WIKIPEDIA

Protected mode

In computing, **protected mode**, also called **protected virtual address mode**,[1] is an operational mode of x86-compatible central processing units (CPUs). It allows system software to use features such as virtual memory, paging and safe multi-tasking designed to increase an operating system's control over application software.[2][3]

When a processor that supports x86 protected mode is powered on, it begins executing instructions in real mode, in order to maintain backward compatibility with earlier x86 processors.[4] Protected mode may only be entered after the system software sets up one descriptor table and enables the Protection Enable (PE) bit in the control register 0 (CR0).[5]

Protected mode was first added to the x86 architecture in 1982,[6] with the release of Intel's 80286 (286) processor, and later extended with the release of the 80386 (386) in 1985.[7] Due to the enhancements added by protected mode, it has become widely adopted and has become the foundation for all subsequent enhancements to the x86 architecture,[8] although many of those enhancements, such as added instructions and new registers, also brought benefits to the real mode.

Contents

History

The Intel 8086, the predecessor to the 286, was originally designed with a 20-bit address bus for its memory.[9] This allowed the processor to access 2^{20} bytes of memory, equivalent to 1 megabyte.[9] At the time, 1 megabyte was considered a relatively large amount of memory,[10] so the designers of the IBM Personal Computer reserved the first 640 kilobytes for use by applications and the operating system and the remaining 384 kilobytes for the BIOS (Basic Input/Output System) and memory for add-on devices.[11]

As the cost of memory decreased and memory use increased, the 1 MB limitation became a significant problem. Intel intended to solve this limitation along with others with the release of the 286.[11]

The 286

The initial protected mode, released with the 286, was not widely used;[11] for example, it was used by Microsoft Xenix (around 1984),[12] Coherent[13] and Minix.[14] Several shortcomings such as the inability to access the BIOS or DOS calls due to inability to switch back to real mode without resetting the processor prevented widespread usage.[15] Acceptance was additionally hampered by the fact that the 286 only allowed memory access in 16 bit segments via each of four segment registers, meaning only $4*2^{16}$ bytes, equivalent to 256 kilobytes, could be accessed at a time.[11] Because changing a segment register in protected mode caused a 6-byte segment descriptor to be loaded into the CPU from memory, the segment register load instruction took many tens of processor cycles, making it much slower than on the 8086; therefore, the strategy of computing segment addresses on-the-fly in order to access data structures larger than 128 kilobytes (the combined size of the two data segments) became impractical, even for those few programmers who had mastered it on the 8086/8088.

The 286 maintained backwards compatibility with its precursor the 8086 by initially entering real mode on power up.[4] Real mode functioned virtually identically to the 8086, allowing the vast majority of existing 8086 software to run unmodified on the newer 286. Real mode also served as a more basic mode in which protected mode could be set up, solving a sort of chicken-and-egg problem. To access the extended functionality of the 286, the operating system would set up some tables in memory that controlled memory access in protected mode, set the addresses of those tables into some special registers of the processor, and then set the processor into protected mode. This enabled 24 bit addressing which allowed the processor to access 2^{24} bytes of memory, equivalent to 16 megabytes.[9]

The 386

With the release of the 386 in 1985,[7] many of the issues preventing widespread adoption of the previous protected mode were addressed.[11] The 386 was released with an address bus size of 32 bits, which allows for 2^{32} bytes of memory accessing, equivalent to 4 gigabytes.[16] The segment sizes were also increased to 32 bits, meaning that the full address space of 4 gigabytes could be accessed without the need to switch between multiple segments.[16] In addition to the increased size of the address bus and segment registers, many other new features were added with the intention of increasing operational security and stability.[17] Protected mode is now used in virtually all modern operating systems which run on the x86 architecture, such as Microsoft Windows, Linux, and many others.[18]

An Intel 80386 microprocessor

Furthermore, learning from the failures of the 286 protected mode to satisfy the needs for multiuser DOS, Intel added a separate virtual 8086 mode,[19] which allowed multiple virtualized 8086 processors to be emulated on the 386. Hardware support required for virtualizing

the protected mode itself, however, had to wait for another 20 years.[20]

