer2 now has decompressed data, and
        ;   ds:[si].mr_cbDst has the size of the decompressed data.

;*  Do something else interesting

        ;* Nothing else to do, this is just an example.

max:    ret

main    endp


;** BEGIN: Include code for MRCCompress/MRCDecompress here!
;
        DefineMRCCompress
        DefineMRCDecompress
;
;** END:   Include code for MRCCompress/MRCDecompress here!


;******************
;* INIT-TIME CODE *
;******************

;** BEGIN: Include code for MRCQuery here!
;
        DefineMRCQuery
;
;** END:   Include code for MRCQuery here!


;**     InitMRC - Get address of MRCI Server
;
;
;       Entry
;           none.
;
;       Exit-Success
;           ax = 0
```

```
;           pmi = pointer to MRCI server MRCINFO structure
;
;       Exit-Failure
;           ax != 0, no MRCI server present
;
;       Uses
;           ax,bx,es,flags

InitMRC proc    near
    assume  ds:nothing,es:nothing,ss:nothing
        call    MRCQuery                ; es:bx -> MRCINFO, if server present
        or      ax,ax                   ; Is server present?
        jnz     imx                     ;   NO, go fail (ax != 0)

;*  Server present, save MRCINFO pointer

        mov     cs:pmi.offst,bx ; Store offset
        mov     ax,es
        mov     cs:pmi.segmt,ax ; Store segment

        xor     ax,ax                   ; Indicate success

imx:    ret

InitMRC endp


code    ends

        end
```

9. MRCI Server Example

```
;***    MRCSRV.ASM - MRCI Server EXAMPLE
;
;       MRCI version 1.00.08  02-Apr-1993
;
;       NOTE: This example is implemented assuming the code segment is
;             writeable, for simplicity.
;
;       EXPORTED FUNCTIONS:
;           Server    - intMRCI interrupt handler
;           Operate   - Compress/Decompress entry point
;           Strategy  - DOS device driver strategy entry point
;           Interrupt - DOS device driver interrupt entry point
;
;
;       INTERNAL FUNCTIONS:
;           InitMRCServer       - Install Server
;           ShouldWeInstall     - Test if we should replace existing server
;           GetInDOSFlagPointer - Get address of InDOS flag
;           DoCompress          - Unimplemented compress routine
;           DoDecompress        - Unimplemented decompress routine


include utility.inc
include mrci.inc


;***********************
;* Private Definitions *
;***********************

MYVERSION   equ 0100h                   ; Version of this server
MRCIVERSION equ 0100h                   ; Version of MRCI supported by this server
MYVENDOR    equ 5446534Dh               ; "MSFT" (reversed for byte ordering)
```

```
MYVENDORhi  equ 5446h                   ; High word of MYVENDOR
MYVENDORlo  equ     534Dh               ; Low word of MYVENDOR
cbMAX       equ 8192                    ; Maximum compression block size

;* This server's capabilities
micapMine   equ micapSTANDARD or micapDECOMPRESS


;******************
;* PRIVATE MACROS *
;******************


;***    EnterCriticalSection - Grab server critical section
;
;       Entry
;           cx - Flag indicating whether the caller may already be inside DOS.
;                mcAPPLICATION (0) - InDOS must be 0.
;                mcSYSTEM (1)      - InDOS may have any value.
;
;       Exit-Success
;           Zero Flag Set, critical section entered
;           InDos is guaranteed to be non-zero
;           fBusy set to non-zero
;
;       Exit-Failure
;           Zero Flag clear, critical section NOT entered
;
;       Uses
;           dx,si,ds,flags

EnterCriticalSection macro  reg
        cli                             ; Grab our lock atomically

        cmp     cs:fBusy,0              ; Are we already working on something?
        jnz     ecsx                    ;   YES, return failure

;*  Make InDOS non-zero
;
;   NOTE: We check for failure here.  We should not have to, since if we
;         are busy, fBusy would be set, and we would have detected it above.
;         The other case is a non-file system client calling, but InDOS is
;         set.  This case should not happen, because if InDOS is set, no
;         non-file system client should gain control of the CPU and try to
;         call us.  So, if such a non-file system client tries this, we
;         fail the call.
;
        lds     si,cs:fpInDOS           ; ds:si -> InDOS flag
        mov     dl,ds:[si]              ; dl = InDOS flag
        or      dl,dl                   ; In DOS?
        jz      ecs10                   ;   NO, okay to take critical section

