x86 MEMORY MANAGEMENT

x86 processors manage memory according to the basic modes of operation discussed.

Protected mode is the most robust and powerful, but it does restrict application programs from directly accessing system hardware.

In **real-address mode**, only 1 MByte of memory can be addressed, from hexadecimal 00000 to FFFFF. The processor can run only one program at a time, but it can momentarily interrupt that program to process requests **(called interrupts)** from peripherals. Application programs are permitted to access any memory location, including addresses that are linked directly to system hardware. The MS-DOS operating system runs in real-address mode, and Windows 95 and 98 can be booted into this mode.

In **protected mode**, the processor can run multiple programs at the same time. It assigns each process (running program) a total of 4 GByte of memory. Each program can be assigned its own reserved memory area, and programs are prevented from accidentally accessing each other's code and data. MS-Windows and Linux run in protected mode.

In **virtual-8086 mode**, the computer runs in protected mode and creates a virtual-8086 machine with its own 1-MByte address space that simulates an 80x86 computer running in real-address mode. Windows NT and 2000, for example, create a virtual-8086 machine when you open a Command window. You can run many such windows at the same time, and each is protected from the actions of the others. Some MS-DOS programs that make direct references to computer hardware will not run in this mode under Windows NT, 2000, and XP. Chapter 11 explains many more details of both real-address mode and protected mode.

x86-64 PROCESSOR vs x86 PROCESSORS

The x86-64 instruction set is an extension of the x86 instruction set, but with several important features, such as 64-bit long addresses that allow for a virtual address space of size 2^{64} bytes, 64-

bit general-purpose registers, and **eight additional general-purpose registers** compared to the x86.

These processors use a 48-bit physical address space, which can support up to 256 terabytes of RAM. However, they do not support 16-bit real mode or virtual-8086 mode when running in native 64-bit mode. While x86-64 is technically an instruction set, it is commonly referred to as a processor type, and for the purpose of learning assembly language, it is not necessary to consider hardware implementation differences between processors that support x86-64.

The x86-64 instruction set is a 64-bit extension of the x86 instruction set, and it is backward-compatible with the x86 instruction set. In 64-bit mode, the processor runs applications that use the 64-bit linear address space, and it enables 64-bit instruction operands. There are two modes in the Intel 64 architecture, namely compatibility mode and 64-bit mode. Compatibility mode enables existing 16-bit and 32-bit applications to run without being recompiled, while 64-bit mode is the native mode for 64-bit Microsoft Windows.

In terms of registers, 64-bit processors have sixteen **64-bit general-purpose registers**, eight **80-bit floating-point registers**, a **64-bit status flags register** named **RFLAGS (only the lower 32 bits are used)**, and a **64-bit instruction pointer named RIP.**

There are also specialized registers for **multimedia processing**, including eight **64-bit MMX registers** and **sixteen 128-bit XMM registers**. By **adding the REX prefix to each instruction**, the operands can be extended to 64 bits, and a total of 16 general-purpose registers become available. The general-purpose registers can access 8-bit, 16-bit, 32-bit, or 64-bit operands, and in 64-bit mode, the default operand size is 32 bits.

Operand Size	Available Registers
8 bits	AL, BL, CL, DL, DIL, SIL, BPL, SPL, R8L, R9L, R10L, R11L, R12L, R13L, R14L, R15L
16 bits	AX, BX, CX, DX, DI, SI, BP, SP, R8W, R9W, R10W, R11W, R12W, R13W, R14W, R15W
32 bits	EAX, EBX, ECX, EDX, EDI, ESI, EBP, ESP, R8D, R9D, R10D, R11D, R12D, R13D, R14D, R15D
64 bits	RAX, RBX, RCX, RDX, RDI, RSI, RBP, RSP, R8, R9, R10, R11, R12, R13, R14, R15

Here are a few more details to remember: In 64-bit mode, a single instruction cannot access both a high-byte register, such as AH, BH, CH, and DH, and at the same time, the low byte of one of the new byte registers (such as DIL). The 32-bit EFLAGS register is replaced by a 64-bit RFLAGS register in 64-bit mode. The two registers share the same lower 32 bits, and the upper 32 bits of RFLAGS are not used. The status flags are the same in 32-bit mode and 64-bit mode.

COMPONENTS OF x86 COMPUTER

The **motherboard** is the central component of a microcomputer, consisting of a flat circuit board that houses the CPU, supporting processors, main memory, input-output connectors, power supply connectors, and expansion slots.

These components are interconnected by a set of wires etched directly on the motherboard, known as the **bus.**