386 additions to protected mode

With the release of the 386, the following additional features were added to protected mode:[2]

- Paging
- 32-bit physical and virtual address space (The 32-bit physical address space is not present on the 80386SX, and other 386 processor variants which use the older 286 bus.[21])
- 32-bit segment offsets
- Ability to switch back to real mode without resetting
- Virtual 8086 mode

Entering and exiting protected mode

Until the release of the 386, protected mode did not offer a direct method to switch back into real mode once protected mode was entered. IBM devised a workaround (implemented in the IBM AT) which involved resetting the CPU via the keyboard controller and saving the system registers, stack pointer and often the interrupt mask in the real-time clock chip's RAM. This allowed the BIOS to restore the CPU to a similar state and begin executing code before the reset. Later, a triple fault was used to reset the 286 CPU, which was a lot faster and cleaner than the keyboard controller method (and does not depend on IBM AT-compatible hardware, but will work on any 80286 CPU in any system).

To enter protected mode, the Global Descriptor Table (GDT) must first be created with a minimum of three entries: a null descriptor, a code segment descriptor and data segment descriptor. In an IBM-compatible machine, the A20 line (21st address line) also must be enabled to allow the use of all the address lines so that the CPU can access beyond 1 megabyte of memory (Only the first 20 are allowed to be used after power-up, to guarantee compatibility with older software written for the Intel 8088-based IBM PC and PC/XT models). After performing those two steps, the PE bit must be set in the CR0 register and a far jump must be made to clear the prefetch input queue.

```
; set PE bit
mov eax, cr0
or eax, 1
mov cr0, eax

; far jump (cs = selector of code segment)
jmp cs:@pm

@pm:
; Now we are in PM.
```

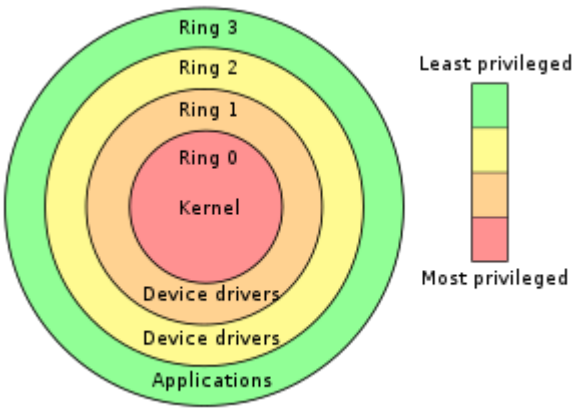
With the release of the 386, protected mode could be exited by loading the segment registers with real mode values, disabling the A20 line and clearing the PE bit in the CR0 register, without the need to perform the initial setup steps required with the 286.

Features

Protected mode has a number of features designed to enhance an operating system's control over application software, in order to increase security and system stability.[3] These additions allow the operating system to function in a way that would be significantly more difficult or even impossible without proper hardware support.[22]

Privilege levels

In protected mode, there are four privilege levels or rings, numbered from 0 to 3, with ring 0 being the most privileged and 3 being the least. The use of rings allows for system software to restrict tasks from accessing data, call gates or executing privileged instructions.[23] In most environments, the operating system and some device drivers run in ring 0 and applications run in ring 3.[23]



Example of privilege ring usage in an operating system using all rings

Real mode application compatibility

According to the *Intel 80286 Programmer's Reference Manual*,[24]

> ...the 80286 remains upwardly compatible with most 8086 and 80186 application programs. Most 8086 application programs can be re-compiled or re-assembled and executed on the 80286 in Protected Mode.

