Documentation

8086tiny is a tiny, free, open source, portable Intel PC emulator/VM, powerful enough to run DOS, Windows 3.0, Excel, MS Flight Simulator, AutoCAD, Lotus 1-2-3, and similar applications. 8086tiny emulates a "late 80's era" PC XT-type machine with the following features:

- Intel 8086/186 CPU
- 1MB RAM
- 3.5" floppy disk controller (1.44MB/720KB)
- Fixed disk controller (supports a single hard drive up to 528MB)
- CGA/Hercules graphics card with 720x348 2-color and 320x200 4-color graphics (64KB video RAM), and CGA 80 x 25 16-color text mode support
- Accurate programmable interrupt timer (PIT)
- Keyboard controller with 83-key XT-style keyboard
- Real-time clock
- PC speaker

The emulator uses the SDL graphics library for portability, and compiles under a range of platforms (Windows, Mac OS X, Linux, Android, iOS, Raspberry Pi).

While 8086tiny as supplied implements only the 8086 instruction set, it can be extended to more complex, modern instruction sets with relative ease.

Build Instructions

The 8086tiny distribution includes a Makefile that will compile unchanged under most UNIX platforms. The 8086tiny source also compiles unchanged under Microsoft Visual Studio C/C++.

- Running `make` compiles the full 8086tiny distribution, which includes audio and CGA/Hercules graphics support via SDL.
- To compile for slower platforms (e.g. Raspberry Pi), build with `make 8086tiny_slowcpu` to increase the graphics emulation frame rate.
- If your platform lacks SDL and/or you do not need support for graphics-mode applications, you can compile without graphics and audio support by running `make no_graphics` to produce a smaller binary.

Usage

```
8086tiny bios-image-file floppy-image-file [@][harddisk-image-file]
```

If `harddisk-image-file` is prefixed with `@` then 8086tiny will boot from the hard disk image. Otherwise, 8086tiny will boot from the floppy disk image.

Under UNIXes, **the keyboard must be set to raw mode using `stty` for the emulator to run**. The distribution includes a script called `runme` which sets the keyboard mode appropriately and runs the emulator with floppy and/or hard disk images as appropriate:

```
#!/bin/sh
clear
stty cbreak raw -echo min 0
if [ -f hd.img ]
then
  ./8086tiny bios fd.img hd.img
else
  ./8086tiny bios fd.img
fi
stty cooked echo
```

Building a Hard Disk Image

To create a hard disk image for use with the emulator, start by generating a flat file called, for example, `hd.img` of the required size (under 528MB), filled with zero bytes, made using`mkfile` or a similar tool.

Preparing the hard disk image for use with the emulator under DOS is done just like a real PC:

- Boot the emulator, and use `FDISK` to partition the hard disk. When it's done FDISK will reboot the emulator.
- Use `FORMAT C:` (or `FORMAT C: /S` to create a bootable disk) to format the disk image, and you are done.

The resulting disk image is in the right format to be mounted on a real Windows PC using e.g. `OSFMount`, on a Mac using `hdiutil`, or on Linux using `mount`, providing an easy way to copy files and programs to and from the disk image. Or, you can install programs from within the emulator itself using regular floppy disk images (see "Floppy Drive Emulation" below).

Keyboard Support

The emulator simulates an XT-style keyboard controlled by an Intel 8042 chip on I/O port 0x60, generating interrupt 9 on each keypress. Because a real 8042 returns scan codes rather than the ASCII characters, for portability, the emulator BIOS does the reverse of a real PC BIOS and converts ASCII characters to scancodes, simulating press/release of the modifier keys (e.g. shift) as necessary to work like a "real" keyboard. The OS (DOS/Windows) then converts them back to ASCII characters and normally this process works seamlessly.

The scan code table in the BIOS maps your local keyboard layout onto scan codes matching a US-layout QWERTY keyboard. If you are using an international keyboard layout everything will work fine with no changes, provided "United States 83-key XT keyboard" or similar is selected if your OS (e.g. Windows 3.0) gives the option.