;*  InDOS is non-zero, see if caller allows this
;   Zero Flag is CLEAR (=> not zero) from falling through the "jz" above
;
;   NOTE: We allow a system client to call even if InDOS is not set, since
;         it is possible that a system client may need to operate at
;         interrupt time for "background" operations.
;
;         *** THIS IS DISCOURAGED BEHAVIOR ***
;
        jcxz    ecsx                    ;   NO, caller is not file system
.errnz mcAPPLICATION

;*  InDOS was zero, or caller was a file system

ecs10:  mov     cs:oldInDOS,dl          ; Save old InDOS flag, for later
```

```
        mov     byte ptr ds:[si],1      ; Set InDOS flag
        mov     cs:fBusy,1              ; We are now busy!
        xor     dl,dl                   ; Set zero flag to indicate success

ecsx:   ; done with macro, zero flag indicates success/failure

        sti                             ; Done being atomic
endm ;; EnterCriticalSection


;***    LeaveCriticalSection - Leave server critical section
;
;       Entry
;           cx - Flag indicating whether the caller may already be inside DOS.
;                mcAPPLICATION (0) - Application client
;                mcSYSTEM (1)      - File System client
;
;       Exit
;           InDos restored to value saved by EnterCriticalSection
;           fBusy set to zero
;
;       Uses
;           ax,si,ds,flags

LeaveCriticalSection macro  reg
        cli                             ; Release our lock atomically

;; Should assert that fBusy is set
;;      cmp     cs:fBusy,0              ; Are we already working on something?
;;      jnz     lcs20                   ;   YES, but not any longer
;;lcs10:
;;
;;      Handle case of server attempting to leave critical section
;;      without being inside critical section!
;;lcs20:

;*  Restore InDOS
        lds     si,cs:fpInDOS           ; ds:si -> InDOS flag

;; Should assert that InDOS is non-zero
;;      cmp     ds:[si],0               ; InDOS set?
;;      jz      lcs10                   ;   NO, but should have been!

        mov     al,cs:oldInDOS          ; al = old InDOS flag
        mov     ds:[si],al              ; Restore old InDOS flag

;*  Clear our busy flag
        mov     cs:fBusy,0              ; No longer busy

        sti                             ; Done being atomic
endm ;; LeaveCriticalSection


;****************
;* CODE segment *
;****************

code    segment public byte
    assume  cs:code,ds:nothing,es:nothing,ss:nothing

;;      org     0       ; Driver starts at 0

;*************
;* DATA area *
;*************

;**     Character Device Header
```

```
;

DHlink  dd      0FFFFFFFFh              ; Next device => none
DHattr  dw      8000h                   ; Simple character device
DHstrat dw      Strategy                ; Device strategy entry point
DHinter dw      Interrupt               ; Device interrupt entry point
DHname  db      'MRCISRV$'              ; Device name


;**     mrcinfo - MRCINFO to return to clients
;
;       NOTE: This structure must be writeable, so that a new Server
;             can install and hook this structure with its values.
;
;       vendor,server version,MRCI version,entry point,capabilities,cbMAX
;
mi      MRCINFO <MYVENDOR,MYVERSION,MRCIVERSION,Operate,micapMine,cbMAX>


;**     fpOldintMRCI - previous contents of intMRCI interrupt vector
;
fpOldintMRCI   dd  ?


;**     pmi - pointer to active MRCINFO structure
;
;       If we replace an old server, then that server's MRCINFO structure
;       is the one that we use.  So, we need a level of indirection to it.
;       The more common case is that we are the only server, so this
;       variable will just point to our mi.
;
pmi     dd  ?


;**     rpDOS - pointer to DOS request packet
;
rpDos   dd      ?


;**     fpInDOS, oldInDOS - pointer to DOS "InDOS" flag, old InDOS flag value
;
;       This is REQUIRED for a software-only MRCI server, to signal that
;       the server is busy, and so prevent reentrancy.  Whenever the server
;       is about to become non-reentrant (starting a compress/decompress
;       operation, for example), it must ensure that InDOS is set.
;
;   ATTENTION--BEGIN
;
;       Furthermore, ALL callers of MRCI *must* ensure that the Windows
;       Disk Critical Section is held *prior* to calling MRCI.  This prevents
;       programs in different Windows virtual machines from reentering MRCI
;       and encountering the MRCI_ERROR_BUSY error.
;
;       Block device drivers, and system extensions like DBLSPACE.BIN, do not
;       need to grab this critical section because the MS-DOS kernel has
;       already done so before giving them control.
;
;       But TSRs and application programs *must* grab this critical section
;       prior to calling MRCI.
;
;   ATTENTION--END
;
;       The following worst-case scenario demonstrates why this is needed,
;       and why returning *Busy* (see fBusy below) is of no help:
;
;       1)  MSBackup calls MRCI server to do a decompress operation
;       2)  While MRCI server is busy, user brings up a TSR (e.g., SideKick)
```