For the most part, the binary compatibility with real-mode code, the ability to access up to 16 MB of physical memory, and 1 GB of virtual memory, were the most apparent changes to application programmers.[25] This was not without its limitations. If an application utilized or relied on any of the techniques below, it wouldn't run:[26]

- Segment arithmetic
- Privileged instructions
- Direct hardware access
- Writing to a code segment
- Executing data
- Overlapping segments
- Use of BIOS functions, due to the BIOS interrupts being reserved by Intel[27]

In reality, almost all DOS application programs violated these rules.[28] Due to these limitations, virtual 8086 mode was introduced with the 386. Despite such potential setbacks, Windows 3.0 and its successors can take advantage of the binary compatibility with real mode to run many Windows 2.x (Windows 2.0 and Windows 2.1x) applications, which run in real mode in Windows 2.x, in protected mode.[29]

Virtual 8086 mode

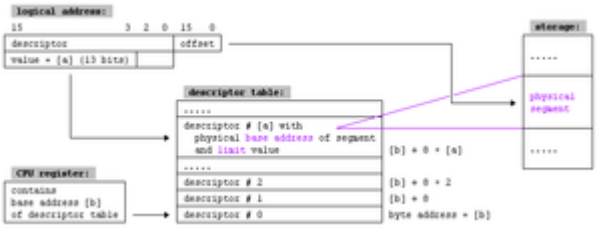
With the release of the 386, protected mode offers what the Intel manuals call *virtual 8086 mode*. Virtual 8086 mode is designed to allow code previously written for the 8086 to run unmodified and concurrently with other tasks, without compromising security or system stability.[30]

Virtual 8086 mode, however, is not completely backwards compatible with all programs. Programs that require segment manipulation, privileged instructions, direct hardware access, or use self-modifying code will generate an exception that must be served by the operating system.[31] In addition, applications running in virtual 8086 mode generate a trap with the use of instructions that involve input/output (I/O), which can negatively impact performance.[32]

Due to these limitations, some programs originally designed to run on the 8086 cannot be run in virtual 8086 mode. As a result, system software is forced to either compromise system security or backwards compatibility when dealing with legacy software. An example of such a compromise can be seen with the release of Windows NT, which dropped backwards compatibility for "ill-behaved" DOS applications.[33]

Segment addressing

In real mode each logical address points directly into physical memory location, every logical address consists of two 16 bit parts: The segment part of the logical address contains the base address of a segment with a granularity of 16 bytes, i.e. a segment may start at physical address 0, 16, 32, ..., 2^{20}-16. The offset part of the logical address contains an offset inside the segment, i.e. the physical address can be calculated as `physical_address := segment_part × 16 + offset` (if the address line A20 is enabled), respectively (segment_part × 16 + offset) mod 2^{20} (if A20 is off) Every segment has a size of 2^{16} bytes.



Virtual segments of 80286

Protected mode

In protected mode, the `segment_part` is replaced by a 16-bit *selector*, in which the 13 upper bits (bit 3 to bit 15) contain the index of an *entry* inside a *descriptor table*. The next bit (bit 2) specifies whether the operation is used with the GDT or the LDT. The lowest two bits (bit 1 and bit 0) of the selector are combined to define the privilege of the request, where the values of 0 and 3 represent the highest and the lowest privilege, respectively. This means that the byte offset of descriptors in the descriptor table is the same as the 16-bit selector, provided the lower three bits are zeroed.

The descriptor table entry defines the real *linear* address of the segment, a limit value for the segment size, and some attribute bits (flags).

286

The segment address inside the descriptor table entry has a length of 24 bits so every byte of the physical memory can be defined as bound of the segment. The limit value inside the descriptor table entry has a length of 16 bits so segment length can be between 1 byte and 2^{16} byte. The calculated linear address equals the physical memory address.

386

The segment address inside the descriptor table entry is expanded to 32 bits so every byte of the physical memory can be defined as bound of the segment. The limit value inside the descriptor table entry is expanded to 20 bits and completed with a granularity flag (G-bit, for short):

- If G-bit is zero limit has a granularity of 1 byte, i.e. segment size may be 1, 2, ..., 2^{20} bytes.
- If G-bit is one limit has a granularity of 2^{12} bytes, i.e. segment size may be 1×2^{12}, 2×2^{12}, ..., $2^{20} \times 2^{12}$ bytes. If paging is off, the calculated linear address equals the physical memory address. If paging is on, the calculated linear address is used as input of paging.

