# **NXVM - by Xu Ha**

*Some contributions/sections here are from Xu, some are from Mike K. These can be separately identified later, if needed.*

## NXVM Background / Objective:

I started building the IBM PC/AT emulator (with 8086) since early 2012, and now it supports MS-DOS 6.22.

It's more likely a skeleton of a PC: only essential devices are implemented (dma, pic, pit, fdc, fdd, hdc, hdd, and a quick & dirty bios).

Recently I spent most of my time to upgrade the 8086 cpu emulator to a 80386 cpu emulator. My goal is to run Windows 3.2 on NXVM.

NXVM can be compiled under linux or windows (visual studio 2010).

I believe the best way to understand a 16-bit emulator is to read the NXVM codes.

## NXVM Overview, Sources, Compilation (rev: nxvm\_0\_3\_010d):

Source Files:

* main.c - entry of the whole program
* console/console.ch - the console to configure/run the NXVM.
* vmachine/\* - contains all the virtual devices I've built, including the cpu.
* vmachine/system/\* - contains the platform-related codes (display & keyboard for win32 console, win32 application and linux console).
* vmachine/debug/\* - contains a 16-bit debugger and its 16-bit assembler and disassembler.
* vmachine/bios/\* - contains BIOS POST routines and INT services. The INT services are written in strings by #define:
  + When the machine starts, my internal assembler translates these strings into machine code and load them into the virtual memory.
  + The INT vector is loaded according to the assembling results.
  + In qddisk, qdcga and qdkeyb, the INT services are not written in string by #define, but by regular C statements. Calls to these INT services are redirected to the 'QDX imm8 (opcode: f1 ib') instruction (see vcpuins.c and qdbios.c).
* makefile - to build NXVM under linux with ncurses library and pthread library.
* hello.com - print 'hello' under ms-dos; test file.
* reset.com - reset the NXVM via special instruction QDX 0xFE (opcode: f1 fe)
* stop.com - stop the NXVM via special instruction QDX 0xFF (opcode: f1 ff)

Compile:

* Under Linux:
  + You need to have gcc, ncursrs and pthread libraries. Then simply make.
* Under Windows:
  + If a build is done at a lower version of MSVC++, say, 2008, and then all the files are duplicated for build runs with a higher version, say, 2010, ensure that all the files in the “Release” and “Debug” temp-folders (assuming these are the correct names) are deleted. Otherwise, the higher level of the compiler may try to process these internal files, and fail.
  + If using MSVC++2008:
    1. Copy vmachine/stdint\_vc6.h to vmachine/stdint.h
    2. Edit stdint.h, and, if the following two entries are not included with the other typedefs, then include them:
       - typedef signed \_\_int64 int64\_t;
       - typedef unsigned \_\_int64 uint64\_t;
  + Introduce all the source code files into MS Visual Studio 2008/2010/2012, except for:
    1. vmachine/stdint\_vc6.h
    2. vmachine/system/linux.c/h
  + Open a Project using existing files, and set it to build the project from all the contents of the NXVM root folder.
  + Omit stdint\_vc6.h, linux.c, linux.h from the project defn.
  + Select “Release” build: {Project Properties}, {Linker}, {System}, {SubSystem}, set to “Console”.
  + Select “Debug” build: {Project Properties}, {Linker}, {System}, {SubSystem}, set to “Console”.
  + Should the C++ “Runtime Library” option be set to /MT or /MD (for the Release build), and /MTd or/MDd (for the Debug build)?
  + Should the Linker have the /DEBUG option set for the Release build? (presumably, it should be set for the Debug build?).
  + To compile, set the Release/Debug option as needed, and Compile.
  + Many “warnings” are issued, but these have not been investigated.
  + As trivial tests, I’ve executed the Release and Debug NXVM.EXEs from 2008, 2010, and 2012, and all run. “Help” and “Exit” work in each. No other testing done yet.

Testing / Experimenting:

1. Start NXVM, and enter 'HELP' for the commands.
2. Commands not listed in Help:
   * mode - change console mode to window mode (select one as the display)
   * start - starts NXVM
   * reset - restarts NXVM
   * stop - stops NXVM
   * debug - starts the debugger. It looks similar to MS-DOS debugger.