```
;       3)  SideKick does a DOS file open and read on a compressed drive
;       4)  DOS FAT file system calls DoubleSpace
;       5)  DoubleSpace calls MRCI server, which returns busy
;
;       At this point, the system is deadlocked!  DoubleSpace has no choice
;       but to try the request again, or return some sort of disk error!
;
;       By setting the DOS InDOS flag, however, the MRCI server signals to
;       TSRs (and Windows) that it is entering a critical section, and so
;       a well-written TSR will not try to make DOS calls or MRCI calls.

fpInDOS     dd      ?   ; Pointer to DOS InDOS flag
oldInDOS    db      ?   ; Old InDOS flag value


;**     fBusy - TRUE if we are compressing/decompressing
;
;       Setting the DOS InDOS flag will protect us in most cases, but
;       having this flag lets us prevent being reentered by an interrupt-
;       time caller.
;
fBusy   db      ?   ; 0 => not busy, !0 => busy


;********
;* CODE *
;********


;***    Strategy - Device driver strategy entry point
;
;       Entry
;           es:bx - pointer to DOS request packet
;
;       Exit
;           es:bx saved in data:rpDos
;
;       Uses
;           None
;
Strategy    proc    far
    assume  cs:code,ds:nothing,es:nothing
        mov     cs:rpDos.offst,bx       ; Store request packet pointer
        mov     cs:rpDos.segmt,es
        ret
Strategy    endp


;***    Interrupt - Device driver interrupt entry point
;
;       Entry
;           rpDOS - pointer to DOS request packet saved by Strategy function
;
;       Exit
;           ???
;
;       Uses
;           None
;
Interrupt   proc    far
        SaveReg <ax,bx,cx,dx,si,di,ds,es>

        push    cs                      ; Make access to driver segment faster
        pop     ds
    assume  ds:code

        les     bx,rpDos                ; es:bx -> DOS request packet
```

```
        mov     al,es:[bx].rpFunction   ; al = function
        cmp     al,DEVINIT              ; Init call?
        mov     ax,STERR+03h            ; Assume ERROR+"Unknown command"
        jne     itrx                    ;   NO, fail call

;*  Init device

        call    GetInDOSFlagPointer     ; Needed for critical section calls

        SaveReg <bx,es>
        call    InitMRCServer
        RestoreReg <es,bx>
        mov     cx,OFFSET cs:EndOfResidentCode ; Assume we are installed
        or      ax,ax                   ; Are we installed?
        jz      itr10                   ;   YES, set break address, note that
                                        ;   ax=status=fine!

;*  Did not install MRCI server
        xor     cx,cx                   ;   NO, make break address small
        mov     ax,STERR                ; What error code should we use?

;*  Set break address
;
;   cx = break address
;   ax = status code

itr10:  mov     es:[bx+14].offst,cx
        mov     es:[bx+14].segmt,cs

;*  es:bx -> DOS request packet
;   ax = status code

itrx:   or      ax,STDONE               ; Set DONE bit in status
        mov es:[bx].rpStatus,ax         ; Store status
        RestoreReg <es,ds,di,si,dx,cx,bx,ax>
        ret
Interrupt   endp


;***    Server - MRCI server interrupt entry point
;
;       Entry
;           mov ax,MRCI function  ; Set function
;           int intMRCI           ; Call us here
;
;       Exit
;           See MRCI spec.  IRET to return.
;
;       Uses
;           AX, DI, ES, flags

Server  proc
    assume  ds:nothing,es:nothing,ss:nothing

;*  Verify that caller is asking for an MRCI server

        cmp     cx,sigOLD_CX            ; Signature match?
        je      srv10                   ;   YES, keep testing

        jmp     dword ptr cs:[fpOldintMRCI] ; Chain to previous hooker

srv10:  cmp     dx,sigOLD_DX            ; Signature match?
        je      srv20                   ;   YES, caller wants us

        jmp     dword ptr cs:[fpOldintMRCI] ; Chain to previous hooker

;*  Dispatch MRCI function
```

```
srv20:  cmp     ax,mrciDETECT           ; Detect call?
        je      dms10                   ;   NO, continue

;*  mrciQUERY call

        push    cs
        pop     es
        mov     di,OFFSET mi            ; es:di -> our MRCINFO structure