The 386 processor also uses 32 bit values for the address offset.

For maintaining compatibility with 286 protected mode a new default flag (D-bit, for short) was added. If the D-bit of a code segment is off (0) all commands inside this segment will be interpreted as 16-bit commands by default; if it is on (1), they will be interpreted as 32-bit commands.

Structure of segment descriptor entry

80286 Segment descriptor
$3_1\ 3_0\ 2_9\ 2_8\ 2_7\ 2_6\ 2_5\ 2_4\ 2_3\ 2_2\ 2_1\ 2_0\ 1_9\ 1_8\ 1_7\ 1_6\ 1_5\ 1_4\ 1_3\ 1_2\ 1_1\ 1_0\ 0_9\ 0_8\ 0_7\ 0_6\ 0_5\ 0_4\ 0_3\ 0_2\ 0_1\ 0_0$
Base[0..15] Limit[0..15]
$6_3\ 6_2\ 6_1\ 6_0\ 5_9\ 5_8\ 5_7\ 5_6\ 5_5\ 5_4\ 5_3\ 5_2\ 5_1\ 5_0\ 4_9\ 4_8\ 4_7\ 4_6\ 4_5\ 4_4\ 4_3\ 4_2\ 4_1\ 4_0\ 3_9\ 3_8\ 3_7\ 3_6\ 3_5\ 3_4\ 3_3\ 3_2$
Unused P DPL S X C R A Base[16..23]

80386 Segment descriptor
$3_1\ 3_0\ 2_9\ 2_8\ 2_7\ 2_6\ 2_5\ 2_4\ 2_3\ 2_2\ 2_1\ 2_0\ 1_9\ 1_8\ 1_7\ 1_6\ 1_5\ 1_4\ 1_3\ 1_2\ 1_1\ 1_0\ 0_9\ 0_8\ 0_7\ 0_6\ 0_5\ 0_4\ 0_3\ 0_2\ 0_1\ 0_0$
Base[0..15] Limit[0..15]
$6_3\ 6_2\ 6_1\ 6_0\ 5_9\ 5_8\ 5_7\ 5_6\ 5_5\ 5_4\ 5_3\ 5_2\ 5_1\ 5_0\ 4_9\ 4_8\ 4_7\ 4_6\ 4_5\ 4_4\ 4_3\ 4_2\ 4_1\ 4_0\ 3_9\ 3_8\ 3_7\ 3_6\ 3_5\ 3_4\ 3_3\ 3_2$
Base[24..31] G D 0 *U* Limit[16..19] P DPL S X C R A Base[16..23]

Where:

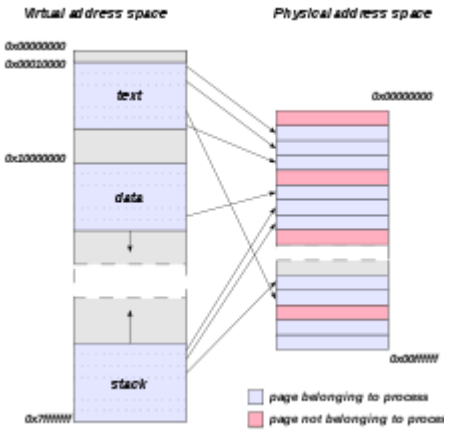
- *A* is the *Accessed* bit;
- *R* is the *Readable* bit;
- *C* (Bit 42) depends on *X*[34]:
  - if *X* = 1 then *C* is the *Conforming* bit, and determines which privilege levels can far-jump to this segment (without changing privilege level):
    - if *C* = 0 then only code with the same privilege level as *DPL* may jump here;
    - if *C* = 1 then code with the same or a lower privilege level relative to *DPL* may jump here.
  - if *X* = 0 then *C* is the *direction* bit:
    - if *C* = 0 then the segment grows *up*;
    - if *C* = 1 then the segment grows *down*.
- *X* is the *Executable* bit[34]:
  - if *X* = 1 then the segment is a code segment;
  - if *X* = 0 then the segment is a data segment.
- *S* is the *Segment type* bit, which should generally be cleared for system segments;[34]
- *DPL* is the *Descriptor Privilege Level*;
- *P* is the *Present* bit;
- *D* is the *Default operand size*;
- *G* is the *Granularity* bit;
- Bit 52 of the 80386 descriptor is not used by the hardware.

