MMX™ Technology Emulator and Macro Package User's Guide

Order Number: 654905-001

Revision	Revision History	Date
-002	Original Issue	4/96

Information in this document is provided in connection with Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products.

Intel Corportation retains the right to make changes to these specifications at any time, without notice.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:
Intel Corporation
P.O. Box 7641
Mt. Prospect IL 60056-764
or call 1-800-879-4683

*Other brands and names are the property of their respective owners.

Contents

Examples

Figures

Overview

1

About the Emulator

The MMX™ Technology Emulator is a tool designed to emulate MMX instructions on Intel Architecture (IA) processors that do not support MMX technology. The emulator enables application developers to write their applications using MMX code on any IA processor. The emulator behaves as an exception handler by handling MMX technology instructions that generate invalid opcode exceptions. When an invalid opcode occurs on IA processors, the emulator decodes the MMX technology instructions, emulates them, and continues to execute the program at the next instruction.

The emulator supports Win32*, Console32, Win16 and VxD application development environments. The emulator provides different levels of support for Windows 95 and Windows NT operating systems.

Methods of Emulation

Emulation can be performed in one of two methods:

- Virtual Device Driver (VxD) method
- C exception method

The VxD method of emulation is supported only on Windows 95 operating systems. The C exception method of emulation is supported on both Windows 95 and Windows NT operating systems.

The emulation process is handled differently by the two methods of emulation. Using the VxD method, the emulator is incorporated in the operating system and is external to the application. Using the C exception method, the emulator is incorporated into the application in the development process.

The VxD method of emulation offers MMX technology debugging support that is provided by the Intel Register Viewing Tool (RVT) (see Appendix B, for detail on the RVT). The RVT enables you to view and modify the contents of the MMX registers. The RVT runs only on Windows 95. The RVT is not accessible when using the C exception method of emulation.

About the Macro Package

The MMX Technology Macro Package is an include file provided with the emulator. It enables you to use MMX instructions when writing your assembly source code. This file includes Microsoft Macro Assembler macros that embed MMX technology instructions into an assembly object file. This macro generates the opcodes for the MMX technology instructions.

The macro package requires Microsoft Macro Assembler, Version 6.11d or later versions. MASM6.11d can be obtained from the Microsoft Windows NT Device Driver Kit (DDK) or directly from Microsoft.

Package Contents

The MMX Technology Emulator and Macro Package, contains the following files:

- Emulation files:
  - `viammxd.386`
  - `mm.c`
- Macro package include file: `iammx.inc`
- C exception include file: `mmuser.h`
- Register Viewing Tool: `rvt.exe`

- Three examples located in the `examples` directory:
  `win32tst`: Sample of a Win32 application with MMX instructions.
  `win16tst`: Sample of a Windows 16-bit application with MMX instructions.
  `vmmxtst`: Sample of a Virtual Device Driver with MMX instructions.

Installing the Emulator

The emulator is automatically installed when you run the Performance Tools CD setup. It will continue to run automatically after rebooting your system. In cases where the emulator is not installed in your system, add the following line to your `system.ini` file:

```
[386Enh]
    device={path to viammxd.386}\viammxd.386
```

If you have installed a previous copy of this emulator on your system, remove this line.

About this Manual

This manual steps you through the emulation process used to develop applications using MMX instructions in the different development environments. As a prerequisite, you should be familiar with the MMX technology, C or C++, and assembly-language programming.

- Chapter 1, Overview
- Chapter 2, Win32 and CONSOLE32 Development Environments, describes how to emulate applications in the Win32 and CONSOLE32 development environments.
- Chapter 3, Win16 Development Environment, describes how to emulate applications in the Win16 development environment.
- Chapter 4, VxD Development Environment, describes how to develop Virtual Device Drivers using MMX instructions.
- Appendix A, Writing MMX Code in C, describes the process for writing MMX code in C using in-line assembly.
- Appendix B, Register Viewing Tool (RVT), describes the features and functions of the Register Viewing Tool used to view MMX register content in the debugging process.

Related Documentation

Refer to the documents below for more information about the Intel Architecture and IA MMX technology.