;*  Update CX/DX signatures so caller knows we are an MRCI server

                                ;   entry: cx='ab'  dx='cd'
        xchg    ch,cl                   ; cx='ba'  dx='cd'
        xchg    dh,dl                   ; cx='ba'  dx='dc'
        xchg    dx,cx                   ; cx='dc'  dx='ba'

        xor     ax,ax                   ; Indicate success
        iret                            ; Return to caller

dms10:  mov     ax,1                    ; Indicate failure

;*  EmptyIRET - address for fpOldintMRCI if intMRCI was unhooked.
;
;   To improve performance of Server when it chains a non-intMRCI request,
;   if we are the first hooker of intMRCI, we store the pointer to this
;   IRET.  This removes the need for Server to check that the fpOldintMRCI
;   value is valid, to prevent from INTing into hyperspace.
;
EmptyIRET:
        iret
Server  endp


;***    Operate - MRCI server operation entry point
;
;       Handle Compress/Decompress requests.
;
;       Entry
;               ax = Requested operation (micapXXXX)
;               cx = File System Caller flag
;                       mcAPPLICATION (0) - Application client
;                       mcSYSTEM (1)      - File System client
;           ds:si -> MRCREQUEST packet
;
;       Exit-Success
;           ax = 0
;           See MRCI spec for details on returned values.
;
;       Exit-Failure
;           See MRCI spec for details on returned values.
;
;       Uses
;           All (but CS:IP, SS:SP)

Operate proc    far
    assume  ds:nothing,es:nothing,ss:nothing

        and     ax,micapMine            ; Is this an operation we support?
        jz      opre                    ;   NO, fail request

;*  Make sure we are not being reentered

        EnterCriticalSection cx         ; Grab critical section
        jc      opre1                   ;   Already grabbed, go fail

        SaveReg <cx>                    ; Save fFileSystemCaller flag
```

```
;*  Dispatch operation
;
;   NOTE: We assume Standard Compress is the most common operation,
;         so we optimize the flow to reduce jumps for that operation.
;
        cmp     ax,micapSTANDARD        ; Standard compress?
        jz      opr10                   ;   YES, go do it

        cmp     ax,micapDECOMPRESS      ; Decompress?
        jnz     opre                    ;   NO, invalid operation (user had
                                        ;   more than one bit set!)
;*  Do Decompress

opr20:  call    DoDecompress
        jmp     short oprLCS

;*  Do Standard Compress

opr10:  call    DoCompress

;*  Release critical section
oprLCS: RestoreReg <cx>                 ; Get fFileSystemCaller flag
        LeaveCriticalSection cx         ; Leave critical section

        ret                             ; Return status


opre:   mov     ax,MRCI_ERROR_NOT_SUPPORTED
        ret

opre1:  mov     ax,MRCI_ERROR_BUSY
        ret
Operate endp


;***    DoCompress - Unimplemented compress routine
;
;
DoCompress  proc    near
DoCompress  endp


;***    DoDecompress - Unimplemented decompress routine
;
;
DoDecompress  proc    near
DoDecompress  endp


;******************
;* INIT-TIME CODE *
;******************

EndOfResidentCode  label  byte  ; Truncate device here after init is complete


;** BEGIN: Include code for MRCQuery here!
;
        DefineMRCQuery
;
;** END:   Include code for MRCQuery here!


;**     InitMRCServer - Initialize our MRCI Server
;
;       If an MRCI server IS NOT present, then we install our server.
```

```
;
;       If an MRCI server IS present, then we get the pointer to its
;       MRCINFO structure, and check to see if we are "better" than the
;       present server.  If so, then we install, and edit the old
;       server's MRCINFO structure so that its clients can call us.
;
;       Entry
;           none.
;
;       Exit-Success
;           ax = 0
;           Server installed
;           pmi = pointer to MRCI server MRCINFO structure
;
;       Exit-Failure
;           ax = 1, another, better server already present
;
;       Uses
;           ax,bx,si,ds,es,flags

InitMRCServer   proc    near
    assume  ds:nothing,es:nothing,ss:nothing

        call    MRCQuery                ; Check for MRCI server presence
        or      ax,ax                   ; Is server present?
        jz      ims20                   ;   NO, install ourselves

;*  Existing Server present
;
;   es:di -> old server's MCRINFO structure
;
;   We need to copy our MRCINFO structure over the old server's MCRINFO
;   structure, so that clients of the previous server that call the
;   server indirectly using the old MRCINFO structure will call us,
;   instead.
;
;   NOTE: We are guaranteed that no one is using the old server right now,
;         because we have the CPU.
;
;         So, we disable interrupts while we update the old MRCINFO structure.