For console (text) applications, there are special key sequences to get Alt+*xxx*, F*xx* and some Ctrl+*xxx* keys, since these are not returned directly by the standard C I/O functions:

- To send an Alt+*xxx* key combination, press Ctrl+A then the key, so for example to type Alt+F, press Ctrl+A then F.
- To send an F*xx* key, press Ctrl+F then a number key. For example, to get the F4 key, press Ctrl+F then 4. To get F10, press Ctrl+F then 0.
- To send Ctrl+A, press Ctrl+F then Ctrl+A. To send Ctrl+F, press Ctrl+F then Ctrl+F.
- To send a Page Down key, press Ctrl+F then O. To send a Page Up key, press Ctrl+F then Q.

For graphics (SDL) applications, all keys will work as per a "real" PC without needing the special sequences above.

The keyboard is polled every `KEYBOARD_TIMER_UPDATE_DELAY` instructions. Decreasing this value will increase keyboard responsiveness, at the expense of emulated CPU speed, and vice versa. The default value of 20000 should be suitable for most platforms.

Floppy Drive Emulation

Emulates a 3.5" high-density floppy drive. Can read, write and format 1.44MB disks (18 sectors per track, 2 heads) and 720KB disks (9 sectors per track, 2 heads).

If you want to install your own software from a set of multiple floppy images (downloaded from e.g. Vetusware), the easiest way to "change disks" is to copy each disk image in turn over the floppy image file you specified on the command line, from a terminal other than the one running 8086tiny. Don't forget to put your original boot disk back at the end!

Hard Drive Emulation

Supports up to 1023 cylinders, 63 sectors per track, 63 heads for disks up to 528MB.

Disk image format used is a subset of the standard "raw" format used by most disk image mount tools. In general, disk images prepared by the emulator will work with disk image tools and other emulators, but not the other way around.

The emulator uses an overly simplistic algorithm to derive a simulated cylinder/sector/head geometry from the disk image file's size. This algorithm often results in not all the space in the image file being available for disk partitions. For example, creating a 40,000,000 byte image file results in DOS FDISK seeing only 31.9MB as the volume size.

8086tiny will boot from a hard disk image if the HD image filename is prefixed with `@` on the command line. For example: `./8086tiny bios fd.img @hd.img`

Text Mode Support

The emulator supports text output via the standard BIOS interrupt 0x10 interface, and also direct video memory access (one page, 4KB video RAM at segment B800) in 80 x 25 CGA 16-color text mode.

BIOS text output calls are converted to simple writes to `stdout`. Direct video memory accesses for the 80 x 25 CGA color text mode are converted to ANSI terminal escape sequences. If you are using a terminal which does not support ANSI (e.g. you are compiling the emulator with MS VC++ and running in a Windows console window) then PC applications that directly write to video memory in text mode may be unusable. Please make sure your terminal window is at least 80 x 25 characters in size.

Video memory in text mode is rendered to the terminal every `8 * KEYBOARD_TIMER_UPDATE_DELAY` instructions. Decreasing this value will increase the text update rate, at the expense of emulated CPU speed, and vice versa. The default value of 20000 should be suitable for most platforms.

The regular PC character code page (437) includes various extended ASCII characters for things like line drawing. You might want to set the font in your terminal program to something that includes these (for

example, on Mac OS X there is a suitable freeware font called Perfect DOS VGA 437). Otherwise, extended characters may render incorrectly (for example as question mark symbols).

Occasionally a DOS application on exit will leave the video hardware in an odd state which confuses the emulator, resulting in subsequent text output being invisible. If this happens, just use the DOS `CLS` command to clear the screen and all will be well again.

Graphics Mode Support

Hercules 720x348 monochrome graphics mode and CGA 320x200 4-color graphics mode are emulated using SDL. Most CGA/Hercules features are supported via the normal I/O interface on ports 0x3D*x* and 0x3B*x* including video memory bank switching (segments B000/B800), which some games use for double-buffered graphics. Resolution reprogramming via the CRTC register is supported by 8086tiny 1.03 and later, as required by, for example, the ETEN Chinese System (which uses 640 x 408). The CGA 640x200 2-color mode is currently not supported.