These documents are included in the Intel Performance Tools for MMX Technology CD:

- *Intel Architecture MMX™ Technology Application Notes.*
- *Intel Architecture MMX™ Technology Programmer's Reference Manual.* Intel Corporation, order number 243007.
- *Intel Architecture MMX™ Technology Computer Based Training (CBT).*
- *Intel Reference C Compiler User's Guide for Win32 Systems.* Intel Corporation, order number 34329
- *Pentium® Processor Computer Based Training (CBT)*

Other information about the Intel Architecture is available from Intel:

- *Pentium® Processor Family User's Manual, Volumes 1, 2, and 3*. Intel Corporation, order numbers 241428, 241429, and 241430.
- *Pentium® Pro Processor Developer's Manual, Volumes 1, 2, and 3*. Intel Corporation, order numbers 242690, 242691, and 242692.

Win32 and CONSOLE32 Development Environments

2

This chapter describes how to develop applications with MMX instructions for Win32 environments (Windows 95 and Windows NT platforms), and for CONSOLE32 environments, using the emulator and macro package.

The Win32 development environment is used for building 32-bit applications that run on Windows 95 and Windows NT operating systems.

The Console32 development environment is used for building 32-bit command-line applications that run on Windows 95 and Windows NT operating systems.

Two methods of emulation can be used in the Win32 and CONSOLE32 development environments (see section "Methods of Emulation" in Chapter 1) .

- emulation using Windows Virtual Device Driver (VxD)
- emulation using C exceptions

Windows 95 supports both VxD emulation and emulation using C exceptions.

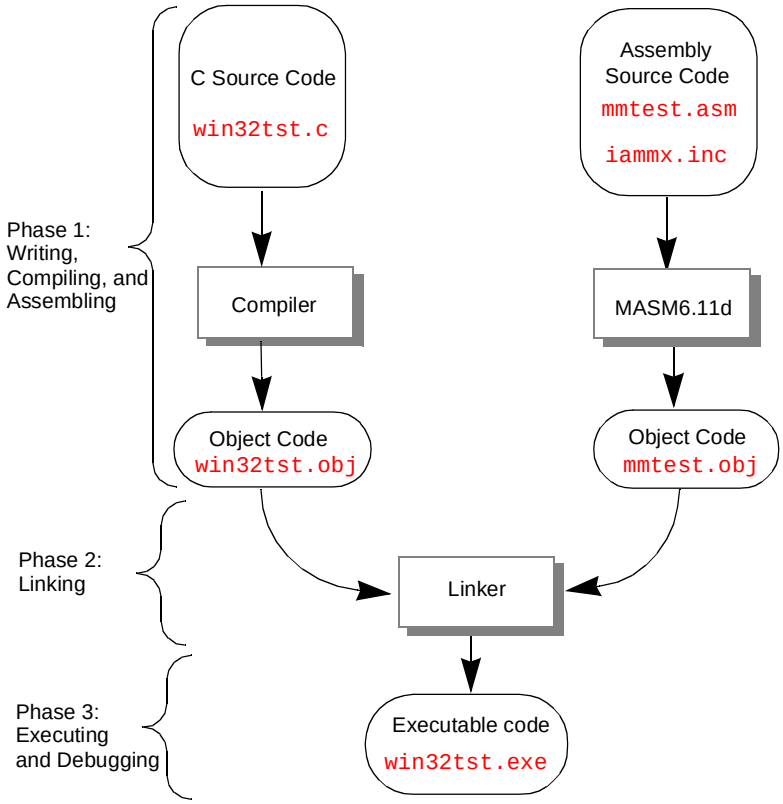
Windows NT only supports emulation using C exceptions.

VxD Method of Emulation Development Process

This section steps you through the process of building Windows 32-bit applications or 32-bit command-line applications with MMX code, using VxD emulation.

Figure 2-1 illustrates the development process for 32-bit applications and 32-bit command-line applications with MMX instructions, using this method of emulation.

Figure 2-1 Application Development Process Using VxD Emulation



Sample Application

A sample application using this method of emulation can be found in the examples directory `win32tst`, provided with this package. A sample file of the assembly source code (`mmtest.asm`) is located in the `include` directory.

The filenames that are given as examples in the development steps and in Figure 2-1, correspond to the filenames used in the sample application.

Writing, Compiling, and Assembling the Code

Follow these steps to write, compile, and assemble your code. These steps correspond to Phase 1 in Figure 2-1.