For example, if you'd like to run MS-DOS under Win32 console:

1. Prepare a 1.44MB floppy disk image with MS-DOS 6.22 (\*.img).
2. Rename it to "fd.img".
3. Or you can load it like this: "device fdd insert fd.img".
4. Then type "start". The NXVM runs.
5. You can press "F9" to stop it.

Supplied a disk image with MS-DOS 6.22. Put it at the same directory with the NXVM binary.

## To Do in NXVM:

Sorted by priority:

1. Upgrade 8086 to 80386 (32-bit protected mode, paging mechanism, multitasking ...)
2. Implement the vga (the current version is quick & dirty, no graphic modes)
3. Implement the keyboard controller (the current version is quick & dirty)
4. Implement the hard disk (current version is quick & dirty)

Quick & dirty means the devices are only implemented as BIOS INT services, but IN/OUT instructions cannot communicate with them.

## Help Needed in NXVM:

Sorted by priority:

1. There are some bugs in the Bochs msvs project files, and not solved yet. I want to compile Bochs on Windows and compare its results with my cpu emulator. Download: <http://bochs.sourceforge.net/cgi-bin/topper.pl?name=See+All+Releases&url=http://sourceforge.net/project/showfiles.phpqmrkgroup_ideq12580> and then access: bochs/2.6.2/bochs-2.6.2-msvc-src.zip.  
     
   *Reply from Mike: the only compile problems I encountered were caused by some files (vapic.cc and sha1.cc) missing in the “Solution Explorer” of MSVC++. I put a public message about this on the BOCHS forums.*
2. It'll be very appreciated if you could build it on MSVS-2008 as well.  
     
   *Reply from Mike: If you mean “Compile BOCHS”, see compile results below. If you mean “Compile NXVM”, see compile notes above.*
3. Any document/manual about the standard keyboard controller for IBM PC?  
     
   *Reply from Mike: See next section below. Also, I have various other BIOS/HW books, manuals, and source-code.*
4. Any document/manual about the VGA standard and adapter info (registers, commands, modes..)?  
     
   *Reply from Mike: See below...*

IBM PC/XT BIOS/MS-DOS References:

* Many copies of BIOS Source-code available.
* IBM PCjr: <http://www.retroarchive.org/dos/docs/ibmpcjrtechref.pdf>
* IBM PC Ops Guide: <http://www.retroarchive.org/dos/docs/ibm_pc_op_guide.pdf>
* IBM PC Tech Ref:
  + <http://www.retroarchive.org/dos/docs/ibm_techref_v202_1.pdf>
  + <http://www.retroarchive.org/dos/docs/ibm_techref_v202_2.pdf>
  + <http://www.retroarchive.org/dos/docs/ibm_techref_v202_3.pdf>
* IBM PC-XT tech Ref: <http://www.retroarchive.org/dos/docs/ibm5160techref.pdf>
* PDF Collection (including the above): <http://www.retroarchive.org/dos/docs/>
* Programmer's Technical Reference for MSDOS (and Windows API) and the IBM PC:
  + <http://www.o3one.org/> vast no. of peripherals, chips, etc
  + <http://www.o3one.org/hwdocs/bios_doc/dosref22.html>

## Testing NXVM?:

How to debug NXVM with Bochs:

1. In each cpu\_loop (bochs/cpu/cpu.cc)...
2. I copied the status of bochs cpu to the nxvm cpu (registers and flags)
3. then execute bochs cpu and nxvm cpu separately.
4. after the execution, I compared the registers in bochs cpu with the nxvm cpu.
5. By investigating the differences occurred, I debugged the nxvm cpu and fixed a few bugs.

WOW!!