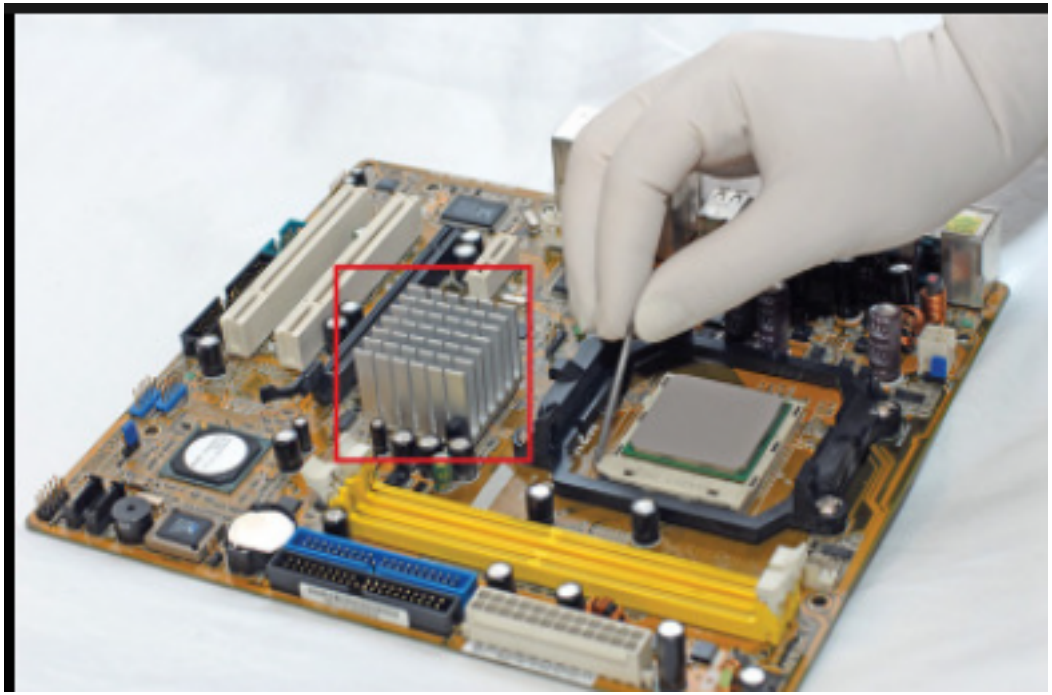
There are various types of motherboards available on the PC market, differing in terms of expansion capabilities, integrated components, and speed.

Some common components found on PC motherboards include a CPU socket (which comes in different shapes and sizes, depending on the type of processor they support), memory slots (which hold small plug-in memory boards), BIOS chips (which hold system software), CMOS RAM (which has a small circular battery to keep it powered), connectors for mass-storage devices like hard drives and CD-ROMs, USB connectors for external devices, and ports for keyboards and mice.

Assembling a computer requires careful consideration of the motherboard and the components that will be connected to it. Understanding the various components and their functions can help you make informed decisions when building or upgrading a system.

Additionally, programming at the assembly language level requires an understanding of how to interface with system hardware, firmware, and operating system functions to perform I/O operations at different levels of access.

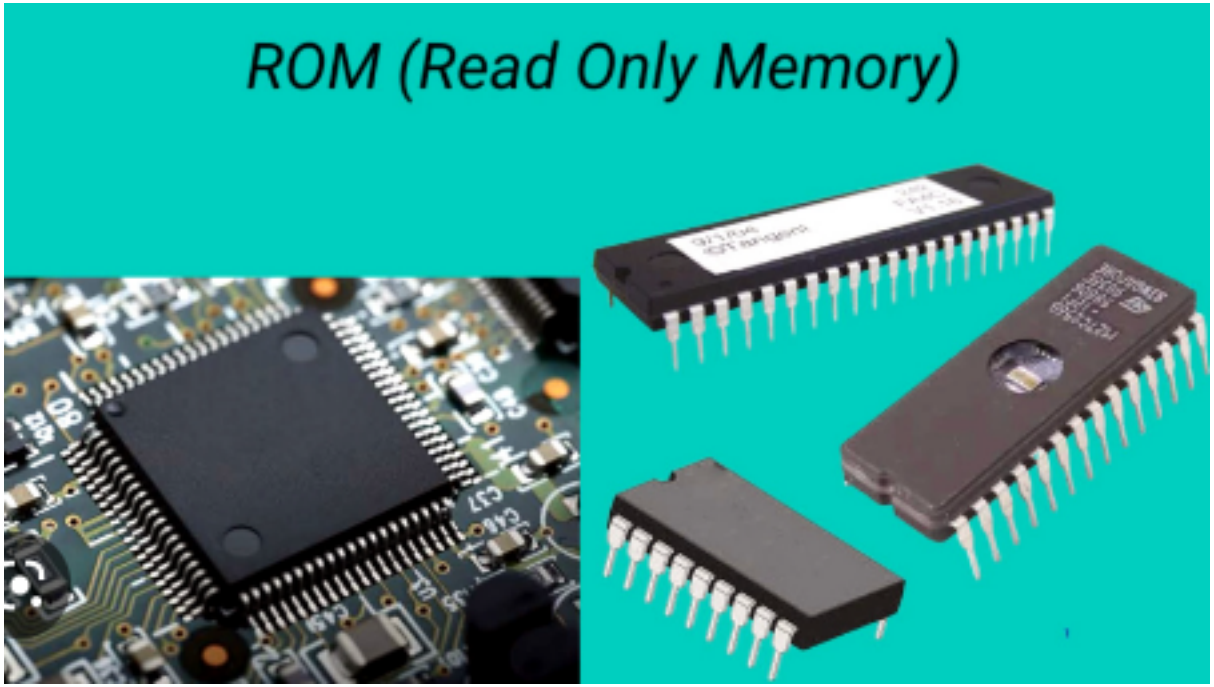
A **motherboard chipset** is a collection of processor chips designed to work together on a specific type of motherboard with various features such as increased processing power, multimedia capabilities, or reduced power consumption.



Intel Fast Memory Access uses an updated Memory Controller Hub (MCH). It can access dual-channel DDR2 memory, at an 800 MHz clock speed.

