



Computer Organization & Assembly Languages

Computer Organization (II)

IA-32 Processor Architecture

Pu-Jen Cheng



Materials

- Some materials used in this course are adapted from
 - The slides prepared by Kip Irvine for the book, *Assembly Language for Intel-Based Computers*, 5th Ed.
 - *Assembly Language & Computer Organization*, NTU
(<http://www.csie.ntu.edu.tw/~cyy/courses/assembly/05fall/news/>)
(http://www.csie.ntu.edu.tw/~acpang/course/asm_2004)



IA-32 architecture

- From 386 to the latest 32-bit processor, P4
- From programmer's point of view, IA-32 has not changed substantially except the introduction of a set of high-performance instructions



IA-32 Processor Architecture

- Modes of operation
- Basic execution environment
- Floating-point unit



Modes of Operation

- Protected mode
 - native mode (Windows, Linux)
- Real-address mode
 - native MS-DOS
- System management mode
 - power management, system security, diagnostics

- Virtual-8086 mode
 - hybrid of Protected
 - each program has its own 8086 computer



Multitasking (supported by protected mode)

- OS can run multiple programs at the same time.
- Multiple threads of execution within the same program.
- Scheduler utility assigns a given amount of CPU time to each running program.
- Rapid switching of tasks
 - gives illusion that all programs are running at once
 - the processor must support task switching.



Basic Execution Environment

- Addressable memory
- General-purpose registers
- Index and base registers
- Specialized register uses
- Status flags
- Floating-point, MMX, XMM registers



Addressable Memory

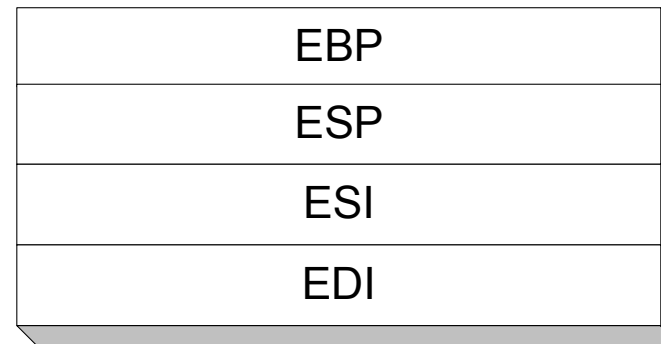
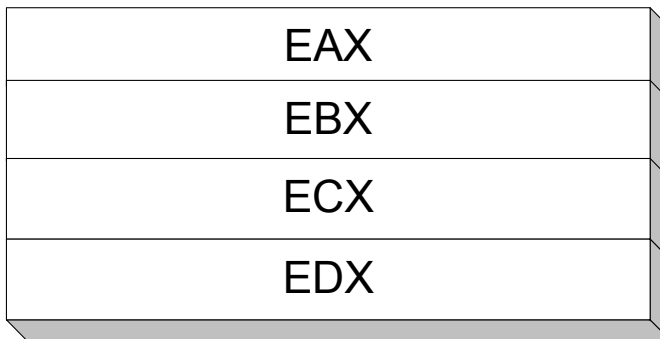
- Protected mode
 - 4 GB
 - 32-bit address
- Real-address and Virtual-8086 modes
 - 1 MB space
 - 20-bit address



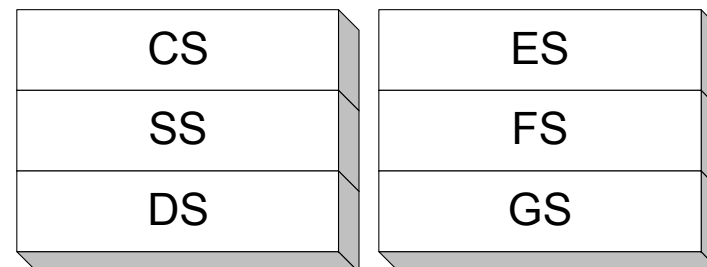
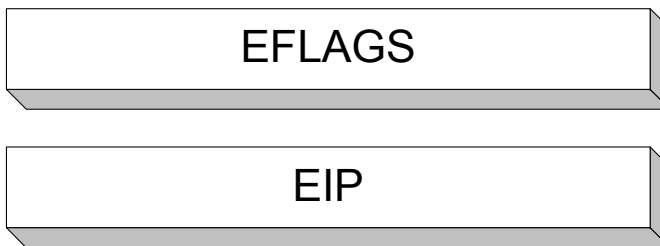
General-Purpose Registers

Named storage locations inside the CPU, optimized for speed.

32-bit General-Purpose Registers

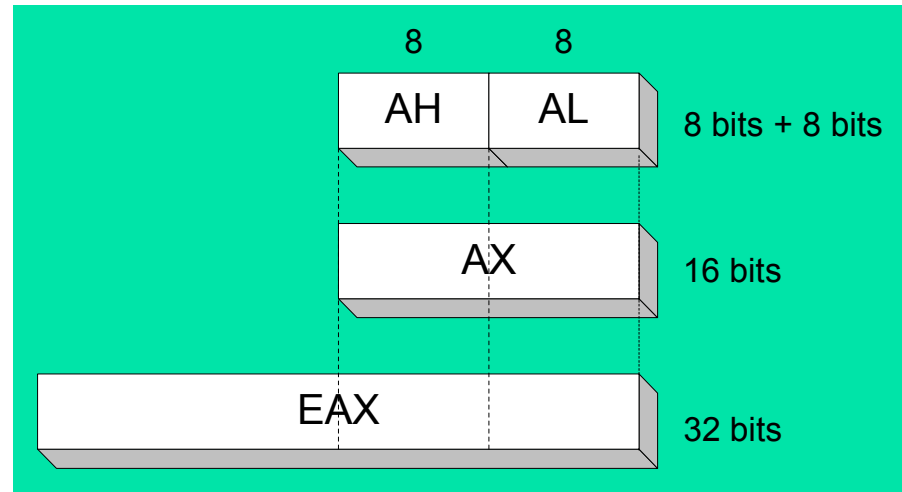


16-bit Segment Registers



Accessing Parts of Registers

- Use 8-bit name, 16-bit name, or 32-bit name
- Applies to EAX, EBX, ECX, and EDX



32-bit	16-bit	8-bit (high)	8-bit (low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL



Index and Base Registers

- Some registers have only a 16-bit name for their lower half
- The 16-bit registers are usually used only in real-address mode

32-bit	16-bit
ESI	SI
EDI	DI
EBP	BP
ESP	SP



Some specialized register uses

- General-Purpose

- EAX – accumulator (automatically used by division and multiplication)
- ECX – loop counter
- ESP – stack pointer (should never be used for arithmetic or data transfer)
- ESI, EDI – index registers (used for high-speed memory transfer instructions)
- EBP – extended frame pointer (stack)



Some specialized register uses (cont.)