Paging

In addition to adding virtual 8086 mode, the 386 also added paging to protected mode.[35] Through paging, system software can restrict and control a task's access to pages, which are sections of memory. In many operating systems, paging is used to create an independent virtual address space for each task, preventing one task from manipulating the

memory of another. Paging also allows for pages to be moved out of primary storage and onto a slower and larger secondary storage, such as a hard disk drive.[36] This allows for more memory to be used than physically available in primary storage.[36]

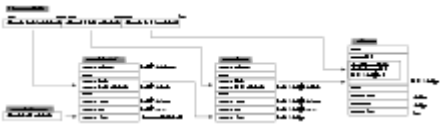
The x86 architecture allows control of pages through two arrays: page directories and page tables. Originally, a page directory was the size of one page, four kilobytes, and contained 1,024 page directory entries (PDE), although subsequent enhancements to the x86 architecture have added the ability to use larger page sizes. Each PDE contained a pointer to a page table. A page table was also originally four kilobytes in size and contained 1,024 page table entries (PTE). Each PTE contained a pointer to the actual page's physical address and are only used when the four-kilobyte pages are used. At any given time, only one page directory may be in active use.[37]



Common method of using paging to create a virtual address space

Multitasking

Through the use of the rings, privileged call gates, and the Task State Segment (TSS), introduced with the 286, preemptive multitasking was made possible on the x86 architecture. The TSS allows general-purpose registers, segment selector fields, and stacks to all be modified without affecting those of another task. The TSS also allows a task's privilege level, and I/O port permissions to be independent of another task's.


Paging (on Intel 80386) with page size of 4K

In many operating systems, the full features of the TSS are not used.[38] This is commonly due to portability concerns or due to the performance issues created with hardware task switches.[38] As a result, many operating systems use both hardware and software to create a multitasking system.[39]

Operating systems

Operating systems like OS/2 1.x try to switch the processor between protected and real modes. This is both slow and unsafe, because a real mode program can easily crash a computer. OS/2 1.x defines restrictive programming rules allowing a *Family API* or *bound* program to run in either real or protected mode. Some early Unix operating systems, OS/2 1.x, and Windows used this mode.

Windows 3.0 was able to run real mode programs in 16-bit protected mode; when switching to protected mode, it decided to preserve the single privilege level model that was used in real mode, which is why Windows applications and DLLs can hook interrupts and do direct hardware access. That lasted through the Windows 9x series. If a Windows 1.x or 2.x program is written properly and avoids segment arithmetic, it will run the same way in both real and protected modes. Windows programs generally avoid segment arithmetic because Windows implements a software virtual memory scheme, moving program code and data in memory when programs are not running, so manipulating absolute addresses is dangerous; programs should only keep handles to memory blocks when not running. Starting an old program while Windows 3.0 is running in protected mode triggers a warning dialog, suggesting to either run Windows in real mode or to obtain an updated version of the application. Updating well-behaved programs using the MARK utility with the MEMORY parameter avoids this dialog. It is not possible to have some GUI programs running in 16-bit protected mode and other GUI programs running in real mode. In Windows 3.1, real mode was no longer supported and could not be accessed.

In modern 32-bit operating systems, virtual 8086 mode is still used for running applications, e.g. DPMI compatible DOS extender programs (through virtual DOS machines) or Windows 3.x applications (through the Windows on Windows subsystem) and certain classes of device drivers (e.g. for changing the screen-resolution using BIOS functionality) in OS/2 2.0 and later, all under control of a 32-bit kernel. However, 64-bit operating systems (which run in long mode) no longer use this, since virtual 8086 mode has been removed from long mode.