When an application enters graphics mode, the emulator will open an SDL window (which will be closed when the application goes back to text mode). On UNIXes, SDL will automatically output graphics via X11 if the DISPLAY environment variable is set up.

The graphics display is updated every `GRAPHICS_UPDATE_DELAY` instructions. Decreasing this value will increase the graphics update rate, at the expense of emulated CPU speed, and vice versa. By default, `GRAPHICS_UPDATE_DELAY` is set to 360000 instructions, which gives good performance for faster platforms. On slower platforms like Raspberry Pi, a smaller value is suitable: building with `make 8086tiny_slowcpu` reduces`GRAPHICS_UPDATE_DELAY` to 50000 instructions.

Some applications (e.g. AutoCAD) support a PC configuration with a CGA card and a Hercules card, for simultaneous text and graphics output on different displays. The emulator simulates this configuration, too, using separate windows for the (terminal) text and (SDL) graphics displays.

If your application only requires text mode, you can compile 8086tiny without SDL by defining `NO_GRAPHICS`.

Real-Time Clock Support

Reading the RTC (both time and date) is emulated via the standard BIOS clock interface, pulling the time/date from the host computer. Setting the time or date is not supported.

Hardware Timer Support

Programmable interrupt timer channels 0 and 2 are emulated through the usual I/O port 0x40-0x43 interface. Only mode 3 (square wave generator) is currently supported, but this is what most applications use. Software that uses timers to control execution speed such as games should run at an accurate pace.

PC Speaker Support

The PC speaker is emulated through the usual port 0x61 interface. The only PC speaker mode supported is via PIT channel 2, so you will hear most music but not non-musical sound effects.

BIOS

Like a real PC, the emulator needs a BIOS to implement boot functionality and the standard interrupt interfaces. The 8086tiny BIOS was written from scratch using documentation in the public domain. It is around 6KB in size and assembles using NASM. Full source code and a pre-built binary are provided.

The BIOS binary comprises a code section and a data section. The code section implements the standard interrupt interfaces for video, disk, timer, clock and so on, much as a "real" PC BIOS does, and also a small timer-controlled video driver to convert video memory formatting into ANSI escape sequences when the emulator is in text mode. The data section includes typical BIOS structures like a scan code table and the BIOS data area, but also a number of look-up tables to assist the emulator with instruction decoding. Full detail is provided in the "CPU, Memory and Register Emulation" section below.

Memory Map and Register Emulation

The emulator simulates a hardware configuration with A20 address line wraparound disabled, making just over 1MB of RAM available to applications.

Memory map is largely as per a real PC, with interrupt vector table at 0:0, BIOS data area including keyboard buffer at `40:0`, CGA text video memory at `B800:0`, Hercules/CGA graphics memory at `B000/B800:0`, and BIOS at `F000:0100`. Unlike a real PC, in the emulator the CPU registers are memory-mapped (at `F000:0`), which enables considerable optimisation of the emulator's instruction execution unit by permitting the unification of memory and register operations, while remaining invisible to the running software.

CPU, Memory and Register Emulation

CPU supports the full 8086 instruction set (plus some 80186 extensions), including undocumented instructions (e.g. `SALC`) and flag behaviors (e.g. `MUL/DIV`), opcode bugs which some applications rely on (e.g. `PUSH SP`), and the trap flag for debugger support.

The focus of 8086tiny is on minimizing code size without comproming emulation accuracy. Due to the complexities of the highly irregular Intel x86 instruction format, instruction decoding is assisted by a number of lookup tables which form part of the BIOS binary. For example, there are many different ways to encode a `MOV` instruction, depending on the types of the source and destination operands (immediate, register, or memory). There are sometimes even multiple ways to encode the same instruction (e.g. `MOV AX, [1234]` usually encodes as `A1 34 12` but can also encode as `8B 06 34 12`). To avoid having to implement similar functionality in the emulator multiple times for each instruction or encoding variant, look-up tables are used to map each instruction to an internal function and subfunction number.