- Segment
 - CS – code segment
 - DS – data segment
 - SS – stack segment
 - ES, FS, GS - additional segments
- EIP – instruction pointer
- EFLAGS
 - control flags (control CPU's operation, e.g. break, interrupt, enter 8086/protected mode)
 - Status flag
 - each flag is a single binary bit (*set* or *clear*)



Status Flags

- Carry (CF)
 - unsigned arithmetic out of range
- Overflow (OF)
 - signed arithmetic out of range
- Sign (SF)
 - result is negative
- Zero (ZF)
 - result is zero
- Auxiliary Carry (AC)
 - carry from bit 3 to bit 4 in 8-bit operand
- Parity (PF)
 - sum of 1 bits in least-significant byte is an even number

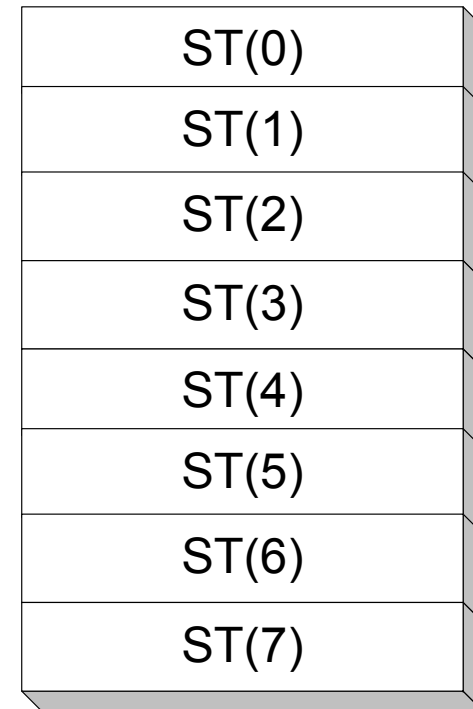


System registers

- Accessed by operating system kernel at highest privilege level, not by application programs
 - IDTR (Interrupt Descriptor Table Register)
 - GDTR (Global Descriptor Table Register)
 - LDTR (Local Descriptor Table Register)
 - Task register
 - Debug registers
 - Control registers (e.g. task switching, paging, enabling cache memory)
 - Model-specific registers (e.g. performance monitoring, checking architecture)

Floating-Point, MMX, XMM Registers

- Eight 80-bit floating-point data registers
 - ST(0), ST(1), . . . , ST(7)
 - arranged in a stack
 - used for all floating-point arithmetic
- Eight 64-bit MMX registers
- Eight 128-bit XMM registers for single-instruction multiple-data (SIMD) operations



SIMD: A single computer instruction perform the same identical action (retrieve, calculate, or store) simultaneously on two or more pieces of data



IA-32 Memory Management

- Real-address mode
- Calculating linear addresses
- Protected mode
- Multi-segment model
- Paging

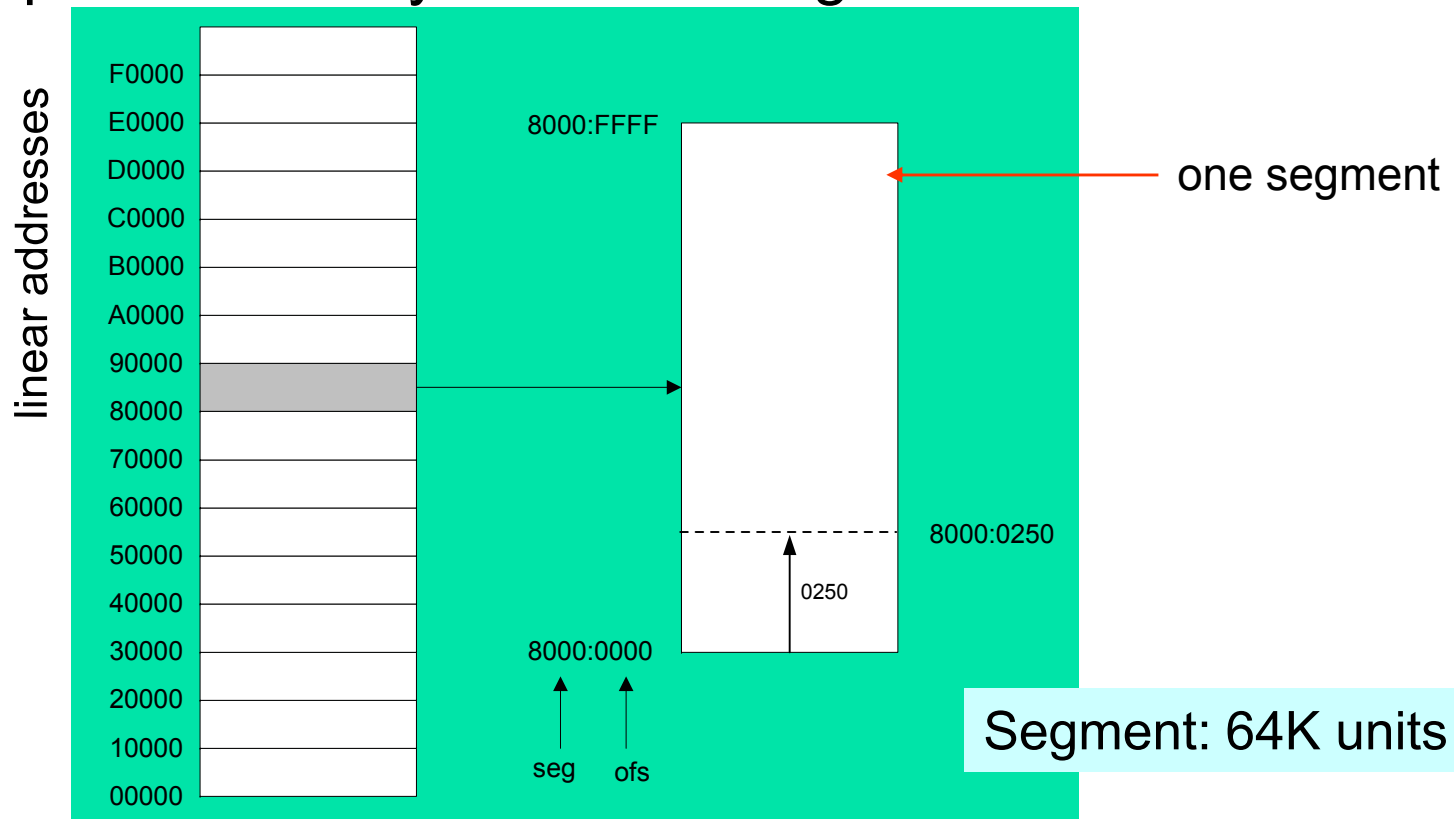


Real-Address mode

- 1 MB RAM maximum addressable
- Application programs can access any area of memory
- Single tasking
- Supported by MS-DOS operating system

Segmented Memory

- Segmented memory addressing: absolute (linear) address is a combination of a 16-bit segment value added to a 16-bit offset
- 8086 processor only has 16-bit registers





Calculating Linear Addresses

- Given a segment address, multiply it by 16 (add a hexadecimal zero), and add it to the offset
- Example: convert 08F1:0100 to a linear address

Adjusted Segment value:	0	8	F	1	0		
Add the offset:			0	1	0	0	
Linear address:			0	9	0	1	0

- A typical program has three segments: code, data and stack. Segment registers CS, DS and SS are used to store them separately.



Your turn . . .

What linear address corresponds to the segment/offset address 028F:0030?

$$028F0 + 0030 = 02920$$

Always use hexadecimal notation for addresses.



Your turn . . .

What segment addresses correspond to the linear address 28F30h?

Many different segment-offset addresses can produce the linear address 28F30h. For example:

28F0:0030, 28F3:0000, 28B0:0430, . . .