1. Write or open your C language source files and assembly language source files.

   The C program file (sample filename: `win32tst.c`) is a 32-bit application that calls a function with MMX instructions from the assembly file. Example 2-2 is an example of C source code with an MMX function call and in-line assembly code.
2. Include the macro package file (`iammx.inc`) in your assembly file (sample filename: `mmtest.asm`). This enables the MASM 6.11d to assemble the MMX instructions. Example 2-1 is an example of assembly source code with MMX instructions.
3. Assemble the assembly code by running the MASM6.11d or later versions to generate the object files (sample filename: `mmtest.obj`).
4. Compile the C code using any compiler to generate the object files (sample filename: `win32tst.obj`). If you are using the Intel Reference C Compiler or MSVC++ Compiler, Version 4.1, see Appendix A "Writing MMX Code in C", for instructions.

Example 2-1 Assembly Source Code with MMX instructions

```
file: mmtest.asm
include iammx.inc
MMXFuntion Proc
                movq    mm0,mm1
                movd    mm0,eax
                emms
                ret
MMXFunction Endp
```

Example 2-2 C Source Code with an MMX Function Call and In-line Assembly

```
file: main.c
main()
{
        //Call an MMX function call
          MMXFunction();
        //In-line assembly MMX instructions
        _asm{
                pxor    mm0,mm0
                mov     eax,0x2345
                movd    mm0,eax
                emms
        }
}
```

Linking the Object Files

Link the object files or libraries (sample filenames: `win32tst.obj and mmtest.obj`) using the Microsoft Visual C++ Linker to produce an executable (sample filename: `win32tst.exe`). The MMX instructions are embedded in the VxD executable.

This step corresponds to Phase 2 of Figure 2-1.

Executing

When you run applications with MMX code, the MMX instructions generate invalid opcodes on IA processors that do not support MMX technology. The invalid opcodes are trapped and handled directly by the VxD emulator. The VxD emulator emulates the MMX instructions by performing operations equivalent to those performed by scalar instructions.

This step corresponds to Phase 3 of Figure 2-1.

Debugging

Currently the Intel DB32 Debugger and most of the latest versions of compilers from new vendors provide the option to view the content of the MMX registers. MSVC 4.1 debugging environment has indirect support for MMX register display, which displays the register values in Hex only in a 64-bit format.

To view the MMX register content in the MSVC 4.1 debugging environment, open the Watch Window, and type: `stn,x`, (where `n` can be register number 0 through to 7).

The Register Viewing Tool (RVT) is a new tool that was designed to enable you to display and modify the contents of the MMX registers during the debugging process (see Appendix B for more detail on the RVT). Use the RVT provided in the Performance Tools package if your debugger does not have this capability.

The RVT is only accessible on Windows 95.

You can run the RVT during a regular debugging session from any debugger.

To invoke the RVT do the following:

From the Windows Explorer, run `bin\rvt.exe` from the directory in which the Performance Tools files are installed.

The RVT displays the contents of the MMX registers after emulation. You can view and modify the contents of the MMX registers while stepping through the code, line by line.

This step corresponds to Phase 3 of Figure 2-1.

C Exceptions Method of Emulation Development Process

C exceptions are constructs provided by the Microsoft C Compiler that handle unexpected events and error conditions. The emulator uses the `_try\_except` construct to trap invalid opcodes and is built into the executable file to emulate MMX instructions.