As an example, we illustrate how the emulator executes the instruction `ADD AX, BX`, which encodes as hex `01 D8`.

- The emulator begins by retrieving index `01` hex (the first byte of the instruction) from `TABLE_XLAT_OPCODE` and `TABLE_XLAT_SUBFUNCTION`, giving a *translated opcode ID* of decimal 9 (which corresponds to the Intel instruction template *`arithmetic_function`* `reg, r/m`) and a *subfunction ID* of 0 (which corresponds to the `ADD` function), respectively.
- The `OPCODE` chain in the source uses the *translated opcode ID* and *subfunction ID* to determine the operation to execute, in this case calling the `OP(+=)` macro followed by `set_CF()` to set the carry flag in accordance with the result of the addition.
- Next, instruction length is computed. Because Intel x86 instructions are of arbitrary length (and, sometimes, multiple encodings of the same instruction can have different lengths), tables are used to determine the instruction length to move IP to the next instruction. The opcode index `01` hex is used as an index into `TABLE_BASE_INST_SIZE`, `TABLE_I_MOD_SIZE`, and `TABLE_I_W_SIZE` and these numbers are added to compute the total instruction length.

- Finally, flags are set. The opcode index `01` hex is then used as an index into`TABLE_STD_FLAGS` to give a bitmask of 3, which is `FLAGS_UPDATE_SZP | FLAGS_UPDATE_AO_ARITH`.
    - `FLAGS_UPDATE_SZP` (1) signifies that this instruction sets the sign, zero and parity flags according to the operation's result in the standard way. Sign and zero flags are set directly from the result, and the parity flag is set by looking up the result in `TABLE_PARITY_FLAG`.
    - `FLAGS_UPDATE_AO_ARITH` (2) signifies that this instruction sets the auxiliary and overflow flags as standard for arithmetic operations.
    - If `FLAGS_UPDATE_OC_LOGIC` (4) were set in the bitmask (it is not here), the overflow and carry flags would be set to 0, as standard for logic operations.

The CPU also implements some "special" two-byte opcodes to help the emulator talk with the outside world. These are:

- `0F 00` (`PUTCHAR_AL`) - output character in register `AL` to terminal
- `0F 01` (`GET_RTC`) - write real-time clock data (as returned by `localtime`) to memory location `ES:BX`
- `0F 02` (`READ_DISK`) - read `AX` bytes from disk at offset `512*(16*SI+BP)` into memory location ES:BX. Disk is specified in `DL` (0 = hard disk, 1 = floppy disk)
- `0F 03` (`WRITE_DISK`) - write `AX` bytes at memory location `ES:BX` to disk at offset`512*(16*SI+BP)`. Disk is specified in `DL` as per `0F 02`

Emulator exit is triggered by a `JMP 0:0` instruction, to allow the user to easily quit the emulator without shutting down the terminal.

Extension of the instruction set supported by 8086tiny can be implemented by appropriate modification to the tables described above in the BIOS source, and addition of a corresponding new `OPCODE` block in the C source.

Supported Application Software

The emulator will run practically any software a real PC (of the spec listed at the top of this page) can.

The author has successfully tested a range of software on the emulator.

