Lecture No.4

Lecture Outlines

2.3 64-Bit x86-64 Processors
- 2.3.1 64-Bit Operation Modes
- 2.3.2 Basic 64-Bit Execution Environment

2.4 Components of a Typical x86 Computer
- 2.4.1 Motherboard
- 2.4.2 Memory

2.5 Input-Output System
- 2.5.1 Levels of I/O Access

2.3 64-Bit x86-64 Processors

In this section, we focus on the basic architectural details of all 64-bit processors that use the x86-64 instruction set. This group the Intel 64 and AMD64 processor families. The instruction set is a 64-bit extension of the x86 instruction set we've already looked at. Here are some of the essential features:

1. It is backward-compatible with the x86 instruction set.
2. Addresses are 64 bits long, allowing for a virtual address space of size 2^{64} bytes. In current chip implementations, only the lowest 48 bits are used.
3. It can use 64-bit general-purpose registers, allowing instructions to have 64-bit integer operands.
4. It uses eight more general-purpose registers than the x86.
5. It uses a 48-bit physical address space, which supports up to 256 terabytes of RAM.

On the other hand, when running in native 64-bit mode, these processors do not support 16-bit real mode or virtual-8086 mode. (There is a *legacy mode* that still supports 16-bit programming, but it is not available in 64-bit versions of Microsoft Windows.)

> ***Note:*** Although *x86-64* refers to an instruction set, we will from this point on treat it as a processor type. For the purpose of learning assembly language, it is not necessary to consider hardware implementation differences between processors that support x86-64.

The first Intel processor to use x86-64 was the Xeon, followed by a host of other processors, including Core i5 and Core i7 processors. Examples of AMD's processors that use x86-64 are Opteron and Athlon 64.

You might also have heard of another 64-bit architecture from Intel known as *IA-64*, later renamed to *Itanium*. The IA-64 instruction set is completely different from x86 and x86-64. Itanium processors are often used for high-performance database and network servers.

2.3.1 64-Bit Operation Modes

The Intel 64 architecture introduces a new mode named *IA-32e*. Technically it contains two submodes, named *compatibility mode* and *64-bit mode*. But it's easier to refer to these as modes rather than submodes, so we will do that from now on.

Compatibility Mode

When running in compatibility mode, existing 16-bit and 32-bit applications can usually run without being recompiled. However, 16-bit Windows (Win16) and DOS applications will not run in 64-bit Microsoft Windows. Unlike earlier versions of Windows, 64-bit Windows does not have a virtual DOS machine subsystem to take advantage of the processor's ability to switch into virtual-8086 mode.

64-Bit Mode

In 64-bit mode, the processor runs applications that use the 64-bit linear address space. This is the native mode for 64-bit Microsoft Windows. This mode enables 64-bit instruction operands.

2.3.2 Basic 64-Bit Execution Environment

In 64-bit mode, addresses can theoretically be as large as 64-bits, although processors currently only support 48 bits for addresses. In terms of registers, the following are the most important differences from 32-bit processors:

- Sixteen 64-bit general purpose registers (in 32-bit mode, you have only eight general-purpose registers)
- Eight 80-bit floating-point registers
- A 64-bit status flags register named RFLAGS (only the lower 32 bits are used)
- A 64-bit instruction pointer named RIP

As you may recall, the 32-bit flags and instruction pointers are named EFLAGS and EIP. In addition, there are some specialized registers for multimedia processing we mentioned when talking about the x86 processor:

- Eight 64-bit MMX registers
- Sixteen 128-bit XMM registers (in 32-bit mode, you have only 8 of these)

General-Purpose Registers

The general-purpose registers, introduced when we described 32-bit processors, are the basic operands for instructions that do arithmetic, move data, and loop through data. The general-purpose registers can access 8-bit, 16-bit, 32-bit, or 64-bit operands (with a special prefix).

In 64-bit mode, the default operand size is 32 bits and there are eight general-purpose registers. By adding the REX (register extension) prefix to each instruction, however, the operands can be 64 bits long and a total of 16 general-purpose registers become available. You have all the same registers as in 32-bit mode, plus eight numbered registers, R8 through R15. Table 2-1 shows which registers are available when the REX prefix is enabled.

Table 2-1 Operand Sizes in 64-Bit Mode When REX Is Enabled.

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.