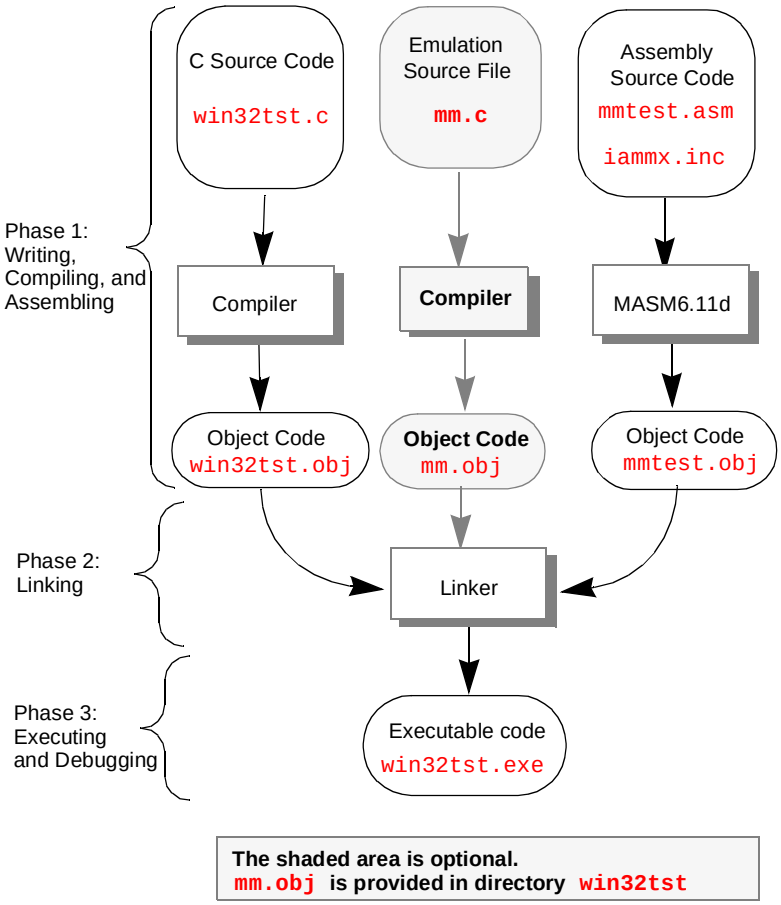
An application that uses C exceptions to trap and emulate MMX instructions can run either on Windows 95 or Windows NT operating systems.

This section steps you through the process of building Windows 32-bit applications or 32-bit command-line applications with MMX code, using the C exception method of emulation.

When using the C exception method of emulation, disable the VxD emulator by removing the `viammxd.386` file from your `system.ini` file if it exists. You can also use the Register Viewer Tool (`rvt.exe`) to disable the VxD emulation (See Appendix B for detail on the RVT).

Figure 2-2 illustrates the development process for applications with MMX instructions, using this method of emulation.

Figure 2-2 Application Development Process Using C Exceptions



Sample Application

A sample application using this method of emulation can be found in the examples directory `win32tst`, provided with this package. A sample file of the assembly source code (`mmtest.asm`) is located in the `include` directory.

The filenames that are given as examples in the development steps and Figure 2-2, correspond to the filenames used in the sample application.

Writing, Compiling, and Assembling the Code

Follow these steps to write, compile, and assemble your code. These steps correspond to Phase 1 in Figure 2-2.

1. Write or open your C language source file and assembly language source file.
2. Include the macro package file (`iammx.inc`) in your assembly file (`mmtest.asm`). This enables the MASM 6.11d to assemble the MMX instructions. Example 2-3 is an example of assembly source code with MMX instructions.
3. Assemble the assembly code by running the MASM6.11d or later versions to generate the object files (sample filename: `mmtest.obj`).
4. Include the `mmuser.h` file in your C source code.
5. Place a function call within the `_try/_except` block that calls an assembly routine with MMX instructions. Example 2-4 is an example of C source code with an MMX function call and in-line assembly code.
6. At the `_except` statement line type the following command: `MMX_HANDLE_EXCEPTION`.
7. Include the emulation file `mm.c` located in the `include` directory of this package in your project.
8. Define `WIN_APP` at the compiler command line or from the setting options when compiling your C source file.
9. Compile the C code using any compiler to generate the object files (sample filename: `win32tst.obj`). If you are using the Intel Reference C Compiler or MSVC++ Compiler Version 4.1, see Appendix A "Writing MMX Code in C".

Example 2-3 Assembly Source Code with MMX instructions

```
file: mmtest.asm
include iammx.inc
MMXFuntion Proc
              movq   mm0,mm1
              movd   mm0,eax
              emms
              ret
MMXFunction Endp
```

Example 2-4 C Source Code with an MMX Function Call and In-line Assembly

```
file: main.c
#include “mmuser.h”
_try {
       //Call an MMX function call
         MMXFunction();
       //In-line assembly MMX instructions
       _asm{
              pxor   mm0,mm0
              mov    eax,0x2345
              movd   mm0,eax
              emms
       }
}_except (MMX_HANDLE_EXCEPTION){}
}
```

Linking the Object Files

Link the object files or libraries using the Microsoft Visual C++ Linker to produce an executable file. The MMX instructions are embedded in the executable.