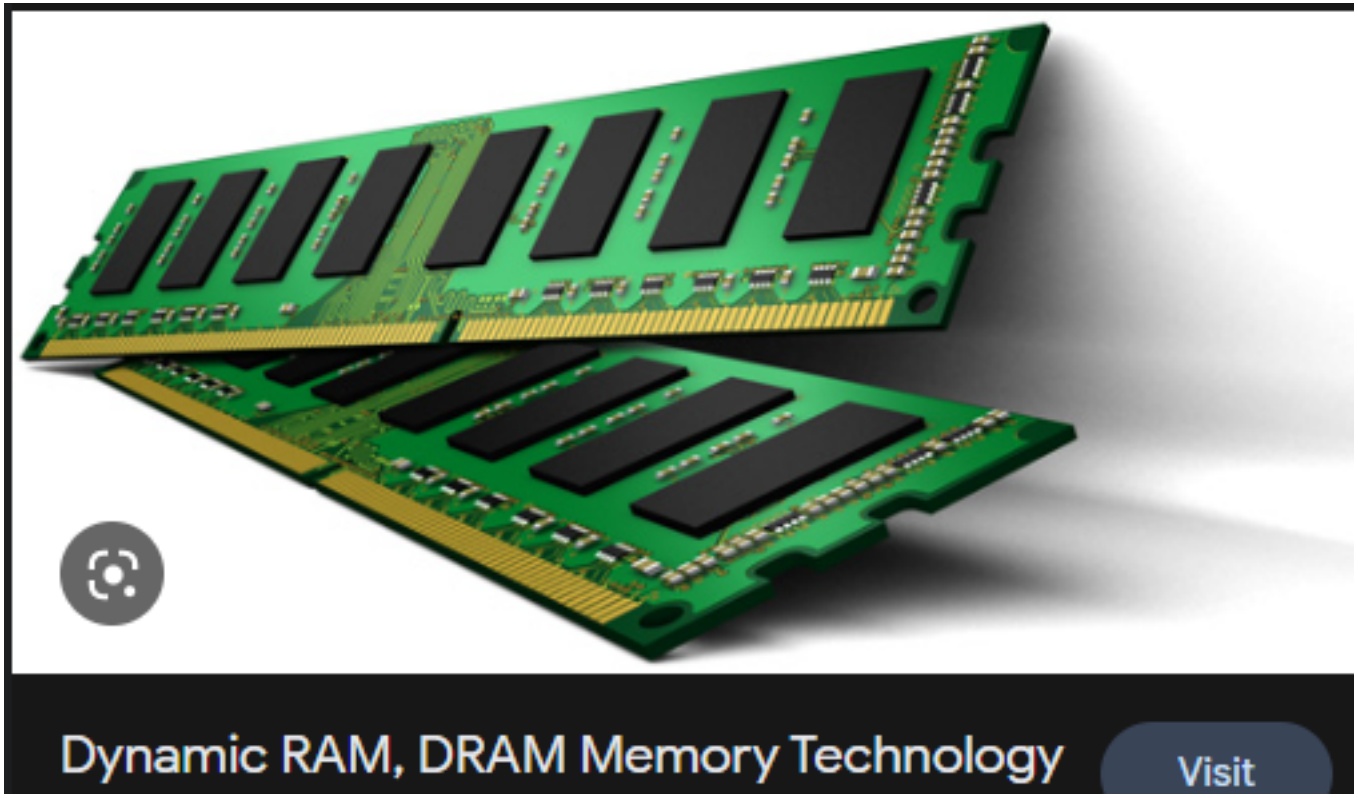
RAM AND ROM

Read-only memory (ROM) is a type of memory that is permanently burned into a chip during the manufacturing process. ROM cannot be altered, and its contents cannot be erased or rewritten. ROM is often used to store the computer's BIOS (basic input-output system), which contains the firmware code that initializes the hardware during boot-up.

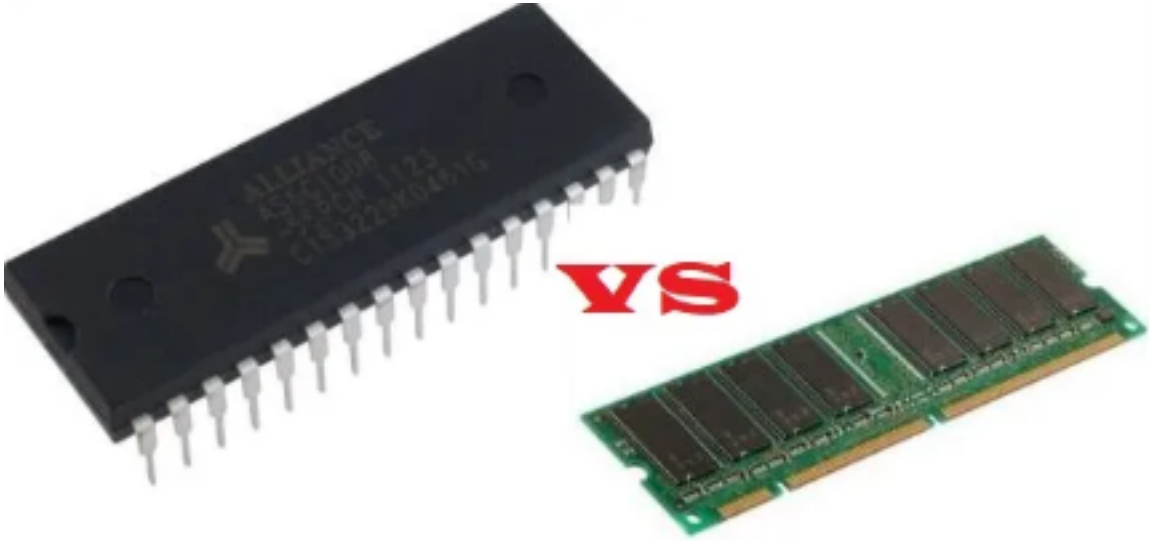


Erasable programmable read-only memory (EPROM) is a type of memory that can be erased and reprogrammed. To erase EPROM, ultraviolet light is used, which can take several minutes. EPROM is primarily used in embedded systems, where firmware needs to be updated frequently.