2.4 Components of a Typical x86 Computer

Let us look at how the x86 integrates with other components by examining a typical motherboard configuration and the set of chips that surround the CPU. Then we will discuss memory, I/O ports, and common device interfaces. Finally, we will show how assembly language programs can perform I/O at different levels of access by tapping into system hardware, firmware, and by calling functions in the operating system.

2.4.1 Motherboard

The heart of a microcomputer is its *motherboard*, a flat circuit board onto which are placed the computer's CPU, supporting processors (*chipset*), main memory, input-output connectors, power supply connectors, and expansion slots. The various components are connected to each other by a *bus*, a set of wires etched directly on the motherboard. Dozens of motherboards are available on the PC market, varying in expansion capabilities, integrated components, and speed. The following components have traditionally been found on PC motherboards:

- A CPU socket. Sockets are different shapes and sizes, depending on the type of processor they support
- Memory slots (SIMM or DIMM), holding small plug-in memory boards
- BIOS (*basic input–output system*) computer chips, holding system software
- CMOS RAM, with a small circular battery to keep it powered
- Connectors for mass-storage devices such as hard drives and CD-ROMs
- USB connectors for external devices
- Keyboard and mouse ports

- PCI bus connectors for sound cards, graphics cards, data acquisition boards, and other input–output devices

The following components are optional:

- Integrated sound processor
- Parallel and serial device connectors
- Integrated network adapter
- AGP bus connector for a high-speed video card

Following are some important support processors in a typical system:

- The *Floating-Point Unit* (FPU) handles floating-point and extended integer calculations.
- The 8284/82C284 *Clock Generator*, known simply as the *clock*, oscillates at a constant speed. The clock generator synchronizes the CPU and the rest of the computer.
- The 8259A *Programmable Interrupt Controller* (PIC) handles external interrupts from hardware devices, such as the keyboard, system clock, and disk drives. These devices interrupt the CPU and make it process their requests immediately.
- The 8253 *Programmable Interval Timer/Counter* interrupts the system 18.2 times per second, updates the system date and clock, and controls the speaker. It is also responsible for constantly refreshing memory because RAM memory chips can remember their data for only a few milliseconds.
- The 8255 *Programmable Parallel Port* transfers data to and from the computer using the IEEE Parallel Port interface. This port is commonly used for printers, but it can be used with other input–output devices as well.

PCI and PCI Express Bus Architectures

The **PCI** (*Peripheral Component Interconnect*) bus provides a connecting bridge between the CPU and other system devices such as hard drives, memory, video controllers, sound cards, and network controllers. More recently, the *PCI Express* bus provides two-way serial connections between devices, memory, and the processor. It carries data in packets, similar to networks, in separate "lanes." It is widely supported by graphics controllers, and can transfer data at very high speeds.

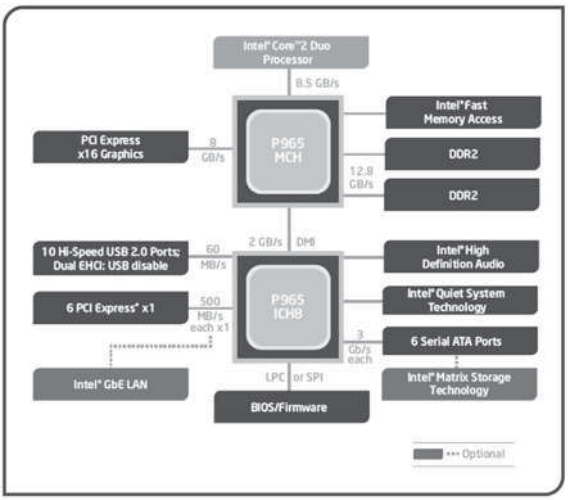
Motherboard Chipset

A *motherboard chipset* is a collection of processor chips designed to work together on a specific type of motherboard. Various chipsets have features that increase processing power, multimedia capabilities, or reduce power consumption. The *Intel P965 Express Chipset* can be used as an example. It is used in desktop PCs, with either an Intel Core 2 Duo or a Pentium D processor. Here are some of its features:

- Intel *Fast Memory Access* uses an updated Memory Controller Hub (MCH). It can access dual-channel DDR2 memory, at an 800 MHz clock speed.
- An I/O Controller Hub (Intel ICH8/R/DH) uses Intel Matrix Storage Technology (MST) to support multiple Serial ATA devices (disk drives).
- Support for multiple USB ports, multiple PCI express slots, networking, and Intel Quiet System technology.
- A high definition audio chip provides digital sound capabilities.