This step corresponds to Phase 2 of Figure 2-2.

Executing

When you run your code and an MMX instruction executes (invalid opcode), the operating system transfers the exception to the `_except {}` block. The MMX_HANDLE_EXCEPTION macro then calls functions in the emulator source file (`mm.c`) that decode and execute MMX instructions.

This section corresponds to Phase 3 of Figure 2-2.

Debugging

Use your development tool debugger to debug your MMX application. The Register Viewer Tool is not accessible on Windows NT.

To view the MMX register content, open the Variable WatchWindow to view the content of **`pMmReg[0-7 array]`**. You can only view the contents of the MMX registers, you cannot modify them.

This section corresponds to Phase 3 of Figure 2-2.

WATCOM Support for 32-bit Application Development

Watcom is releasing a new version of tools with MMX technology support, the current Watcom working environment does not have macro package support. The current versions of Watcom tools can be used with the VxD method of emulation on Windows 95, with the exception of the WASM (WATCOM Assembler) which does not provide MMX technology support. Assembling your MMX code can only be done using the MASM6.11d (Microsoft Assembler).

The WATCOM tools can be used for the following:

- Compiling C code
- Linking
- Debugging

The WATCOM debugger can be used in the C exceptions method of emulation on Windows NT. For settings, follow these instructions:

1. From the WATCOM IDE (Integrated Development Environment) in the Options menu, select Debug switches.
2. Choose the TrapFile option and in the text field type the following command: `std;2`. By default the WATCOM debugger traps the exceptions before the user. The `std;2` command enables the user to trap the exceptions first.
3. From the IDE, run the debugger.
4. For single stepping through your code; from the Run menu choose the option: Next Sequential (x), or alternatively press the shortcut key X.

Win16 Development Environment

3

This chapter describes how to develop 16-bit applications with MMX instructions in the Win16 environment, using the emulator and macro package.

The Win16 environment only supports emulation for MMX technology applications running on Windows 95 operating systems, using the VxD method of emulation. You cannot use the C exception method of emulation in this development environment.

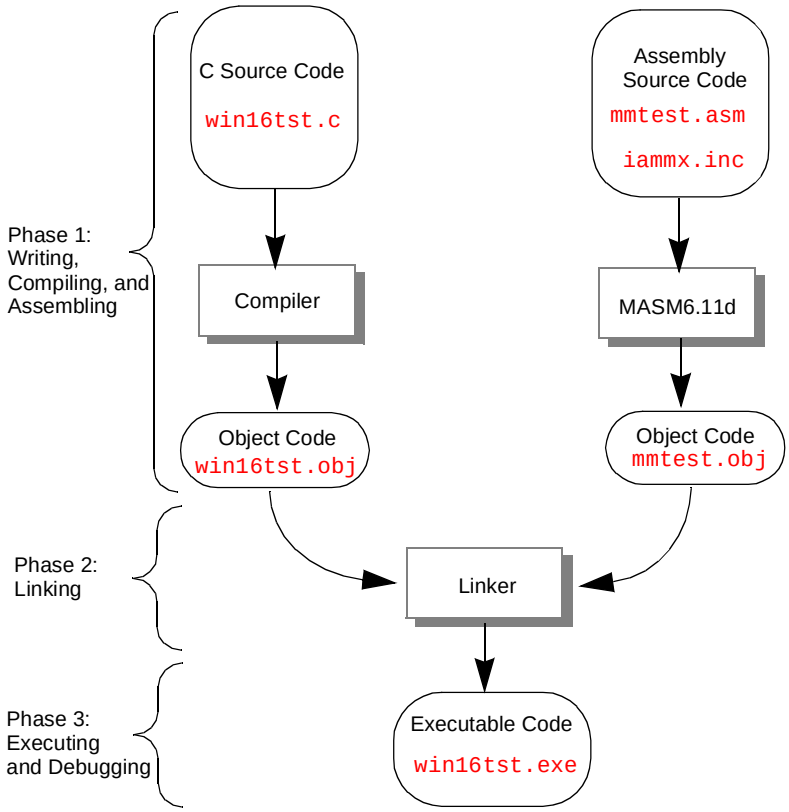
Win16 development environment is used to develop 16-bit applications for Windows 3.1.