        ;; dbgPrint 'Existing server present'

        call    ShouldWeInstall
        or      ax,ax                   ; Do install?
        jz      ims10                   ;   YES, go install

        ret                             ;   NO, return "did not install"

;*  Copy our MRCINFO to old server's MRCINFO

ims10:  cli                             ; Protect old MRCINFO from being
                                        ; in an inconsistent state.
        push    cs
        pop     ds
    assume  ds:code

        mov     si,OFFSET mi            ; ds:si -> our MRCINFO structure
        mov     cx,SIZE mi
        cld
        rep     movsb                   ; Copy our MRCINFO to old one

;*  Fall through to hook interrupt chain.
;
;   NOTE: Interrupts remain off, until we have hooked the interrupt chain.
;         We have to leave the old server in the chain, so that it can
;         chain on to previous non-MRCI hookers.
```

```
;*  Hook into intMRCI chain

ims20:  xor     ax,ax                   ; Segment of interrupt vector table
        mov     ds,ax
    assume  ds:nothing
        mov     si,intMRCI*4            ; ds:si -> intMRCI vector

        cli                             ; Editing the interrupt vector table

        mov     ax,OFFSET Server
        xchg    ds:[si].offst,ax        ; Set offset, ax = old offset
        mov     bx,cs
        xchg    ds:[si].segmt,bx        ; Store segment, bx = old segment

        or      ax,ax                   ; Old pointer valid?
        jnz     ims30                   ;   YES, go save

        mov     ax,OFFSET EmptyIRET     ;   NO, point at on of our IRETs
        mov     bx,cx

ims30:  mov     cs:fpOldintMRCI.offst,ax ; Save old offset, for chaining
        mov     cs:fpOldintMRCI.segmt,bx ; Save old segment, for chaining

        sti                             ; Done editing the intvec table

        ret
InitMRCServer   endp


;***    ShouldWeInstall - check if our server should supercede another server
;
;       We can install only if all of the following are true:
;           1) Our capabilities are at least as good as present server
;           2) Our MRCI version is at least as high
;           3) If the vendor is the same, our vendor server version must
;              be greater than the present server.  If the vendor does not
;              match, then we cannot compare vendor versions, so we assume
;              we should install.
;
;       Entry
;           es:di -> MRCINFO structure of current installed server
;
;       Exit-Success
;           ax = 0, we should install
;
;       Exit-Failure
;           ax = 1, we should not install
;           Existing driver is more capable than we are.
;
;       Uses
;           ax,bx,flags

ShouldWeInstall proc    near
;*  Our capabilities must be at least as good as present server

        mov     ax,es:[di].mi_flCapability ; server capabilities
        and     ax,not micapMine        ; ax = server caps that I do not have
        or      ax,ax                   ; Does the server have caps I do not?
        jnz     swie                    ;   YES, do NOT install

;*  We are at least as good on capabilities, so...
;*  Compare MRCI versions

        mov     ax,es:[di].mi_wMRCIVersion ; ax = present server MRCI version
        cmp     ax,MRCIVERSION          ; Are we at least as good as server?
        ja      swie                    ;   NO, do not install
```

```
        jb      swi10                   ; We are better than server, install

;*  We support the same MRCI version, so...
;*  See if we supercede the old <vendor,vendor version>

        mov     ax,es:[di].mi_lVendor.loword
        cmp     ax,MYVENDORlo           ; Match low word?
        jne     swi10                   ;   NO, DO install

        mov     ax,es:[di].mi_lVendor.hiword
        cmp     ax,MYVENDORhi           ; Match high word?
        jne     swi10                   ;   NO, DO install

        mov     ax,es:[di].mi_wVendorVersion
        cmp     ax,MYVERSION            ; Am I a newer driver version?
        jae     swie                    ;   NO, do NOT install

;*  Tests pass -- we should install

swi10:  xor     ax,ax                   ; Indicate DO install
        ret

swie:   mov     ax,1                    ; Indicate do NOT install
        ret

ShouldWeInstall endp


;***    GetInDOSFlagPointer - Get address of InDOS flag
;
;       Entry
;           none
;
;       Exit
;           fpInDOS = pointer to DOS InDOS flag
;
;       Uses
;           ax,flags

GetInDOSFlagPointer proc    near
        SaveReg <bx,es>
        mov     ah,34h
        int     21h                     ; Get In DOS flag

        mov     cs:fpInDOS.offst,bx
        mov     cs:fpInDOS.segmt,es
        RestoreReg <es,bx>
        ret

GetInDOSFlagPointer endp

code    ends

        end
```