Dynamic random-access memory (DRAM) is the most commonly used type of memory in computers. DRAM is used as main memory where programs and data are stored when a program is running. DRAM is inexpensive, but it needs to be refreshed every millisecond to prevent data loss. Some systems use ECC (error checking and correcting) memory, which can detect and correct memory errors.

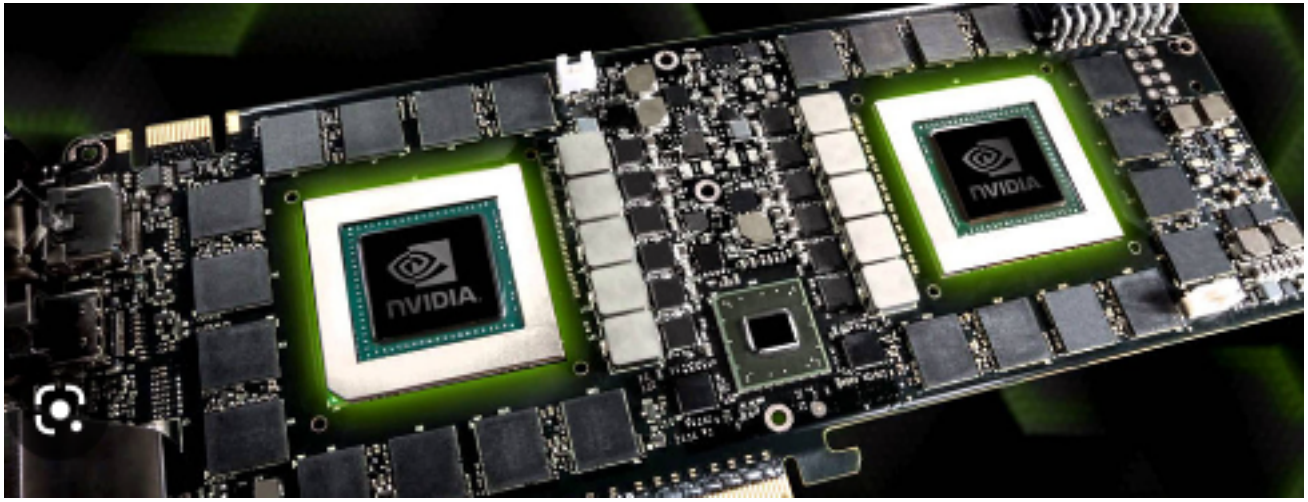


Static RAM (SRAM) is a type of memory that is used primarily for expensive, high-speed cache memory. SRAM does not need to be refreshed, and it is faster than DRAM. CPU cache memory is made up of SRAM.