VxD Method of Emulation Development Process

This section steps you through the process of building 16-bit applications with MMX code, using VxD emulation.

Figure 3-1 illustrates the development process for 16-bit applications with MMX instructions, using this method of emulation.

Figure 3-1 Application Development Process Using VxD Emulation



Sample Application

A sample application using this method of emulation can be found in the examples directory `win16tst`, provided with this package. A sample file of the assembly source code (`mmtest.asm`) is located in the `include` directory.

The filenames that are given as examples in the development steps and in Figure 3-1, correspond to the filenames used in the sample application.

Writing, Compiling, and Assembling the Code

Follow these steps to write, compile, and assemble your code. These steps correspond to Phase 1 in Figure 3-1.

1. Write or open your C language source file and assembly language source file.

   The C program file (sample filename: `win16tst.c)` is a Windows 16-bit application that calls a function with MMX instructions from the assembly file. Example 3-2 is an example of C source code with an MMX function call and in-line assembly code with MMX instructions.
2. Define `APP_16BIT` at the assembly command line.
3. Include the macro package file (`iammx.inc`) in your assembly file (sample filename: `mmtest.asm`). This enables the MASM6.11d to assemble the MMX instructions. Example 3-1 is an example of assembly source code with MMX instructions.
4. Assemble the assembly code by running the MASM6.11d or later versions to generate the object files (sample filename: `mmtest.obj`).
5. Compile the C code using a compiler that generates 16-bit code to generate the object files (sample filename: `win16tst.obj`). If you are using the Intel Reference C Compiler or MSVC++ Compiler Version 4.1, see Appendix A "Writing MMX Code in C", for further instructions.

Example 3-1 Assembly Source Code with MMX Instructions

```
file: mmtest.asm
APP_16BIT EQU 1
include iammx.inc
MMXFuntion Proc
              movq   mm0,mm1
              movd   mm0,ax
              emms
              ret
MMXFunction Endp
```

Example 3-2 C Source Code with an MMX Function Call and In-line Assembly

```
file: main.c
main()
{
       //Call an MMX function call
         MMXFunction();
       //In-line assembly MMX instructions
       }
}
```

Linking the Object Files

Link the object files or libraries (sample filenames: `win16tst.obj` and `mmtest.obj`) using a linker that supports 16-bit development environments to produce an executable (sample filename: `win16tst.exe`). The MMX instructions are embedded in the VxD executable.

This step corresponds to Phase 2 of Figure 3-1.

Executing

When you run applications with MMX code, the MMX instructions generate invalid opcodes on IA processors that do not support MMX technology. The invalid opcodes are trapped and handled directly by the VxD emulator. The VxD emulator emulates the MMX instructions by performing operations equivalent to those performed by scalar instructions.

This step corresponds to Phase 3 of Figure 3-1.

Debugging

Use a 16-bit debugger (e.g., Microsoft Visual C++1.v5x) to debug your code. To view the MMX register contents use the Register Viewing Tool (RVT) (see Appendix B for more detail on the RVT).

You can invoke the RVT during a regular debugging session from any 16-bit debugger.

To invoke the RVT, do the following:

From the Windows Explorer, run `bin\rvt.exe` from the directory in which the Performance Tools files are installed.

The RTV displays the contents of the MMX register after emulation. This enables you to view and modify the MMX register content while stepping through the code, line by line.

This step corresponds to Phase 3 of Figure 3-1.

VxD Development Environment

4

This chapter describes how to develop Virtual Device Drivers with MMX instructions, using the emulator and macro package.

A Virtual Device Driver (VxD) is a 32-bit ring 0 driver.

VxD development is only supported by the VxD method of emulation and only uses assembly language source code. The development process is similar to that as the Win32 development environment, without processing the C source code.