- OSes/GUIs
    - MS-DOS 6.22
    - FreeDOS 0.82pl3
    - Linux/ELKS 0.1.4
    - Windows 3.0
    - DESQview 2.8
    - ETEN Chinese System
- Professional software
    - Lotus 1-2-3 R2.4
    - AsEasyAs 5.7
    - Excel 2.1 for Windows
    - AutoCAD 2.5
    - WordStar 4
- Programming languages
    - QBASIC
    - GWBASIC
    - Turbo C++

- Games
  - Carrier Command
  - Police Quest
  - SimCity
  - Alley Cat
  - MS Flight Simulator 4
  - Lots of freeware Windows games
- Diagnostic/benchmark software
  - Manifest
  - Microsoft MSD
  - InfoSpot
  - CheckIt

Download 8086tiny Release 1.25

The 8086tiny distribution includes:

- C source and Makefile
- BIOS source (builds with NASM) and ready-made BIOS binary
- FreeDOS 1.44MB floppy boot disk image from Joris_VR
- Documentation and license
- Alley Cat PC game

Comprehensive documentation is provided which should answer most questions you may have. Further questions are welcome in the forum.

Build Instructions

To build 8086tiny, you will need a C compiler of your choice (such as gcc or Microsoft Visual Studio 2013) and, for graphics and audio, SDL 1.2.

- On most platforms, just run `make` to build, and then `./runme` to launch
- On slower platforms (e.g. Raspberry Pi), build with `make 8086tiny_slowcpu` to increase the graphics emulation frame rate
- If your platform lacks SDL, build with `make no_graphics` to compile without graphics/audio support

Download Links

- ZIP format, for Windows: 8086tiny_125.zip
- BZIP2 format, for UNIX (Mac OS X/Linux/Android): 8086tiny_125.tar.bz2

A highly condensed version of 8086tiny (just 4043 bytes of C source) won the 2013International Obfucated C Code Contest. The distribution and documentation are available courtesy of the IOCCC website.

GitHub Repository

8086tiny is now on GitHub.

8086tiny is more than an emulator, it's a community. If you want to extend 8086tiny to do great things, forking the repository is highly encouraged. If you want details of your fork to be published here, get in touch.

If you find a bug in 8086tiny, consider submitting a pull request.

External Resources and Disk Images

- NASM 2.11: Freeware x86 assembler for rebuilding the BIOS binary.
- OSFMount: Windows tool for mounting emulator disk images on a real PC.
- FreeDOS: Free MS-DOS-compatible OS. As an alternative to the supplied FreeDOS boot disk, the Rugxulo 1.44MB boot disk image also works well with the emulator.
- ELKS: A Linux distribution for 8086 that runs on 8086tiny. Try this ELKS 0.1.5 floppy disk boot image.
- Vetusware: Legacy DOS/Windows software source. Sample 40MB hard disk imagecontaining a range of software can be found here.

Release History

- 1.25 (March 19th, 2014): Major CPU, graphics, text and audio performance improvements. Support for DOS Plus and CPM-86. BIOS bug fixes.
- 1.20 (February 20th, 2014): 2X faster CPU, 4X faster graphics. PC speaker, CGA graphics and programmable interrupt timer support. Revamped SDL keyboard support. BIOS keyboard and video bug fixes. Improved support for slower platforms. Source code clean-up.
- 1.15 (February 9th, 2014): Full keyboard support for graphics applications - type directly into the SDL window, rather than the console. Unrolling of 8086 repeat operations and optimization of instruction decoding to improve performance. BIOS disk and video bug fixes.
- 1.11 (February 5th, 2014): Compiled binary is now 20% smaller and 10% faster. BIOS binary is 30% smaller. Fixed bug where arrow characters were not displayed on some systems. Fixed bug causing dropped keystrokes on slower systems (e.g. Raspberry Pi). With thanks to the Raspberry Pi Foundation for the donation of a Raspberry Pi Model B.
- 1.10 (February 3rd, 2014): Support for running ELKS/Linux. Hard disk boot support. Fixed disk and video bugs in BIOS. CRT hardware cursor support added. Direct video memory access speed improvements. Improved Makefile for compiling without SDL.
- 1.03 (January 26th, 2014): Hercules CRTC resolution reprogramming now supported, needed by some applications e.g. ETEN Chinese System.
- 1.02 (January 24th, 2014): Source code tidy-up to remove compiler warnings.
- 1.01 (January 23rd, 2014): License changed to MIT License. Minor source revisions.
- 1.00 (January 21st, 2014): Initial release.