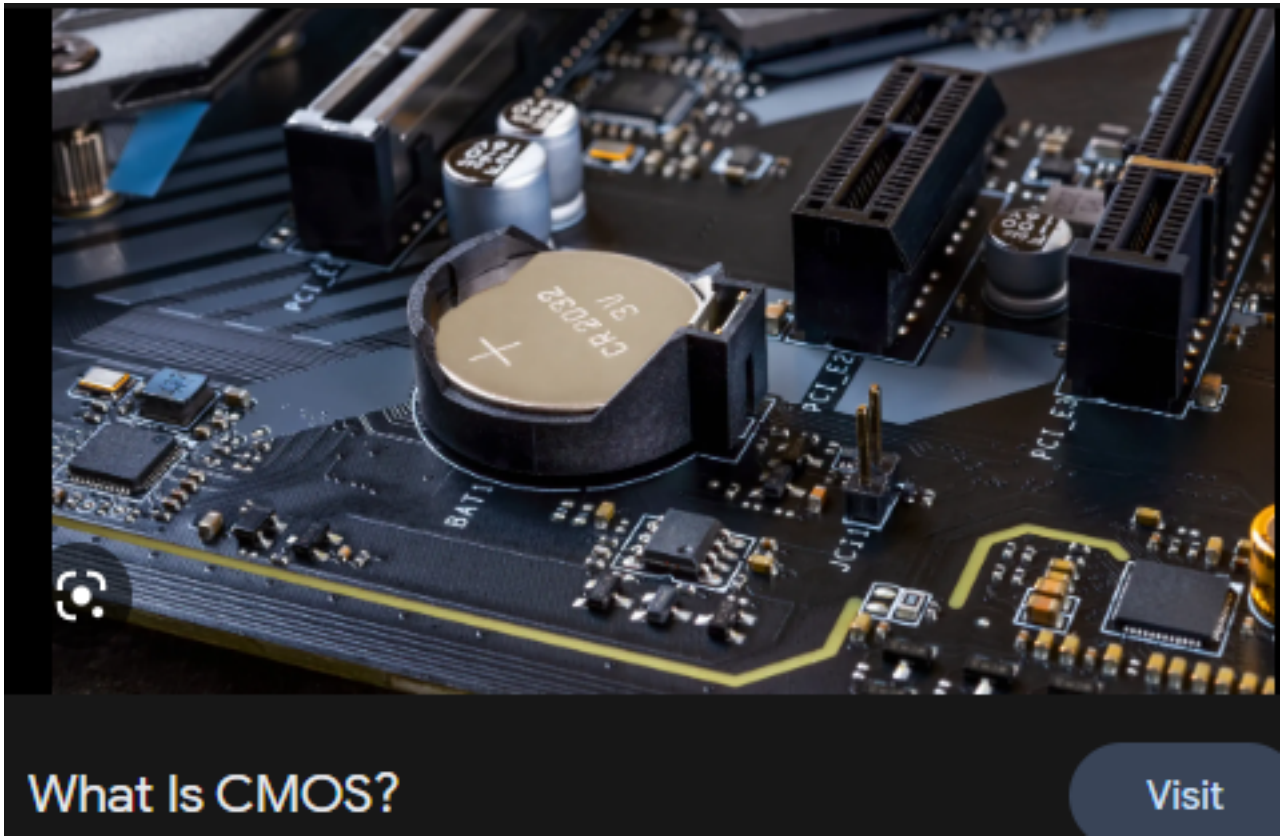


SRAM VS DRAM

Video RAM (VRAM) is a type of memory that is used to hold video data. VRAM is dual ported, which means that one port can continuously refresh the display while the other port writes data to the display. VRAM is designed to work with video controllers and graphics processing units (GPUs).

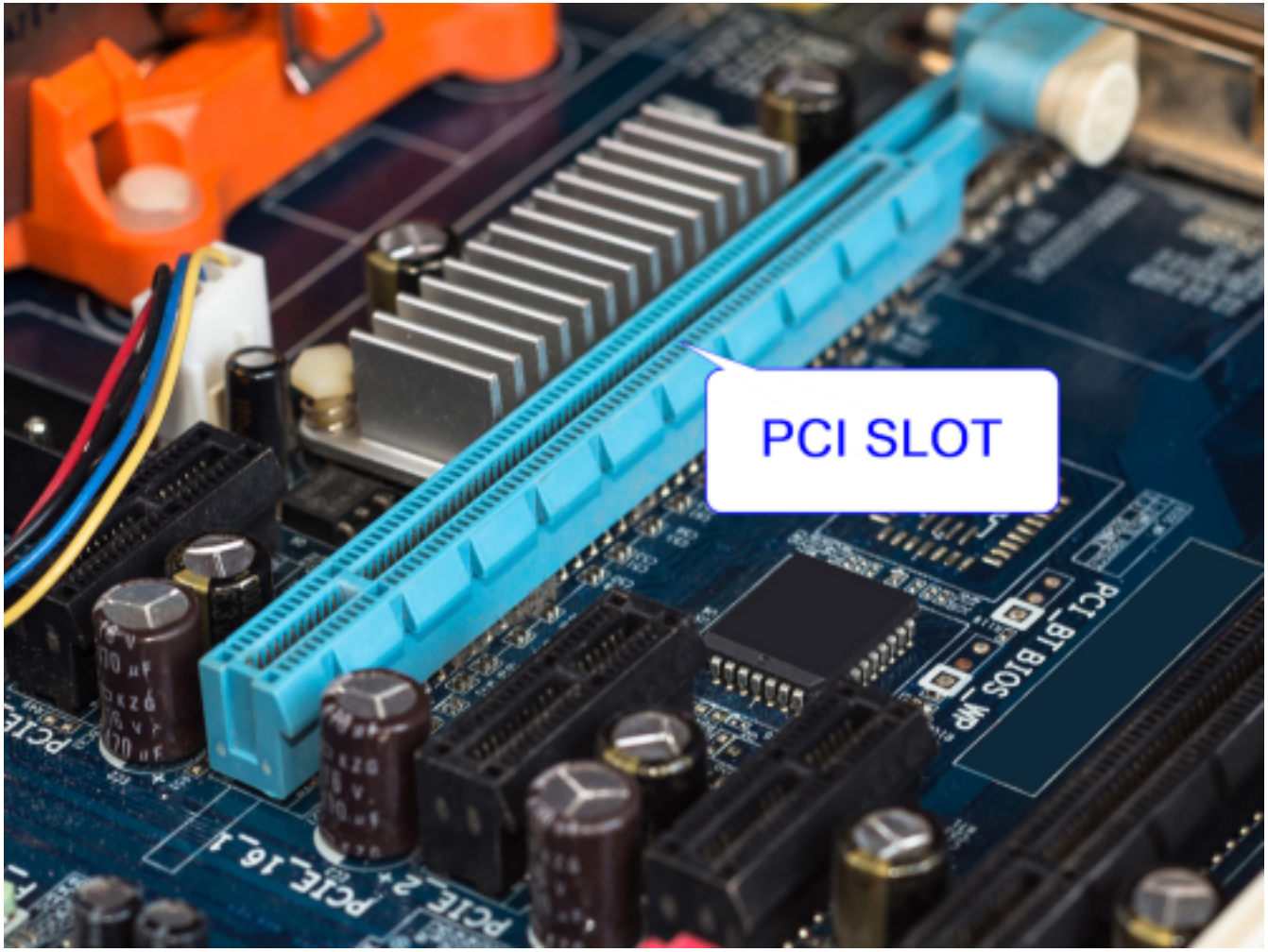


Complimentary metal oxide semiconductor (CMOS) RAM is a type of memory that is used to store system setup information on the motherboard. CMOS RAM is refreshed by a battery, which means that its contents are retained even when the computer is turned off. CMOS RAM is used to store information such as the date and time, as well as hardware configuration settings.