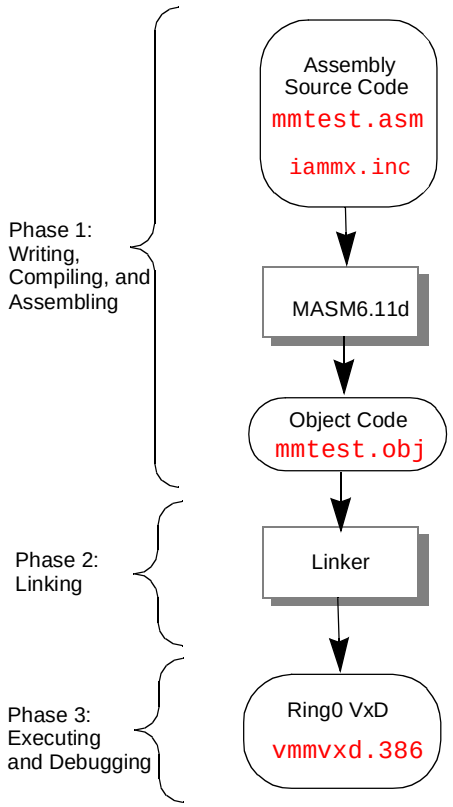
An application that uses VxD emulation can only run on Windows 95.

Development Process

This section steps you through the process of building Virtual Device Drivers with MMX code, using VxD emulation.

Figure 4-1 illustrates the development process for developing a Virtual Device Driver with MMX instructions, using this method of emulation.

Figure 4-1 Development Process for Virtual Device Drivers using VxD Emulation



Sample Application

A sample application using this method of emulation can be found in the examples directory `vmmxtst`, provided with this package. A sample file of the assembly source code (`mmtest.asm`) is also located in the `vmmxtst` directory.

The filenames that are given as examples in the development steps and Figure 4-1, correspond to the filenames used in the sample application.

Writing and Assembling the Code

Follow these steps to write and assemble your code. These steps correspond to Phase 1 in Figure 4-1.

1. Write or open your assembly language source file.
2. Include the macro package file (`iammx.inc`) in your assembly file. This enables the MASM 6.11d to assemble the MMX instructions.
3. Assemble the assembly code by running the MASM6.11d or later versions to generate the object files (sample filename: `mmtest.obj`).

Linking the Object Files

Link the object files or libraries using the standard tools, for example, Windows 3.1 DDK or the latest version of Windows 95 VxD DDK, to produce a Virtual Device Driver. The MMX instructions are embedded in the VxD executable.

This step corresponds to Phase 2 of Figure 4-1.

Executing

Install the VxD emulator before executing. To install the VxD emulator, add the following line to your `system.ini` file:

```
[386Enh]
     device={path to viammxd.386}\viammxd.386
```

MMX instructions generate invalid opcodes on IA processors that do not support MMX technology. The invalid opcodes are trapped and handled directly by the VxD emulator. The VxD emulator emulates the MMX instructions by performing operations equivalent to those performed by scalar instructions.

This step corresponds to Phase 3 of Figure 4-1.

Debugging

Soft-Ice/W*, Version 2.0b or later versions have debugging support for MMX technology using VxD emulation. The debugger can be obtained directly from Nu-Mega Technologies, Inc. This tool allows you to view and modify the MMX register contents after emulation.

This step corresponds to Phase 3 of Figure 4-1.

Writing MMX Code in C

A

Currently there are two compilers that have added support for MMX technology:

- The Intel Reference C Compiler for Win32 Systems
- The Microsoft Visual C++ Compiler, Version 4.1

The Intel Reference C Compiler provides support for MMX technology by enabling you to write MMX code using in-line assembly support or using MMX intrinsics. See the Intel Reference C Compiler User's Guide for Win32 Systems, included in this CD, for complete details and operating instructions.

The Microsoft Visual C++ Compiler, Version 4.1 provides in-line assembly support. It does not support MMX intrinsics.

When compiling your in-line assembly with MMX code, the macro package is not required.

Example A-1 shows an example of how to use in-line assembly with MMX instructions in your C source file.

Example A-1 Writing MMX Code Using In-line Assembly

```
main()
{
    _asm{
        pxor    mm0,mm0
        mov     eax,0x2345
        movd    mm0,ea
        emms
    }
}
```

Setting Options for the MSVC++ Compiler, Version 4.1

To enable the MSVC++ Compiler to detect the MMX instructions, you need to specify the following option at the compiler command line: `/G5M`

To set the `/G5M` option from the IDE, do the following:

1. From the Build menu, under the Setting option, select the C/C++ tab.
2. In the Setting For window, press the Target button and select one of the targets.
3. In the Project Options text box, at the end of the line add the option `/G5M`.