Protected Mode

- 4 GB addressable RAM
 - (00000000 to FFFFFFFFh)
- Each program assigned a memory partition which is protected from other programs
- Designed for multitasking
- Supported by Linux & MS-Windows

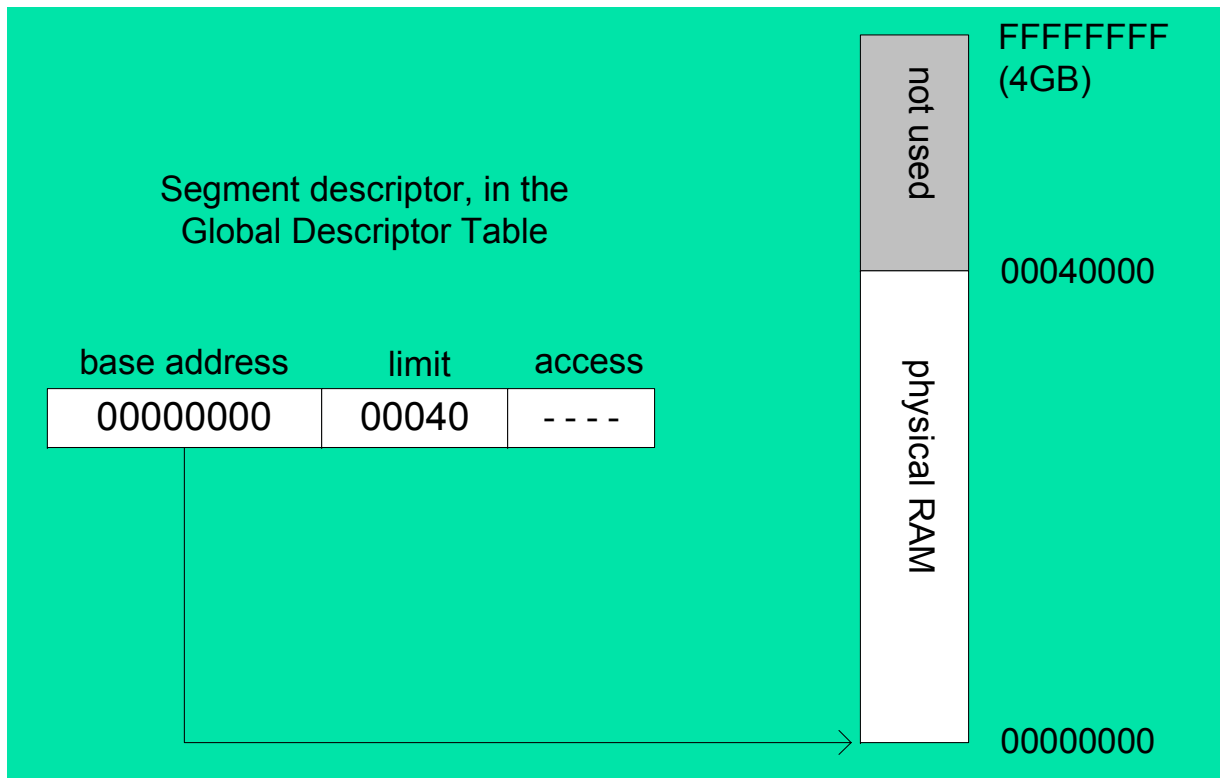


Protected Mode (cont.)

- Segments
 - variable-sized areas of memory for code & data
- Segment descriptor
 - 64-bit value identifying and describing a single memory segment
 - contains segment's base address, access rights, size limit, type, and usage
- Segment descriptor table
 - contains segment descriptors by which OS keep track of locations of individual program segments
- Segment registers
 - points to segment descriptor tables
- Program structure
 - code, data, and stack areas
 - CS, DS, SS segment descriptors
 - global descriptor table (GDT)
- MASM Programs use the Microsoft **flat** memory model

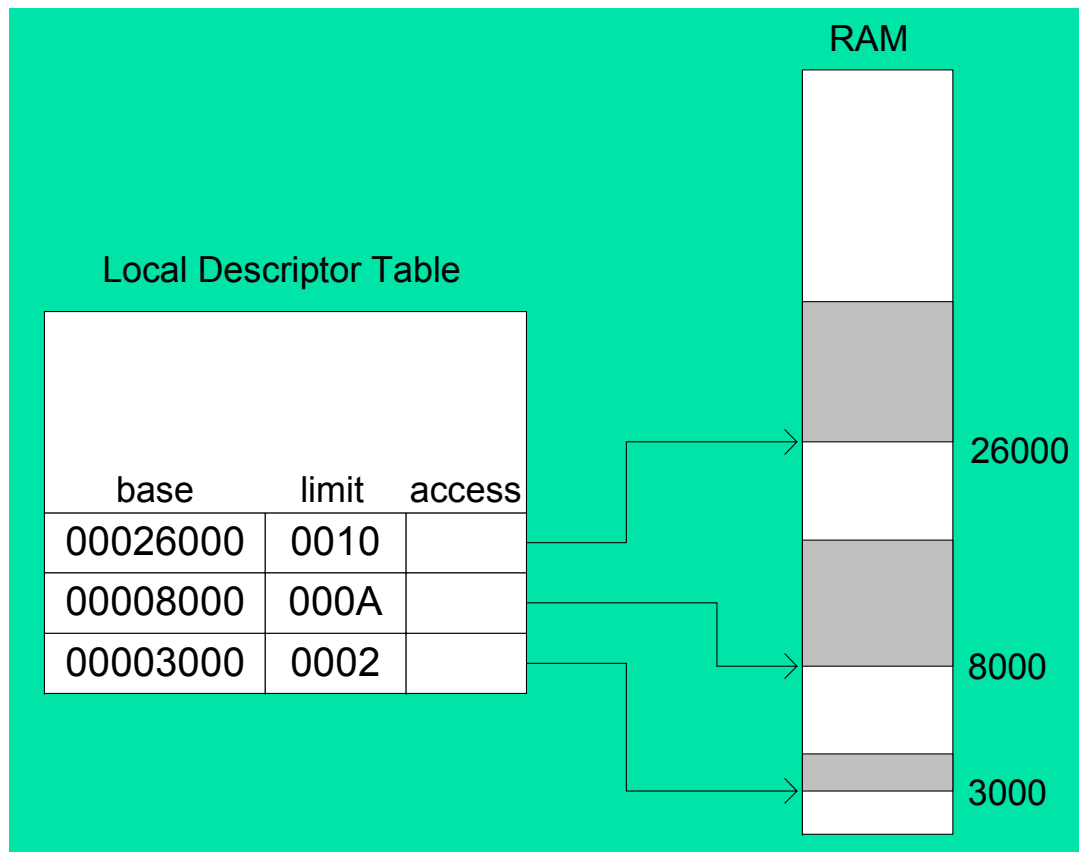
Flat Segment Model

- Single global descriptor table (GDT) whose base address is in GDTR
- Created when OS switches the processor into protected mode during boot up
- All segments mapped to entire 32-bit address space



Multi-Segment Model

- Each program has a local descriptor table (LDT)
 - holds descriptor for each segment used by the program