A diagram may be seen in Figure 2-6. Motherboard manufacturers will build products around specific chipsets. For example, the P5B-E P965 motherboard by Asus Corporation uses the P965 chipset.

Figure 2–6 Intel 965 express chipset block diagram.



Source: The Intel P965 Express Chipset (product brief), © 2006 by Intel Corporation, used by permission.

2.4.2 Memory

Several basic types of memory are used in Intel-based systems: read-only memory (ROM), erasable programmable read-only memory (EPROM), dynamic random-access memory (DRAM), static RAM (SRAM), video RAM (VRAM), and complimentary metal oxide semiconductor (CMOS) RAM:

- **ROM** is permanently burned into a chip and cannot be erased.
- **EPROM** can be erased slowly with ultraviolet light and reprogrammed.
- **DRAM**, commonly known as main memory, is where programs and data are kept when a program is running. It is inexpensive, but must be refreshed every millisecond to avoid losing its contents. Some systems use ECC (error checking and correcting) memory.
- **SRAM** is used primarily for expensive, high-speed cache memory. It does not have to be refreshed. CPU cache memory is comprised of SRAM.
- **VRAM** holds video data. It is dual ported, allowing one port to continuously refresh the display while another port writes data to the display.
- **CMOS RAM** on the system motherboard stores system setup information. It is refreshed by a battery, so its contents are retained when the computer's power is off.

2.5 Input–Output System

Tip: Because computer games are so memory and I/O intensive, they push computer performance to the max. Programmers who excel at game programming often know a lot about video and sound hardware, and optimize their code for hardware features.

2.5.1 Levels of I/O Access

Application programs routinely read input from keyboard and disk files and write output to the screen and to files. I/O need not be accomplished by directly accessing hardware—instead, you can call functions provided by the operating system. I/O is available at different access levels, similar to the virtual machine concept shown in Chapter 1. There are three primary levels:

- **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.

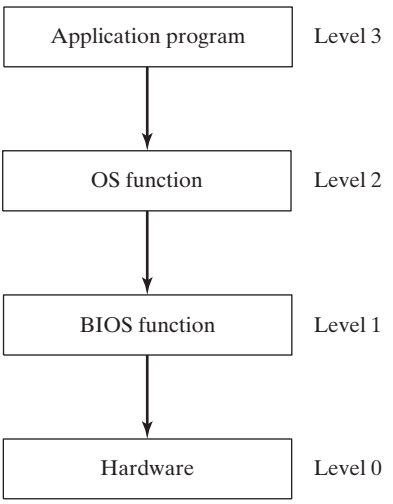
Device Drivers *Device drivers* are programs that permit the operating system to communicate directly with hardware devices and the system BIOS. For example, a device driver might receive a request from the OS to read some data; the device driver satisfies the request by executing code in the device 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 (Fig. 2-7). The following steps are involved:

1. A statement in the application program calls an HLL library function that writes the string to standard output.
2. The library function (Level 3) calls an operating system function, passing a string pointer.

3. 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.
4. 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.
5. The video controller card (Level 0) generates timed hardware signals to the video display that control the raster scanning and displaying of pixels.

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



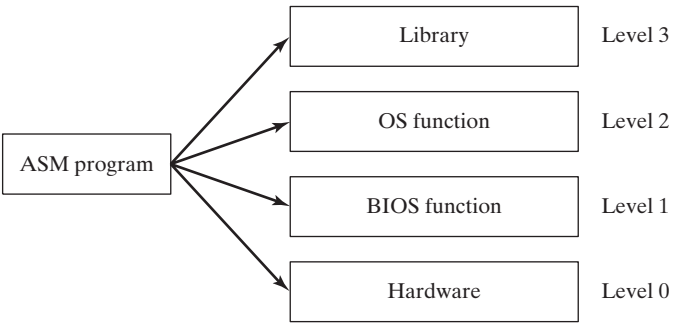
Programming at Multiple Levels Assembly language programs have power and flexibility in the area of input-output programming. They can choose from the following access levels (Figure 2-8):

- 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.

Figure 2–8 Assembly language access levels.



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.