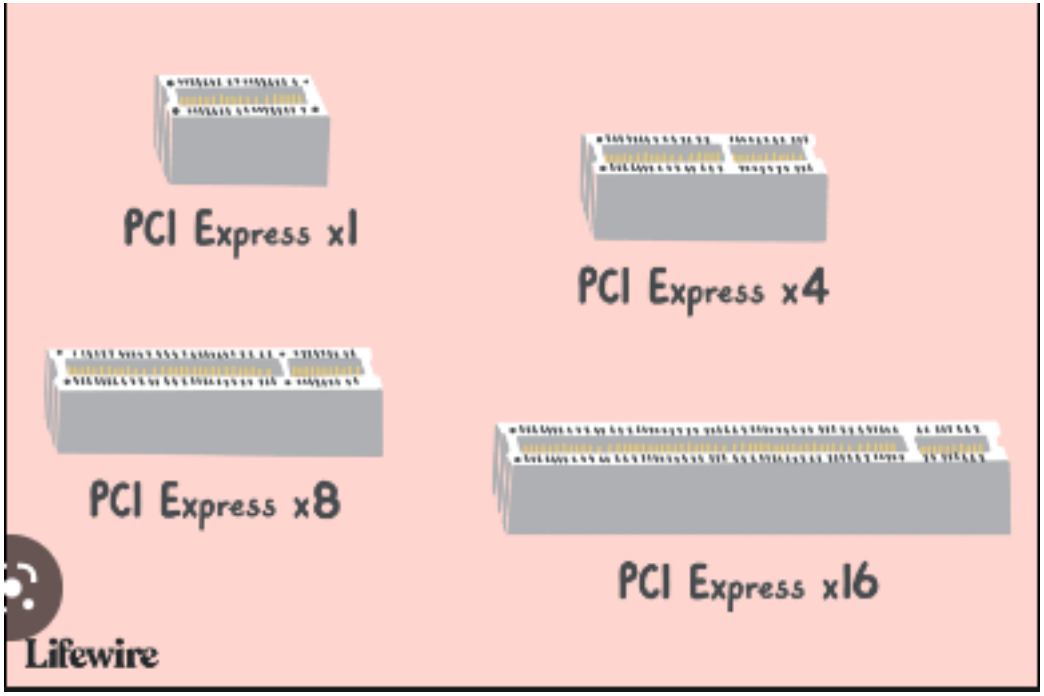


PCI AND PCI EXPRESS BUS ARCHITECTURE

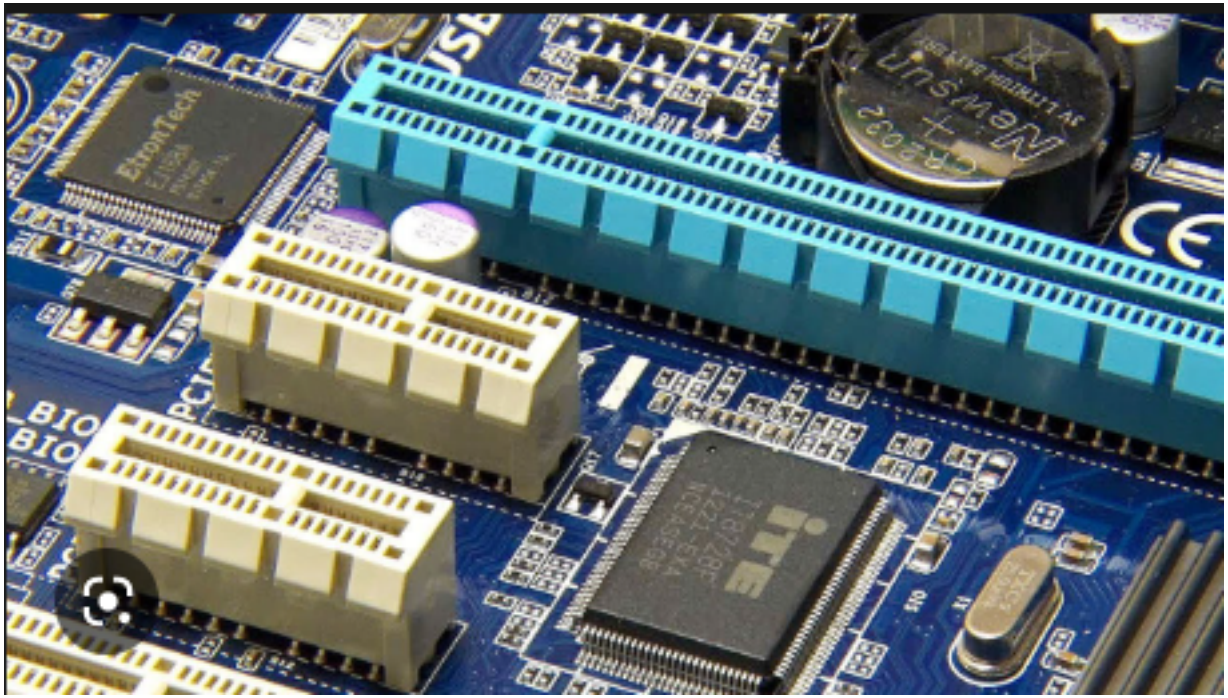
The Peripheral Component Interconnect (PCI) bus is a standard for connecting various peripherals to the CPU in a computer system. It provides a high-speed data path between the CPU and devices such as hard drives, memory, video controllers, sound cards, and network controllers. The PCI bus provides a bridge between these devices and the CPU, allowing them to communicate with each other.