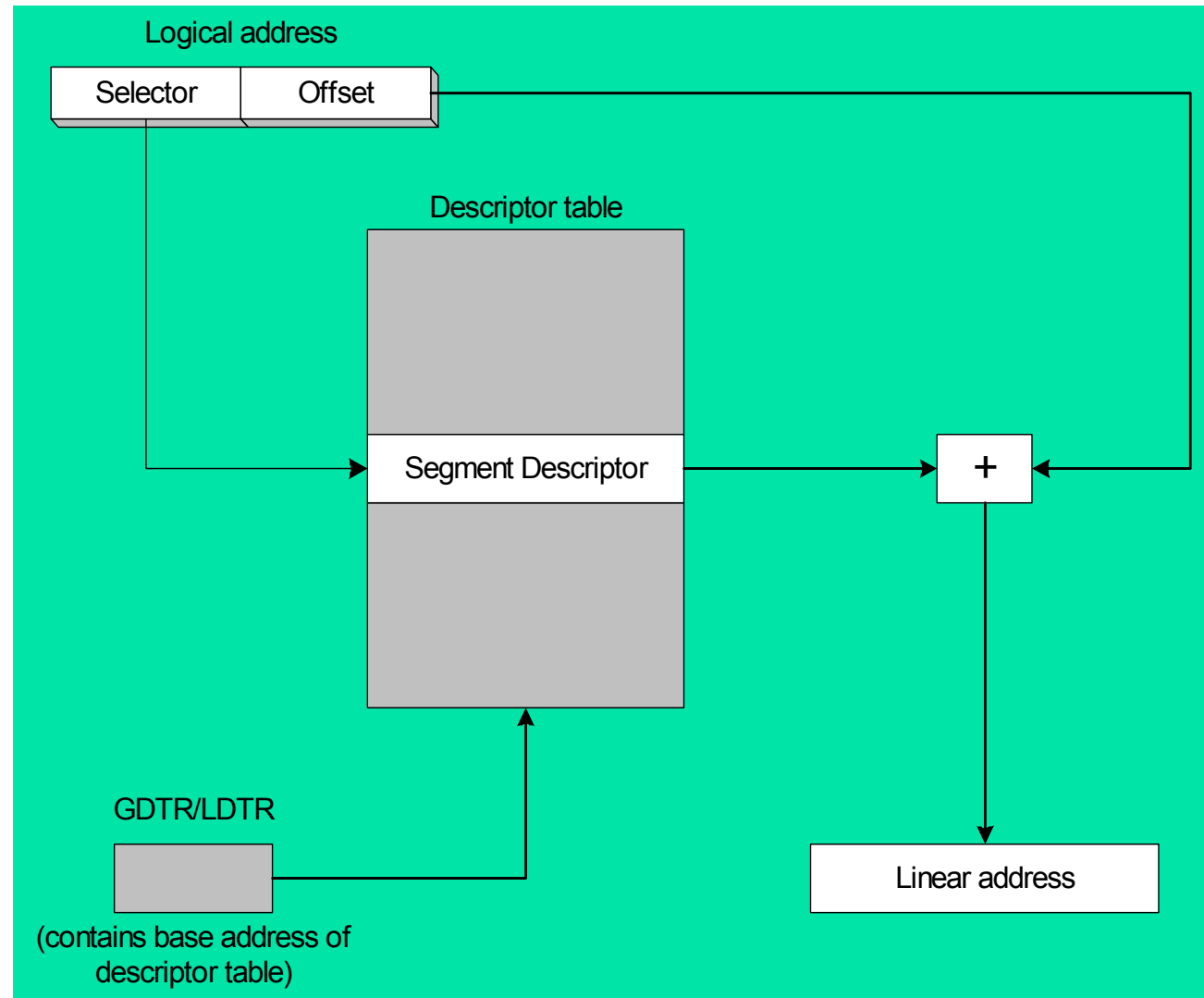


Translating Addresses (See Ch11.4)

- The IA-32 processor uses a one- or two-step process to convert a variable's logical address into a unique memory location.
- The first step combines a segment value (16-bit, segment register) with a variable's offset (32-bit) to create a **linear address**.
- The second optional step, called **page translation**, converts a linear address to a physical address.

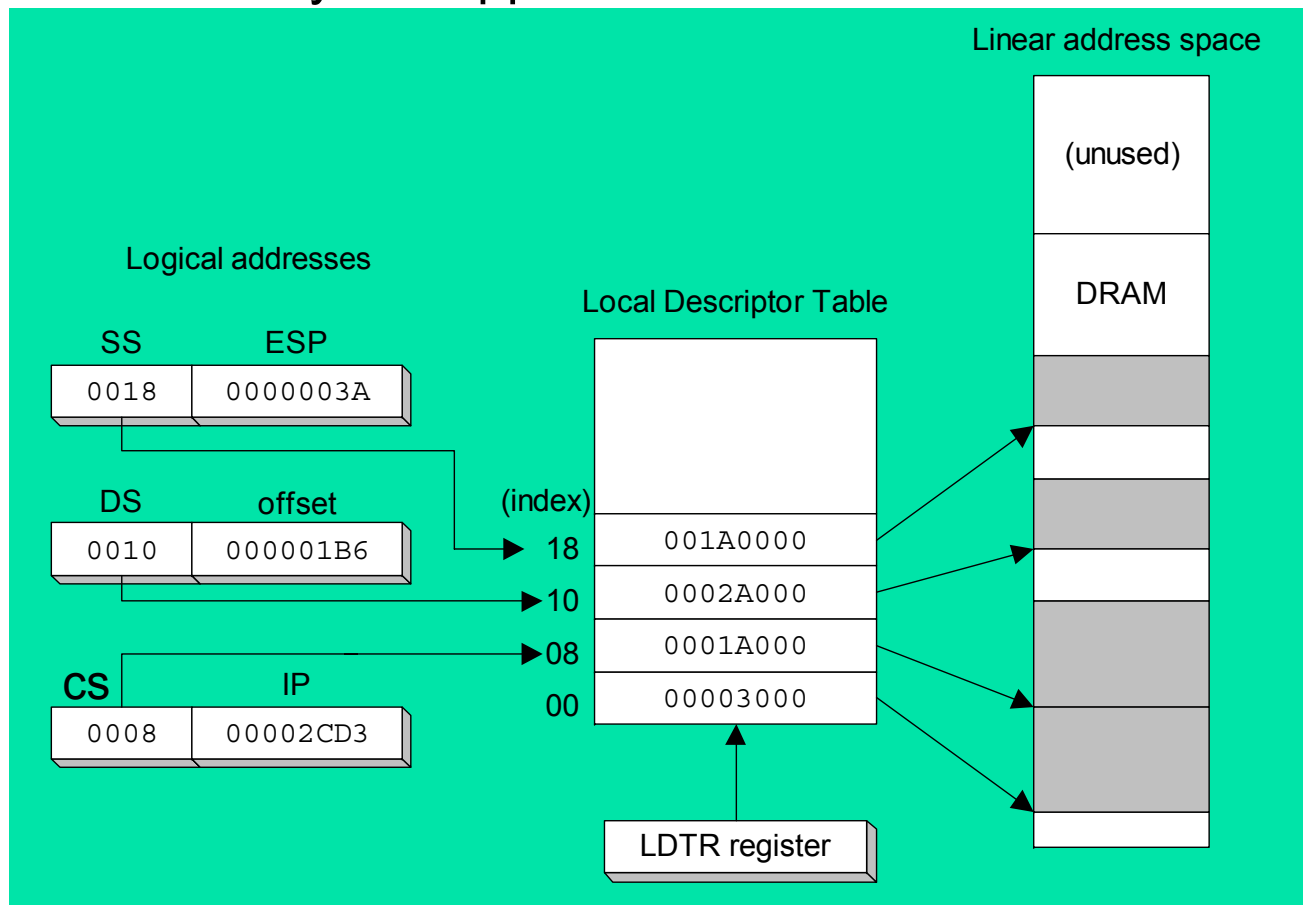
Converting Logical to Linear Address

The segment selector (16-bit) points to a **segment descriptor**, which contains the base address of a memory segment. The 32-bit offset from the logical address is added to the segment's base address, generating a 32-bit **linear address**.



Indexing into a Descriptor Table

- Each segment descriptor indexes into the program's local descriptor table (LDT)
- Each table entry is mapped to a linear address:





Virtual Memory Concepts (cont.)