## Compiling BOCHS:

Options:

* The compilers: MSVC 2008, 2010, 2012.
* The sources: BOCHS 2.6.2 Stable, and 2.6.2 SVN (latest snapshot).
* Two builds of each were attempted: “Release” and “Debug”.

|  |  |  |  |
| --- | --- | --- | --- |
| ***Compiler*** | ***BOCHS*** | ***Release?*** | ***Debug?*** |
| 2008 | Stable | Compiled OK | Compiled OK |
|  | SVN | Compiled OK | Compiled OK |
| 2010 | Stable | Compiled OK | Compiled OK |
|  | SVN | Compiled OK | Compiled OK |
| 2012 | Stable | 2 Errors (see below)  Then Compiled OK | Compiled OK |
|  | SVN | 2 Errors (see below)  Then Compiled OK | Compiled OK |

Notes:

1. In all cases, the “???.SLN” (Solution) file was opened.
2. Many Compile/Link “Warnings” appeared - not investigated.
3. The resultant EXEs have not been “tested”!
4. Sometimes, with the 2010 and 2012 compilers, a Compile/Link run gave errors indicating that some files were “in use” and could not be used. Just rerunning (F7) eliminated these errors!!
5. The Errors:
   1. error CVT1100: duplicate resource. type:MANIFEST, name:1, language:0x0409 D:\DosBox\Compiles\2012\BOCHS-Stable\vs2008\CVTRES
   2. error LNK1123: failure during conversion to COFF: file invalid or corrupt D:\DosBox\Compiles\2012\BOCHS-Stable\vs2008\LINK
   3. Solution:
      1. Edit win32res.rc file (in the root folder).
      2. Locate the comment line about the “manifest”, and the subsequent line that references the manifest file. Eg:
         * // Manifest for both 32-bit and 64-bit Windows
         * 1 24 build\win32\bochs.manifest
      3. Change the latter to a comment (also), and re-compile:
         * // Manifest for both 32-bit and 64-bit Windows
         * // 1 24 build\win32\bochs.manifest // MJK

## NTVDM64?:

The NTVDM64 is just an idea at this moment - There should be some way to integrate the 8086 emulator with Windows, but it's not sure. All I have now is the source code of NXVM.

## “Bug” in Dosbox:

Just for your own information:

1. We spent a long time configuring Dosbox, and trying many of the options, and of the various builds/patches/etc.
2. It worked very well for our app (an MS-DOS database app, with some internal DPMI drivers), and was very fast.
3. BUT...
4. Randomly, maybe after many hours, or even days, it crashed.
5. It always crashed on the same IRET (I think the error-code was something like "Unhandled Exception 06"), and the error message suggested that the return CS:IP was unreachable (or invalid).
6. The CS:IP on the stack looked just fine, and the very same IRET and CS:IP values had already been executed hundreds of times in the app.
7. We spent weeks playing with the Dosbox debugger, trying to identify what was wrong, and, so far, have not succeeded. It seems like a Dosbox bug, perhaps the RAM pages allocated to our app are invalid...
8. We got GIGABytes of debug logs, and had to write a large "log-analyser" utility to parse the log-files, and show what functions were calling what functions, changed registers, stack usage, etc, etc...
9. While trying to analyse the debug-logs, we discovered 4/5/6 bugs in the Dosbox debugger code itself, which I have documented, and will happily share with you if you want them - they're in a "Google-Doc". We did not make actual changes to Dosbox, because we're not sufficiently familiar with the codebase in there, standards, conventions, etc, and we also don't know which build(s) to update.
10. And, the result is that, so far, we've not solved the IRET issue!!

About your problem: Exception 6 refers to the undefined opcode exception (#UD). Seems like your program runs into a wrong memory location where the opcodes are not executable. Since it happens randomly and you caught it at IRET, it might be caused by some hardware interrupts in DosBox.

Seems to be a bug in DosBox. Please try Bochs first :-)

If possible, please also try it on QEMU, VMware, VirtualBox and Virtual PC.

## DosBox / BOCHS:

If you want to run a large complex dos app, DosBox may not be the best choice. Have you tried Bochs? This should be the best open source x86 emulator now.

The lines of codes in Bochs and DosBox are too many.

We did look at Bochs about 1 year ago, but I think the performance of Dosbox was supposed to be much better, so we then tried Dosbox. I don't recall if any of the other chaps on the project tried Bochs, but I certainly did not. (There may have been some other reasons also for choosing Dosbox then, or not choosing Bochs, but I don't remember them now).

I might give BOCHS a run:

* It might also show up the IRET issue - in which case it's probably not a Dosbox problem, or
* It might NOT crash, so the problem is almost definitely in Dosbox.

[Mike Kennedy]

[End]