See also

- Assembly language
- Intel
- Ring (computer security)
- x86 assembly language

References

1. "Memory access control method and system for realizing the same" (https://web.archive.org/web/20070926220520/http://www.patentstorm.us/patents/5483646-claims.html). *US Patent 5483646*. May 23, 1995. Archived from the original (http://www.patentstorm.us/patents/5483646-claims.html) (Patent) on September 26, 2007. Retrieved 2007-07-14. "The memory access control system according to claim 4, wherein said first address mode is a real address mode, and said second address mode is a protected virtual address mode."
2. *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture* (https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-software-developers-manual-volume-1-basic-architecture). Intel. May 2019. Section 2.1.3 The Intel 386 Processor (1985).
3. root (July 14, 2007). "Guide: What does protected mode mean?" (http://www.delorie.com/djgpp/doc/ug/basics/protected.html) (Guide). Delorie Software. Retrieved 2007-07-14. "The purpose of protected mode is not to protect your program. The purpose is to protect everyone else (including the operating system) from your program."
4. *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture* (https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-software-developers-manual-volume-1-basic-architecture). Intel. May 2019. Section 3.1 Modes of Operation.
5. Collins, Robert (2007). "Protected Mode Basics" (http://archive.wikiwix.com/cache/20110707003604/ftp://ftp.utcluj.ro/pub/users/nedevschi/PMP/protected86/collinsprot.PDF) (PDF). ftp.utcluj.ro. Archived from the original (ftp://ftp.utcluj.ro/pub/users/nedevschi/PMP/protected86/collinsprot.PDF) (PDF) on 2011-07-07. Retrieved 2009-07-31.
6. *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture* (https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-software-developers-manual-volume-1-basic-architecture). Intel. May 2019. Section 2.1.2 The Intel 286 Processor (1982).
7. "Intel Global Citizenship Report 2003" (https://web.archive.org/web/20080322075839/http://www.intel.com/intel/finance/gcr03/39-years_of_innovation.htm). Archived from the original (http://www.intel.com/intel/finance/gcr03/39-years_of_innovation.htm) (Timeline) on 2008-03-22. Retrieved 2007-07-14. "1985 Intel launches Intel386 processor"
8. *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture* (https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-software-developers-manual-volume-1-basic-architecture). Intel. May 2019. Section 2.1 Brief History of Intel 64 and IA-32 Architecture.
9. "A+ - Hardware" (http://www.brainbell.com/tutors/A+/Hardware/PC_Microprocessor_Developments_and_Features.htm) (Tutorial/Guide). *PC Microprocessor Developments and Features Tutorials*. BrainBell.com. Retrieved 2007-07-24.
10. Risley, David (March 23, 2001). "A CPU History" (https://web.archive.org/web/20080829165311/http://www.pcmech.com/show/processors/35/). PCMechanic. Archived from the original (http://www.pcmech.com/show/processors/35/) (Article) on August 29, 2008. Retrieved 2007-07-24. "What is interesting is that the designers of the time never suspected anyone would ever need more than 1 MB of RAM."
11. Kaplan, Yariv (1997). "Introduction to Protected-Mode" (https://web.archive.org/web/20070622205752/http://www.internals.com/articles/protmode/introduction.htm). Internals.com. Archived from the original (http://www.internals.com/articles/protmode/introduction.htm) (Article) on 2007-06-22. Retrieved 2007-07-24.