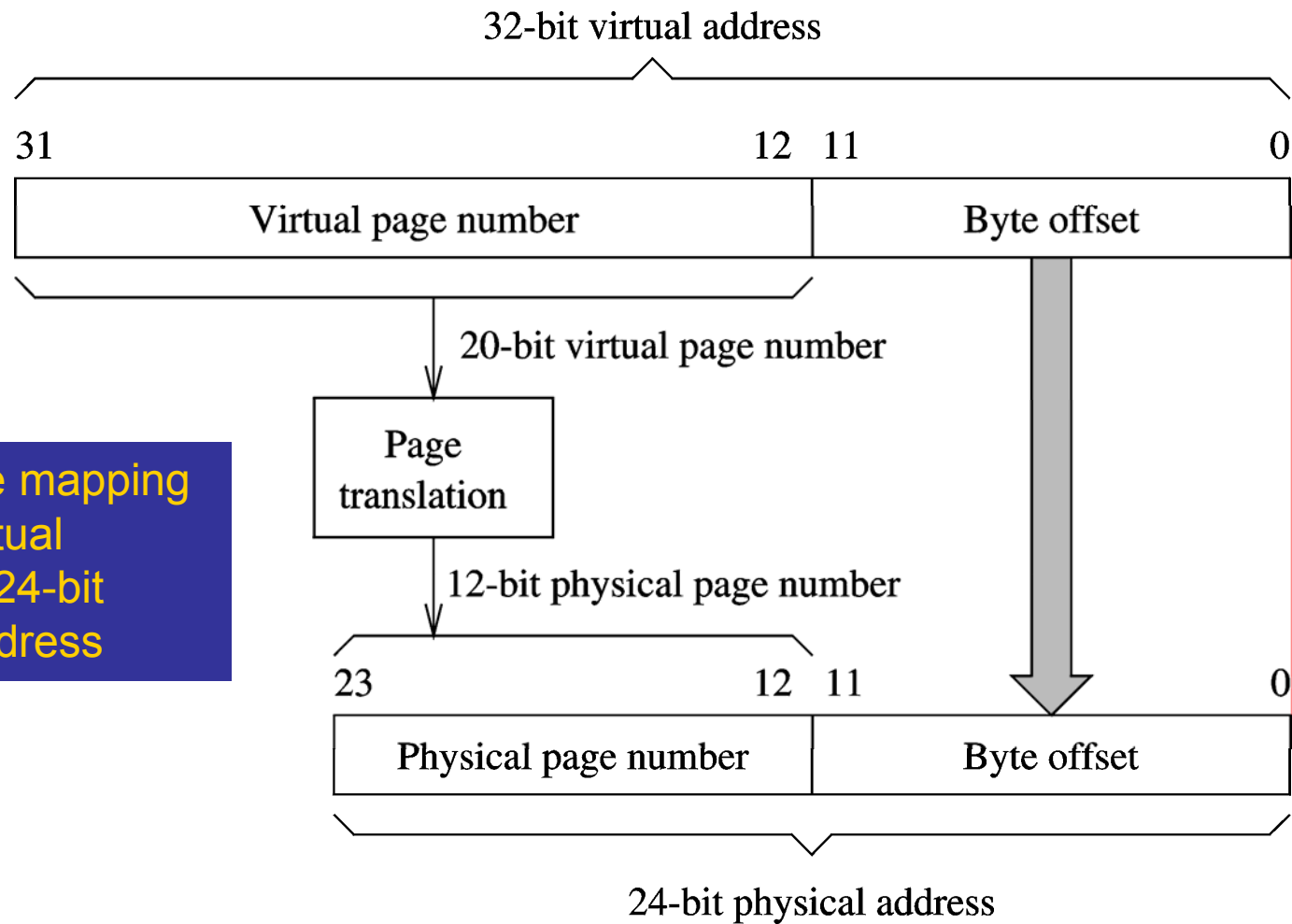
- Virtual address space is divided into fixed-size chunks
 - These chunks are called ***virtual pages***
 - Virtual address is divided into
 - Virtual page number
 - Byte offset into a virtual page
 - Physical memory is also divided into similar-size chunks
 - These chunks are referred to as ***physical pages***
 - Physical address is divided into
 - Physical page number
 - Byte offset within a page



Virtual Memory Concepts (cont.)

- Page size is similar to cache line size
- Typical page size
 - 4 KB
- Example
 - 32-bit virtual address to 24-bit physical address
 - If page size is 4 KB
 - Page offset: 12 bits
 - Virtual page number: 20 bits
 - Physical page number: 12 bits
 - Virtual memory maps 2^{20} virtual pages to 2^{12} physical pages

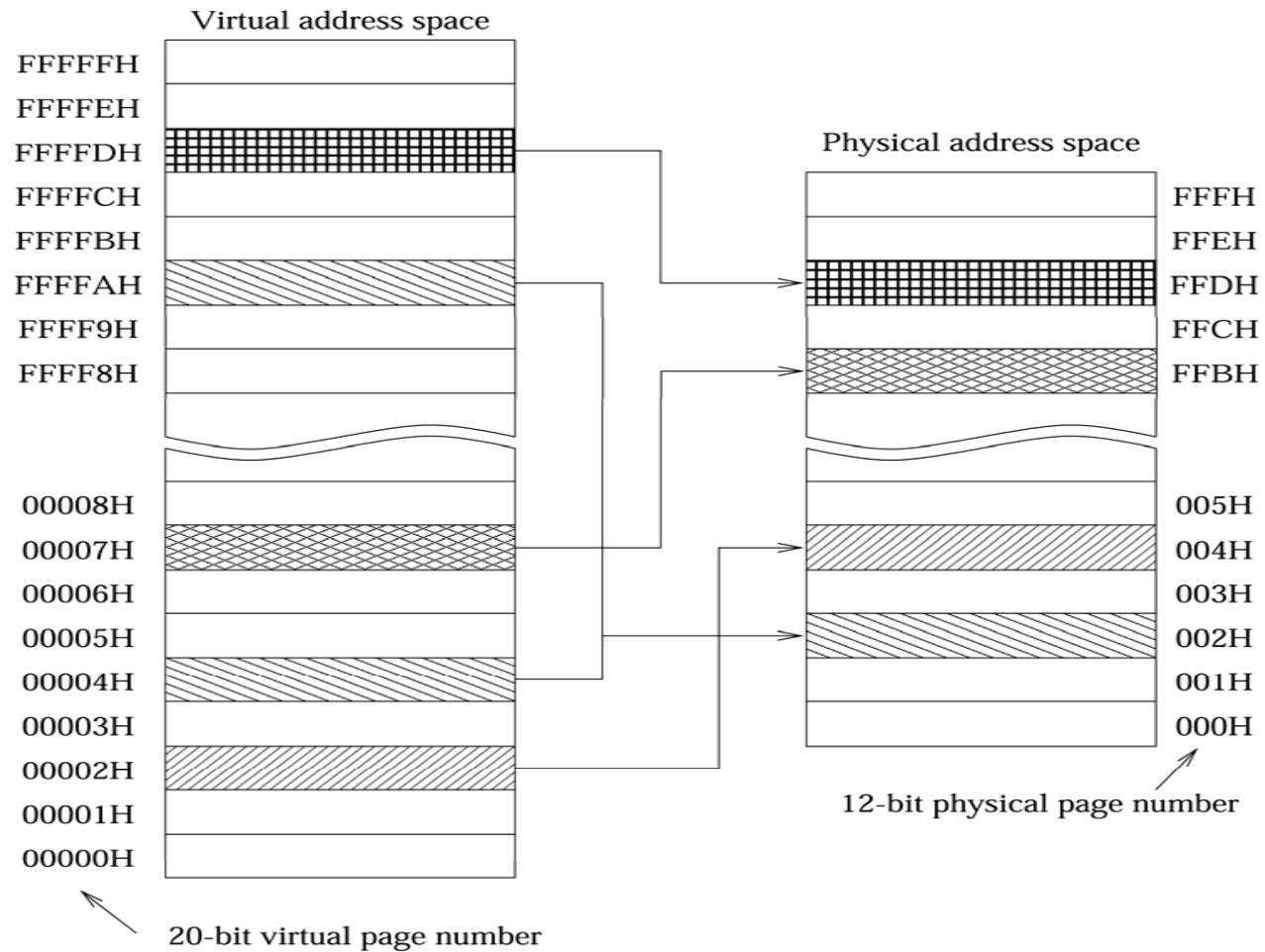
Virtual Memory Concepts (cont.)