Register Viewing Tool B

The Register Viewing Tool (RVT) is a Windows 95 applet that allows you to view and modify the content of the MMX registers. RVT works in conjunction with the VxD emulator and uses the debugging features (INT1 & INT3) of the existing IA processor.

RVT displays the contents of the MMX registers after each single-step (INT1) or breakpoint (INT3). In processors without MMX technology, the MMX registers are emulated. On a Pentium® processor with MMX technology, the contents of the actual MMX registers are displayed.

Invoking the RVT:

You can invoke the RVT from any debugging session, after installing the VxD emulator.

If the VxD emulator is not installed, add the following line to your `system.ini` file:

```
[386Enh]
     device={path to viammxd.386}\viammxd.386.
```

To invoke the RVT:

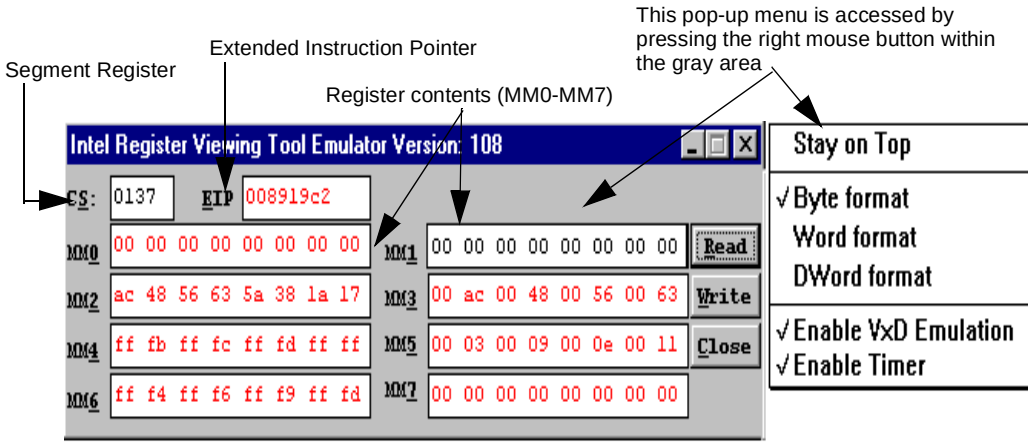
From the Windows Explorer, run `bin\rtv.exe` from the directory in which the Performance Tools are installed.

The RVT displays the contents of the MMX registers after emulation.

Register Display:

Figure B-1 shows the RVT Display window. The options and features are described below.

Figure B-1 Register Viewing Tool Display



The Register Viewing Tool displays the contents of the MMX registers in either byte, word or doubleword format.

- *MM0 - MM7*: Displays one of the following:
  - — Displays the contents of the MMX register of the debugged application. When the values of the MMX register change, the new values are displayed in red.
  - — *No Context*: Displays *No Context* if no application is debugged, or if the application is terminated.
  - — *Empty*: Displays *Empty* when the tag bits of the corresponding MMX register are empty.
- *Segment Register (CS)*: Displays the trapped segment register address.
- *Extended Instruction Pointer (EIP)*: Displays the trapped EIP. RVT looks for a change in the EIP and then updates the MMX registers. You should monitor the EIP to insure that the VxD does not trap an INT1 or INT3 that is generated by another application.

- *Read* button: Reads the contents of the MMX registers and updates the display. This is useful when the Enable Timer option is disabled.
- *Write* button: Writes the value you entered in the display into the MMX register.
- *Close* button: Closes the applet.

Pop-up Menu

The Pop-up menu is accessed by pressing the right mouse button within the gray area (see Figure B-1). This menu has the following options:

- *Stay on Top*: Keeps the RVT window on top of all other windows.
- *Byte Format*: Displays the data in the MMX registers as bytes. In this format, each MMX register contains eight bytes.

  Word Format: Displays the data in the MMX registers as 16 bits. In this format, each MMX register contains four words.

  DWord Format: Displays the data in the MMX registers as 32 bits. In this format, each MMX register contains two doublewords.
- *Enable VxD Emulation*: Toggles the VxD Emulation, available only on processors without MMX technology. When enabled, the VxD traps and emulates the MMX instruction. When disabled, no emulation is performed.
- *Enable Timer*: Toggles the use of the timer. When enabled, the MMX registers are updated automatically every second. When disabled, use the Read button to update the display.