12. "Microsoft XENIX 286 Press Release" (http://www.tenox.net/docs/xenix/microsoft_xenix_30_286_press_release.pdf) (PDF) (Press release). Microsoft.
13. "General Information FAQ for the Coherent Operating System" (http://textfiles.com/internet/FAQ/coherent.faq). January 23, 1993.
14. "MINIX Information Sheet" (https://web.archive.org/web/20140107074722/http://minix.net/minix/minix.html). Archived from the original (http://minix.net/minix/minix.html) on January 7, 2014.
15. Mueller, Scott (March 24, 2006). "P2 (286) Second-Generation Processors" (http://www.informit.com/articles/article.aspx?p=481859&seqNum=13). *Upgrading and Repairing PCs, 17th Edition* (http://www.informit.com/store/product.aspx?isbn=0789734044) (Book) (17 ed.). Que. ISBN 0-7897-3404-4. Retrieved 2017-07-11.
16. *80386 Programmer's Reference Manual* (http://bitsavers.org/pdf/intel/80386/230985-001_80386_Programmers_Reference_Manual_1986.pdf) (PDF). Santa Clara, CA: Intel. 1986. Section 2.1 Memory Organization and Segmentation.
17. *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture* (https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-software-developers-manual-volume-1-basic-architecture). Intel. May 2019. Section 3.1 Modes of Operation.
18. Hyde, Randall (November 2004). "12.10. Protected Mode Operation and Device Drivers" (http://safari.oreilly.com/1593270038/ns1593270038-CHP-12-SECT-10). *Write Great Code*. O'Reilly. ISBN 1-59327-003-8.
19. Charles Petzold, Intel's 32-bit Wonder: The 80386 Microprocessor, ***PC Magazine***, November 25, 1986, pp. 150-152
20. Tom Yager (6 November 2004). "Sending software to do hardware's job" (http://www.infoworld.com/article/2664741/computer-hardware/sending-software-to-do-hardware-s-job.html). *InfoWorld*. Retrieved 24 November 2014.
21. Shvets, Gennadiy (June 3, 2007). "Intel 80386 processor family" (http://www.cpu-world.com/CPUs/80386/index.html) (Article). Retrieved 2007-07-24. "80386SX — low cost version of the 80386. This processor had 16 bit external data bus and 24-bit external address bus."
22. *Intel 80386 Programmer's Reference Manual 1986* (http://bitsavers.org/components/intel/80386/230985-001_80386_Programmers_Reference_Manual_1986.pdf) (PDF). Santa Clara, CA: Intel. 1986. Chapter 7, Multitasking.
23. *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture* (https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-software-developers-manual-volume-1-basic-architecture). Intel. May 2019. Section 6.3.5 Calls to Other Privilege Levels.
24. *80286 and 80287 Programmer's Reference Manual* (http://bitsavers.org/components/intel/80286/210498-005_80286_and_80287_Programmers_Reference_Manual_1987.pdf) (PDF). Santa Clara, CA: Intel. 1987. Section 1.2 Modes of Operation.
25. *80286 and 80287 Programmer's Reference Manual* (http://bitsavers.org/components/intel/80286/210498-005_80286_and_80287_Programmers_Reference_Manual_1987.pdf) (PDF). Santa Clara, CA: Intel. 1987. Section 1.3.1 Memory Management.
26. *80286 and 80287 Programmer's Reference Manual* (http://bitsavers.org/components/intel/80286/210498-005_80286_and_80287_Programmers_Reference_Manual_1987.pdf) (PDF). Santa Clara, CA: Intel. 1987. Appendix C 8086/8088 Compatibility Considerations.
27. "Memory access control method and system for realizing the same" (http://www.freepatentsonline.com/6105101.html) (Patent). *US Patent 5483646*. May 6, 1998. Retrieved 2007-07-25. "This has been impossible to-date and has forced BIOS development teams to add support into the BIOS for 32 bit function calls from 32 bit applications."
28. Robinson, Tim (August 26, 2002). "Virtual 8086 Mode" (https://web.archive.org/web/20021003235610/http://osdev.berlios.de/v86.html). berliOS. Archived from the original (http://osdev.berlios.de/v86.html) (Guide) on October 3, 2002. Retrieved 2007-07-25. "...secondly, protected mode was also incompatible with the vast amount of real-mode code around at the time."
29. Robinson, Tim (August 26, 2002). "Virtual 8086 Mode" (https://web.archive.org/web/20021003235610/http://osdev.berlios.de/v86.html). berliOS. Archived from the original (http://osdev.berlios.de/v86.html) (Guide) on October 3, 2002. Retrieved 2007-07-25.
30. *Intel 64 and IA-32 Architectures Software Developer's Manual Combined Volumes 3A, 3B, 3C, and 3D: System Programming Guide* (https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-sdm-combined-volumes-3a-3b-3c-and-3d-system-programming-guide). Intel. May 2019. Section 20.2 Virtual 8086 Mode.

31. *Intel 64 and IA-32 Architectures Software Developer's Manual Combined Volumes 3A, 3B, 3C, and 3D: System Programming Guide* (https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-sdm-combined-volumes-3a-3b-3c-and-3d-system-programming-guide). Intel. May 2019. Section 20.2.7 Sensitive Instructions.
32. Robinson, Tim (August 26, 2002). "Virtual 8086 Mode" (https://web.archive.org/web/20021003235610/http://osdev.berlios.de/v86.html). berliOS. Archived from the original (http://osdev.berlios.de/v86.html) (Guide) on October 3, 2002. Retrieved 2007-07-25. "A downside to using V86 mode is speed: every IOPL-sensitive instruction will cause the CPU to trap to kernel mode, as will I/O to ports which are masked out in the TSS."
33. Dabak, Prasad; Millind Borate (October 1999). *Undocumented Windows NT* (Book). Hungry Minds. ISBN 0-7645-4569-8.
34. "Global Descriptor table - OSDev Wiki" (https://wiki.osdev.org/Global_Descriptor_Table#Structure).
35. "ProtectedMode overview [deinmeister.de]" (http://www.deinmeister.de/x86modes.htm#c1) (Website). Retrieved 2007-07-29.
36. "What Is PAE X86?" (http://technet2.microsoft.com/windowsserver/en/library/efc41320-713f-4004-bc81-ddddfc8552651033.mspx?mfr=true) (Article). Microsoft TechNet. May 28, 2003. Retrieved 2007-07-29. "The paging process allows the operating system to overcome the real physical memory limits. However, it also has a direct impact on performance because of the time necessary to write or retrieve data from disk."
37. Gareau, Jean. "Advanced Embedded x86 Programming: Paging" (http://www.embedded.com/98/9806fe2.htm) (Guide). Embedded.com. Retrieved 2007-07-29. "Only one page directory may be active at a time, indicated by the CR3 register."
38. zwanderer (May 2, 2004). "news: Multitasking for x86 explained #1" (https://web.archive.org/web/20070212161434/http://neworder.box.sk/newsread.php?newsid=10562). *NewOrer*. NewOrder. Archived from the original (http://neworder.box.sk/newsread.php?newsid=10562) (Article) on 2007-02-12. Retrieved 2007-07-29. "The reason why software task switching is so popular is that it can be faster than hardware task switching. Intel never actually developed the hardware task switching, they implemented it, saw that it worked, and just left it there. Advances in multitasking using software have made this form of task switching faster (some say up to 3 times faster) than the hardware method. Another reason is that the Intel way of switching tasks isn't portable at all"
39. zwanderer (May 2, 2004). "news: Multitasking for x86 explained #1" (https://web.archive.org/web/20070212161434/http://neworder.box.sk/newsread.php?newsid=10562). *NewOrer*. NewOrder. Archived from the original (http://neworder.box.sk/newsread.php?newsid=10562) (Article) on 2007-02-12. Retrieved 2007-07-29. "...both rely on the Intel processors ability to switch tasks, they rely on it in different ways."

External links

- Protected Mode Basics (http://www.rcollins.org/articles/pmbasics/tspec_a1_doc.html)
- Introduction to Protected-Mode (https://web.archive.org/web/20070622205752/http://www.internals.com/articles/protmode/introduction.htm)
- Overview of the Protected Mode Operations of the Intel Architecture (http://www.intel.com/design/intarch/papers/exc_ia.htm)
- TurboIRC.COM tutorial to enter protected mode from DOS (http://www.turboirc.com/asm)
- Protected Mode Overview and Tutorial (http://viralpatel.net/taj/tutorial/protectedmode.php)
- Code Project Protected Mode Tutorial (http://www.codeproject.com/KB/system/asm.aspx)
- Akernelloader switching from real mode to protected mode (https://code.google.com/p/akernelloader/)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Protected_mode&oldid=917615863"

This page was last edited on 24 September 2019, at 17:09 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.