An example mapping of 32-bit virtual address to 24-bit physical address

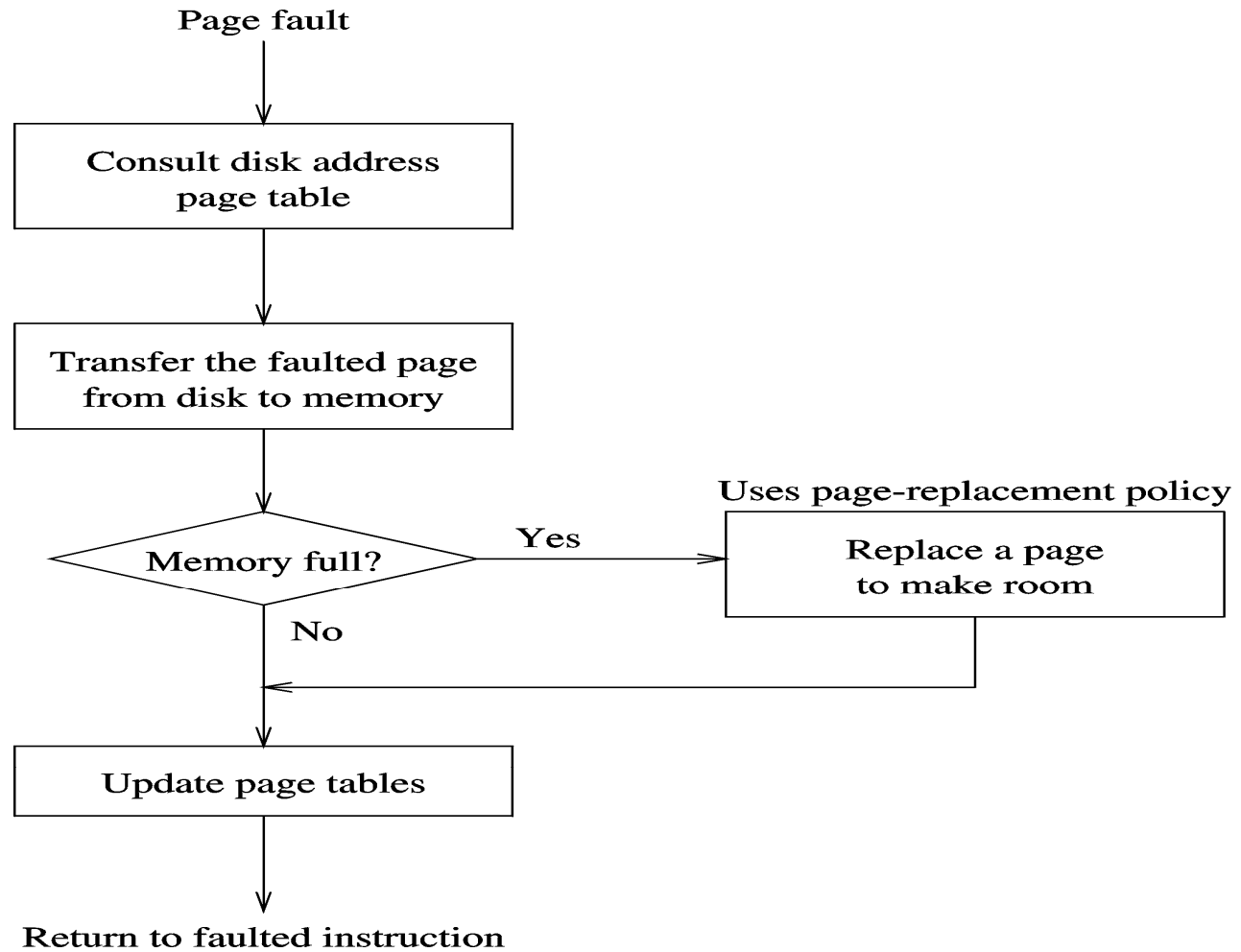
Virtual Memory Concepts (cont.)

Virtual to physical address mapping



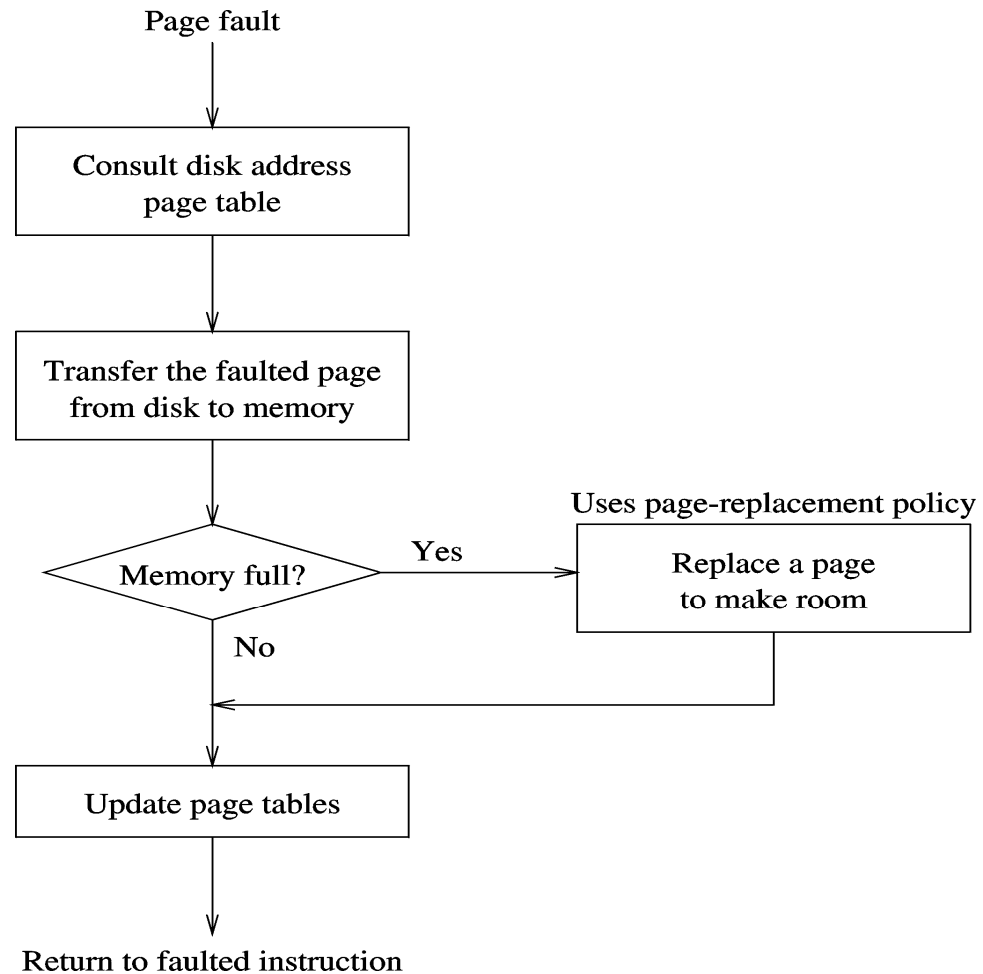
Virtual Memory Concepts (cont.)

Page fault handling routine



Virtual Memory Concepts (cont.)

- A virtual page can be
 - In main memory
 - On disk
- Page fault occurs if the page is not in memory
 - Like a cache miss
- OS takes control and transfers the page





Paging

- Supported directly by the CPU
- Divides each segment into 4096-byte blocks called **pages**
- Sum of all programs can be larger than physical memory
- Part of running program is in memory, part is on disk
- **Virtual memory manager** (VMM) – OS utility that manages the loading and unloading of pages
- As the program runs, the processor selectively unloads inactive pages from memory and loads other pages that are immediately required.