PCI Express is a newer and faster version of the PCI bus architecture. It provides two-way serial connections between devices, memory, and the processor, and carries data in packets, similar to networks. It uses separate "lanes" for data transfer, which allows it to transfer data at much higher speeds than the older PCI bus architecture. PCI Express is widely supported by graphics controllers, and is commonly used for high-speed data transfer in modern computer systems.


PCI Express x1
PCI Express x4
PCI Express x8
PCI Express x16
Lifewire

EG.



LEVELS OF I/0 ACCESS

I/O can be performed by calling functions provided by the operating system, instead of directly accessing hardware.

There are three primary levels of I/O access: **high-level language functions**, **operating system functions**, and **BIOS low-level subroutines** that communicate directly with hardware devices.

High-level language functions: A high-level programming language such as C++ or Java contains functions to perform input–output. These functions are portable because they work on a variety of different computer systems and are not dependent on any one operating system.

Operating system: Programmers can call operating system functions from a library known as the API (application programming interface). The operating system provides high-level operations such as writing strings to files, reading strings from the keyboard, and allocating blocks of memory.

BIOS: The basic input–output system is a collection of low-level subroutines that communicate directly with hardware devices. The BIOS is installed by the computer's manufacturer and is tailored to fit the computer's hardware. Operating systems typically communicate with the BIOS.

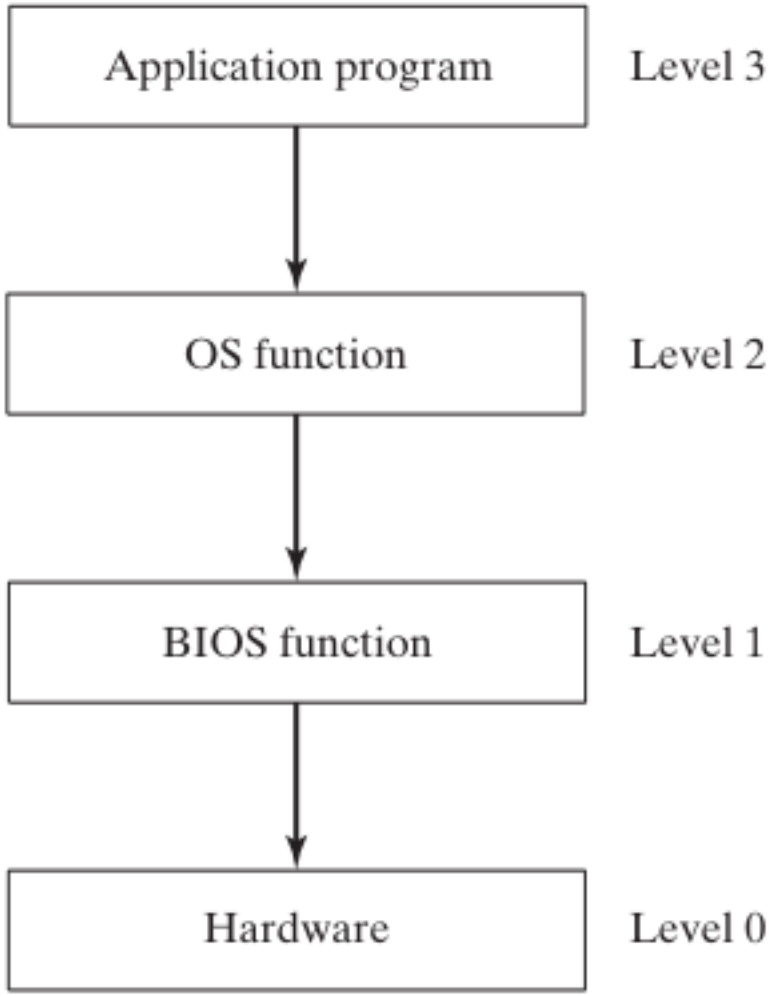
DEVICE DRIVERS

Device drivers are programs that permit the operating system to communicate directly with hardware devices and the system BIOS. For example, a device driver might receive a request from the OS to read some data; the device driver satisfies the request by executing code in the device firmware that reads data in a way that is unique to the device.

Device drivers are usually installed in one of two ways: (1) before a specific hardware device is attached to a computer, or (2) after a device has been attached and identified. In the latter case, the OS recognizes the device name and signature; it then locates and installs the device driver software onto the computer.

We can put the I/O hierarchy into perspective by showing what happens when an application program displays a string of characters on the screen.

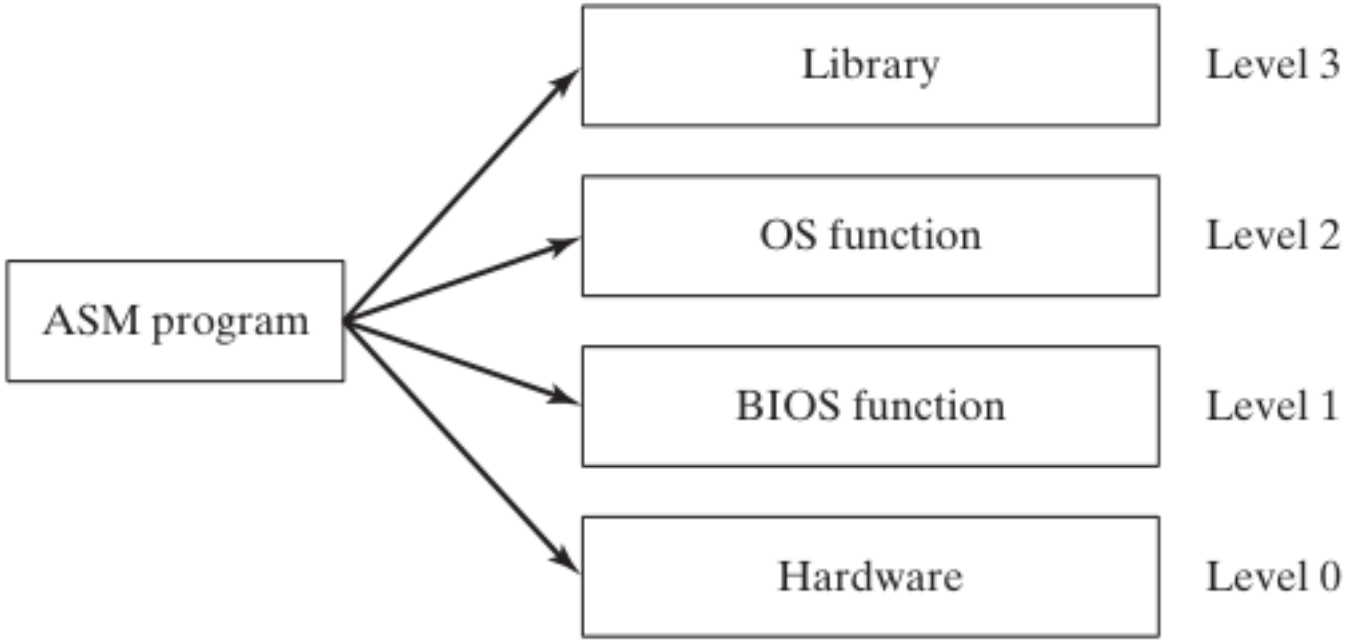
Figure 2–7 Access levels for input–output operations.



The following steps are involved: A statement in the application program calls an HLL library function that writes the string to standard output. The library function (Level 3) calls an operating system function, passing a string pointer. The operating system function (Level 2) uses a loop to call a BIOS subroutine, passing it the ASCII code and color of each character. The operating system calls another BIOS subroutine to advance the cursor to the next position on the screen. The BIOS subroutine (Level 1) receives a character, maps it to a particular system font, and sends the

character to a hardware port attached to the video controller card. The video controller card (Level 0) generates timed hardware signals to the video display that control the raster scanning and displaying of pixels.

FIGURE 2–8 Assembly language access levels.



Programming at Multiple Levels

Assembly language programs have power and flexibility in the area of input-output programming. They can choose from the following access levels (Figure 2-8).

Level 3: Call library functions to perform generic text I/O and file-based I/O. We supply such a library with this book, for instance.

Level 2: Call operating system functions to perform generic text I/O and file-based I/O. If the OS uses a graphical user interface, it has functions to display graphics in a device-independent way.

Level 1: Call BIOS functions to control device-specific features such as color, graphics, sound, keyboard input, and low-level disk I/O.

Level 0: Send and receive data from hardware ports, having absolute control over specific devices.

This approach cannot be used with a wide variety of hardware devices, so we say that it is not portable. Different devices often use different hardware ports, so the program code must be customized for each specific type of device. What are the tradeoffs? Control versus portability is the primary one.

Level 2 (OS) works on any computer running the same operating system. If an I/O device lacks certain capabilities, the OS will do its best to approximate the intended result. Level 2 is not particularly fast because each I/O call must go through several layers before it executes.

Level 1 (BIOS) works on all systems having a standard BIOS, but will not produce the same result on all systems. For example, two computers might have video displays with different resolution capabilities. A programmer at Level 1 would have to write code to detect the user's hardware setup and adjust the output format to match.

Level 1 runs faster than Level 2 because it is only one level above the hardware. Level 0 (hardware) works with generic devices such as serial ports and with specific I/O devices produced by known manufacturers. Programs using this level must extend their coding logic to handle variations in I/O devices.

Real-mode game programs are prime examples because they usually take control of the computer. Programs at this level execute as quickly as the hardware will permit.

Suppose, for example, you wanted to play a WAV file using an audio controller device. At the OS level, you would not have to know what type of device was installed, and you would not be concerned with nonstandard features the card might have.

At the BIOS level, you would query the sound card (using its installed device driver software) and find out whether it belonged to a certain class of sound cards having known features.

At the hardware level, you would fine tune the program for certain models of audio cards, taking advantage of each card's special features.

General-purpose operating systems rarely permit application programs to directly access system hardware, because to do so would make it nearly impossible for multiple programs to run simultaneously.

Instead, hardware is accessed only by device drivers, in a carefully controlled manner. On the other hand, smaller operating systems for specialized devices often do connect directly to hardware.

They do this in order to reduce the amount of memory taken up by operating system code, and they almost always run just a single program at one time.

The last Microsoft operating system to allow programs to directly access hardware was MS-DOS, and it was only able to run one program at a time.

What characteristics distinguish BIOS-level input/output?