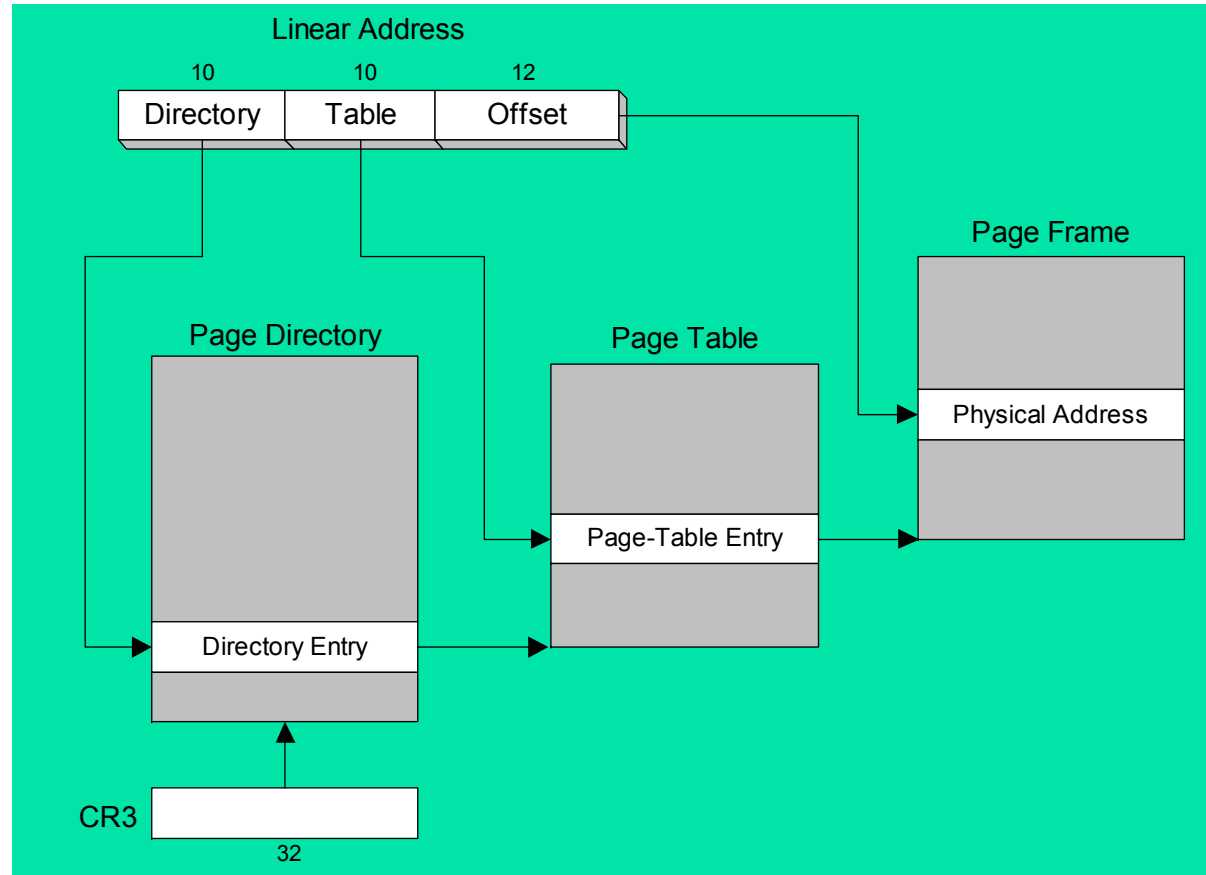


Paging (cont.)

- OS maintains **page directory** and **page tables**
- **Page translation**: CPU converts the linear address into a physical address
- **Page fault**: issued by CPU when a needed page is not in memory, and CPU interrupts the program
- OS copies the page into memory, program resumes execution

Page Translation

A linear address is divided into a page directory field, page table field, and page frame offset. The CPU uses all three to calculate the physical address.





Paging

- Supported directly by the CPU
- Divides each segment into 4096-byte blocks called **pages**
- Sum of all programs can be larger than physical memory
- Part of running program is in memory, part is on disk
- **Virtual memory manager** (VMM) – OS utility that manages the loading and unloading of pages
- **Page fault** – issued by CPU when a page must be loaded from disk



Intel Microprocessor History

- Intel 8086, 80286
- IA-32 processor family
- P6 processor family
- CISC and RISC



Early Intel Microprocessors

- Intel 8080

- 64K addressable RAM
- 8-bit registers
- CP/M operating system
- S-100 BUS architecture
- 8-inch floppy disks!

- Intel 8086/8088

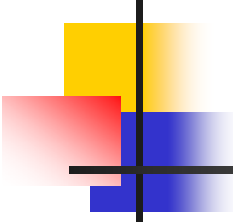
- IBM-PC Used 8088
- 1 MB addressable RAM
- 16-bit registers
- 16-bit data bus (8-bit for 8088)
- separate floating-point unit (8087)



The IBM-AT

- Intel 80286

- 16 MB addressable RAM
- Protected memory
- several times faster than 8086
- introduced IDE bus architecture
- 80287 floating point unit



Intel IA-32 Family

- Intel386

- 4 GB addressable RAM, 32-bit registers, paging (virtual memory)

- Intel486

- instruction pipelining

- Pentium

- superscalar, 32-bit address bus, 64-bit internal data path



Intel P6 Family

- Pentium Pro
 - advanced optimization techniques in microcode
- Pentium II
 - MMX (multimedia) instruction set
- Pentium III
 - SIMD (streaming extensions) instructions
- Pentium 4 and Xeon
 - Intel NetBurst micro-architecture, tuned for multimedia



What's Next

- General Concepts
- IA-32 Processor Architecture
- **IA-32 Memory Management**
- Components of an IA-32 Microcomputer
- Input-Output System



What's Next

- General Concepts
- IA-32 Processor Architecture
- IA-32 Memory Management
- **Components of an IA-32 Microcomputer**
- Input-Output System



Components of an IA-32 Microcomputer

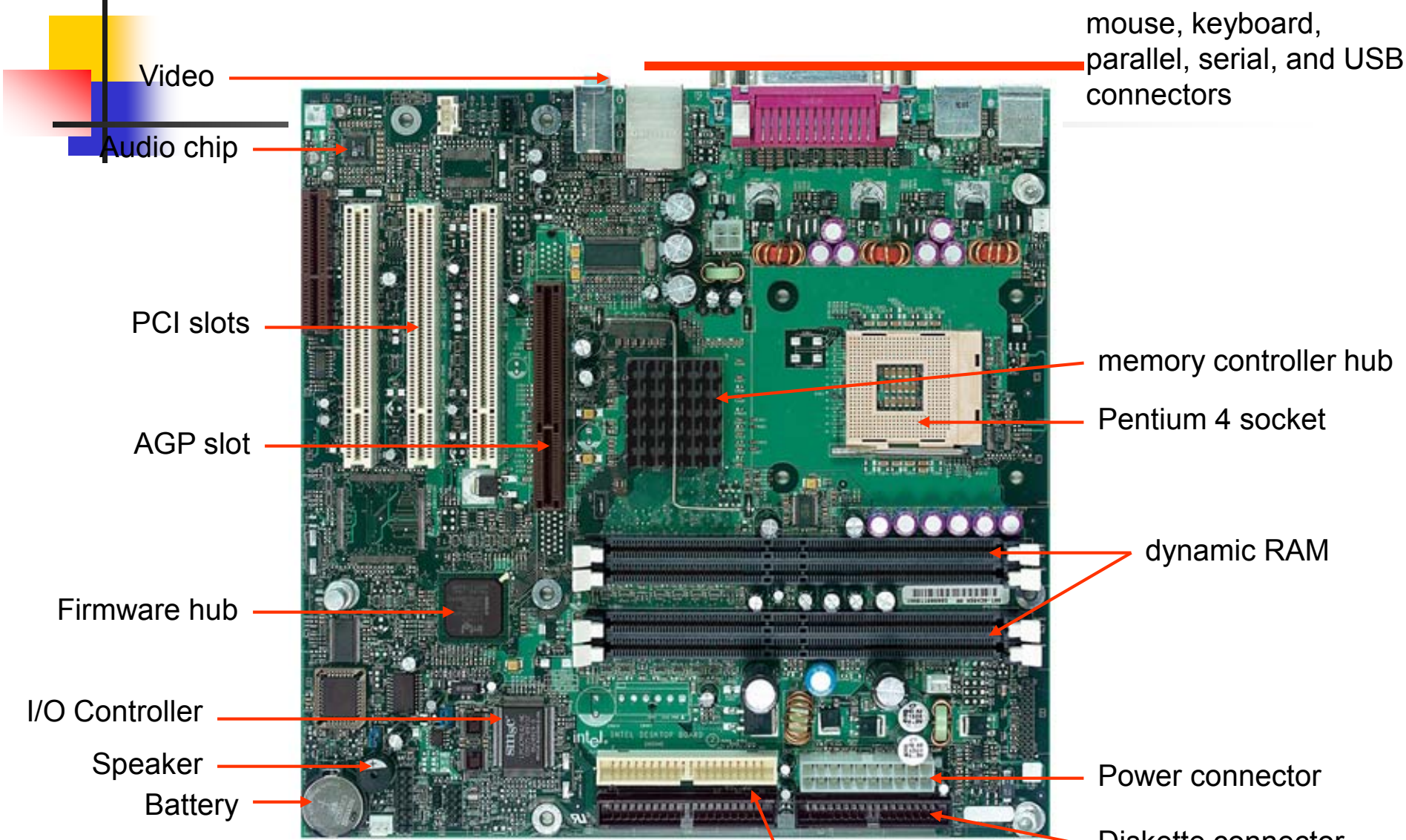
- Motherboard
- Video output
- Memory
- Input-output ports



Motherboard

- CPU socket
- External cache memory slots
- Main memory slots
- BIOS chips
- Sound synthesizer chip (optional)
- Video controller chip (optional)
- IDE, parallel, serial, USB, video, keyboard, joystick, network, and mouse connectors
- PCI bus connectors (expansion cards)

Intel D850MD Motherboard



Source: Intel® Desktop Board D850MD/D850MV Technical Product Specification

IDE drive connectors

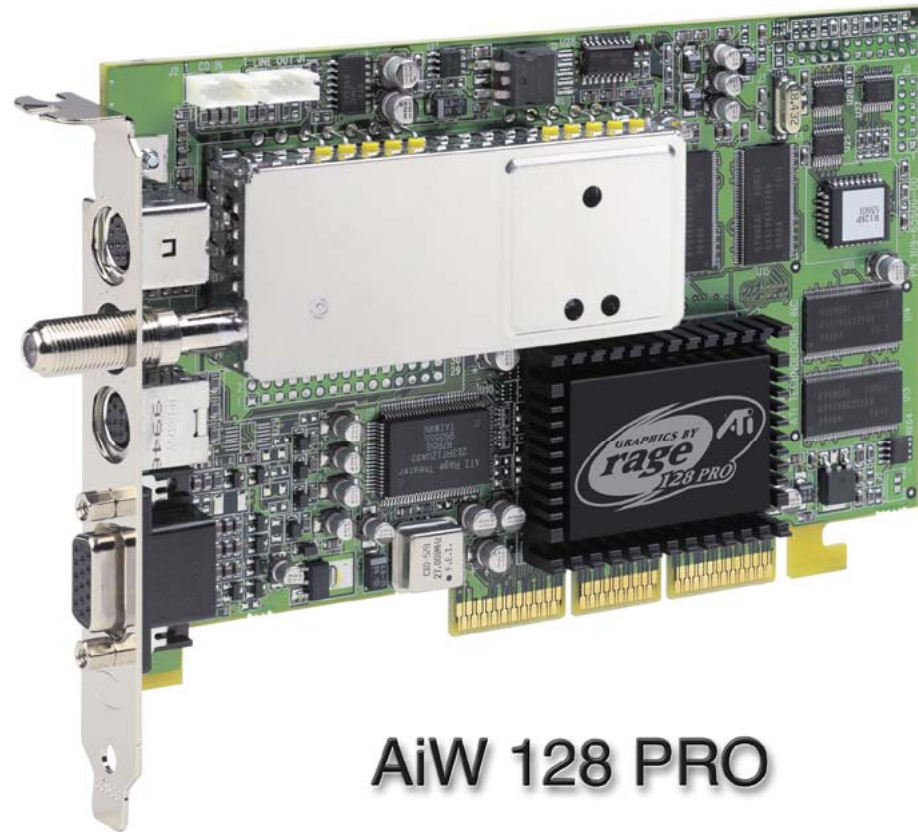


Video Output

- Video controller
 - on motherboard, or on expansion card
 - AGP ([accelerated graphics port technology](#))*
- Video memory (VRAM)
- Video CRT Display
 - uses raster scanning
 - horizontal retrace
 - vertical retrace
- Direct digital LCD monitors
 - no raster scanning required

* This link may change over time.

Sample Video Controller (ATI Corp.)



AiW 128 PRO



Memory

- ROM
 - read-only memory
- EPROM
 - erasable programmable read-only memory
- Dynamic RAM (DRAM)
 - inexpensive; must be refreshed constantly
- Static RAM (SRAM)
 - expensive; used for cache memory; no refresh required
- Video RAM (VRAM)
 - dual ported; optimized for constant video refresh
- CMOS RAM
 - complimentary metal-oxide semiconductor
 - system setup information
- See: [Intel platform memory](#) (Intel technology brief: link address may change)



Input-Output Ports

- USB (universal serial bus)
 - intelligent high-speed connection to devices
 - up to 480 megabits/second (USB version 2.0)
 - USB hub connects multiple devices
 - *enumeration*: computer queries devices
 - supports *hot* connections
- Parallel
 - short cable, high speed
 - common for printers
 - bidirectional, parallel data transfer
 - Intel 8255 controller chip



Input-Output Ports (cont)

- Serial

- RS-232 serial port
- one bit at a time
- uses long cables and modems
- 16550 UART (universal asynchronous receiver transmitter)
- programmable in assembly language



What's Next

- General Concepts
- IA-32 Processor Architecture
- IA-32 Memory Management
- Components of an IA-32 Microcomputer
- **Input-Output System**

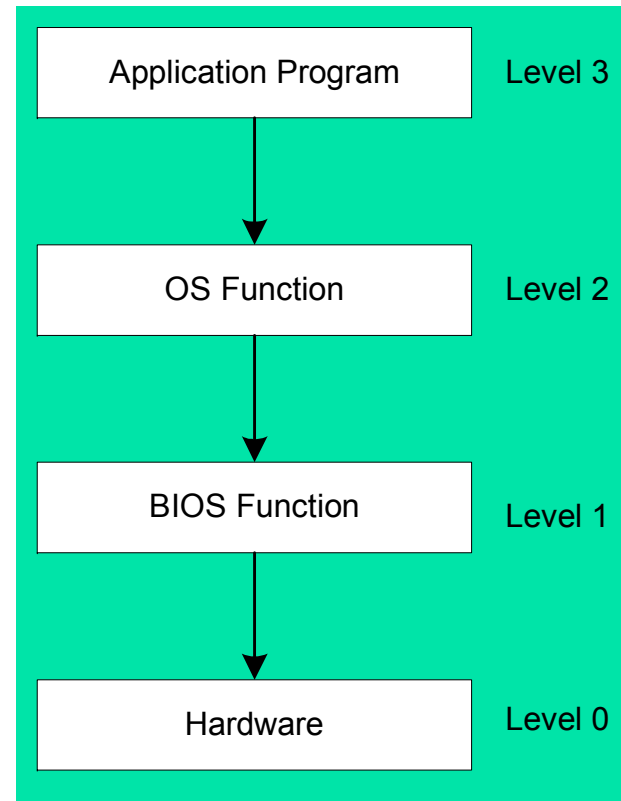


Levels of Input-Output

- Level 3: Call a library function (C++, Java)
 - easy to do; abstracted from hardware; details hidden
 - slowest performance
- Level 2: Call an operating system function
 - specific to one OS; device-independent
 - medium performance
- Level 1: Call a BIOS (basic input-output system) function
 - may produce different results on different systems
 - knowledge of hardware required
 - usually good performance
- Level 0: Communicate directly with the hardware
 - May not be allowed by some operating systems

Displaying a String of Characters

When a HLL program displays a string of characters, the following steps take place:



ASM Programming levels

ASM programs can perform input-output at each of the following levels:

