

Operating System Principles

操作系统原理（v2）

Chapter 02

Operating System Environment

操作系统运行环境

LeeXudong

—Nankai Univ. SE.



Objectives

- 计算机系统组成
- 中央处理器 (CPU)
- 存储系统
- 中断机制
- I/O 技术
- 时钟
- 操作系统的启动



计算机系统的组成

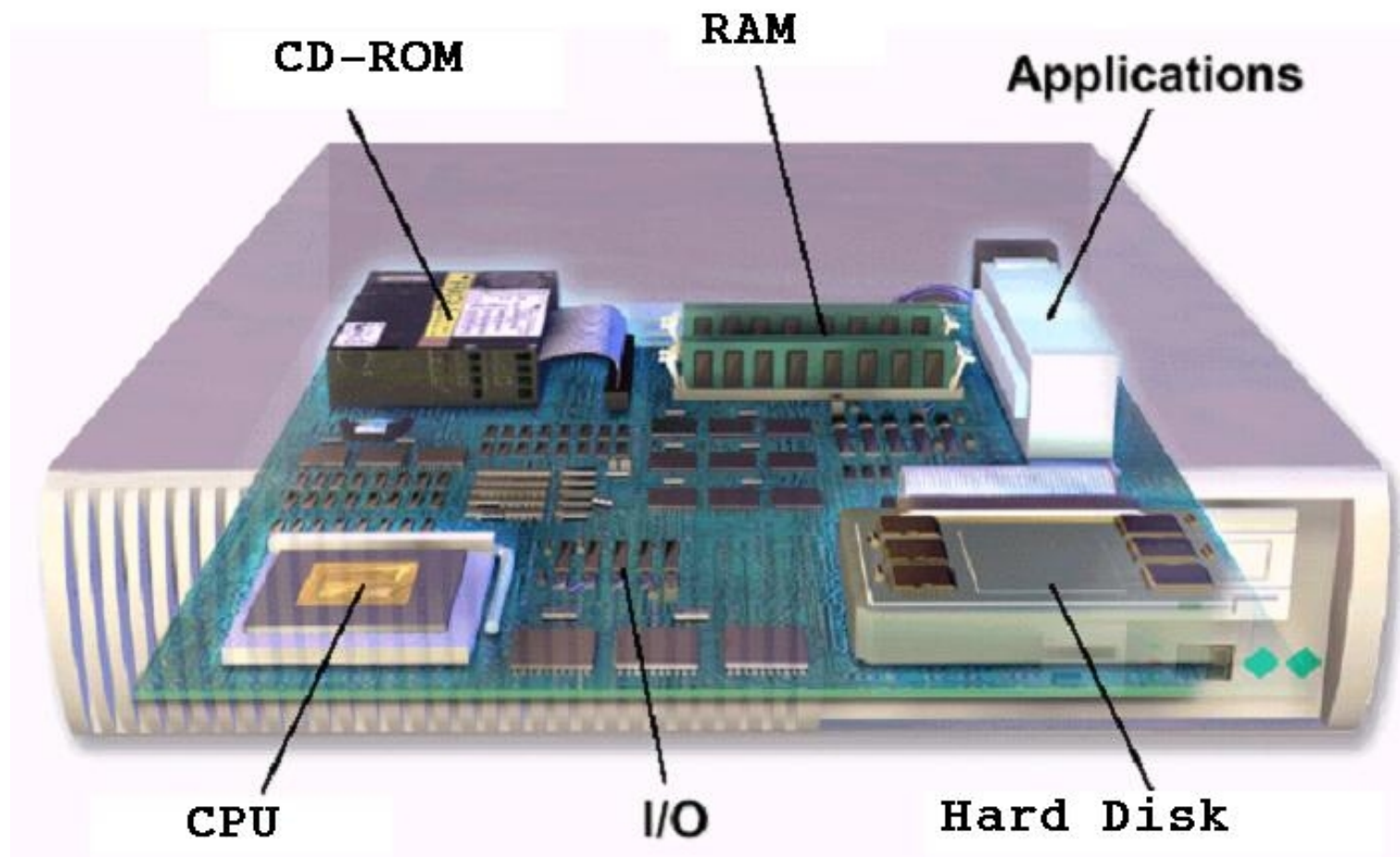
- 硬件子系统

- CPU, 主存储器, 外存储器 (磁盘、磁带等)
- 各种类型的输入 / 输出设备 (键盘、鼠标、显示器、打印机)

- 软件子系统

- 系统软件：操作系统、编译系统
- 支撑软件
 - 数据库、网络通信程序、多媒体支持软件、硬件接口程序、实用软件工具、软件开发工具
- 应用软件：字处理软件、游戏软件等

硬件结构





- EDVAC(电子离散变量自动计算机)
- 1822 年 Babbage 构想
- 1945 年 John von Neumann 采用

数据总线





冯·诺伊曼型计算机的特点

- 机器以**运算器**为中心，I/O 设备与存储器之间的数据传送都要经过运算器
- 由控制器集中控制各部分的操作及其相互之间的联系
- 采用存储程序的思想，在程序控制下，机器的操作按顺序执行指令，指令计数器指明要执行的指令在存储器中的地址，分支则由程序中的转移指令实现
- **程序指令**和**处理数据**在存储器中同等对待，均采用二进制编码
- 存储器是一个顺序线形编址的一维空间，每个存储单元的二进制的位数是固定的，地址是唯一定义的
- 指令的形式为低级的机器语言（二进制语言），驱动机器进行操作。一般分为操作码和操作数两部分
- 软件与硬件完全分开。硬件结构采用固定性逻辑，即其功能是不变的，完全依靠编制软件来适应不同的应用需要



中央处理器 CPU

- CPU 由“算术 - 逻辑”运算单元 (ALU)、控制单元 (CU)、一系列的寄存器以及高速缓存构成
 - 运算器包括算术运算和逻辑运算，它是计算机计算的**核心**
 - 控制器负责控制程序运行的流程，包括取指令、译码、维护 CPU 状态、CPU 与内存的交互等
 - 寄存器用于暂存数据、地址以及指令信息。在计算机的存储系统中，寄存器具有最快的访问速度



CPU 的寄存器 1/2

- 两大类寄存器：用户可见寄存器、控制和状态寄存器
- 1. 用户可见寄存器
 - 段寄存器 *
 - CS (Code Segment), DS (Data Segment)
 - SS (Stack Segment), ES (Extra Segment), FS, GS
 - 数据寄存器 (通用寄存器 ,16 位)Data Register
 - AX 累加器 , 算术运算和所有 I/O 指令的数据传送 , EAX
 - BX 通用寄存器和基址寄存器 , EBX
 - CX 通用寄存器和循环 loop 等计数器 , ECX
 - DX 通用寄存器 (双字节运算) 和某些 I/O 操作 , EDX
 - 指针以及变址寄存器 (也可作数据运算)
 - SP:Stack Pointer 堆栈指针寄存器 , ESP
 - BP:Base Pointer 基址指针寄存器 , EBP
 - SI:Source Index 源变址寄存器 , ESI
 - DI:Destination Index 目的变址寄存器 , EDI



CPU 的寄存器 2/2

- 2. 控制和状态寄存器
 - IP(Instruction Pointer, or PC) 指令指针寄存器 , EIP
 - CS:EIP
 - IR (Instruction Register) 指令寄存器
 - PSW(Program Status Word) 程序状态字 , **EFLAGS**
 - OF(Overflow Flag), SF(Sign Flag),
 - ZF(Zero Flag), CF(Carry Flag), PF(Parity Flag),
 - AF(Auxiliary carry Flag), DF(Direction Flag),
 - IF(Interrupt Flag), TF(Trap Flag)
- x86
 - Complex Instruction Set Computing, CISC
- PowerPC
 - Reduced Instruction Set Computing, RISC



“算术 - 逻辑”运算单元

■ ALU

- 一个非常快的计算器，包括一组通用寄存器和状态寄存器
- 算术：加法、减法、乘法、除法
- 逻辑：比较、逻辑与、逻辑或、逻辑非
- 通用寄存器：32~64 个，每个寄存器可保存 32bits
- ？浮点运算单元？

■ 例如： $a=b+c; d=a-100$ 的汇编指令

load R3,b

load R4,c

add R3,R4

store R3,a

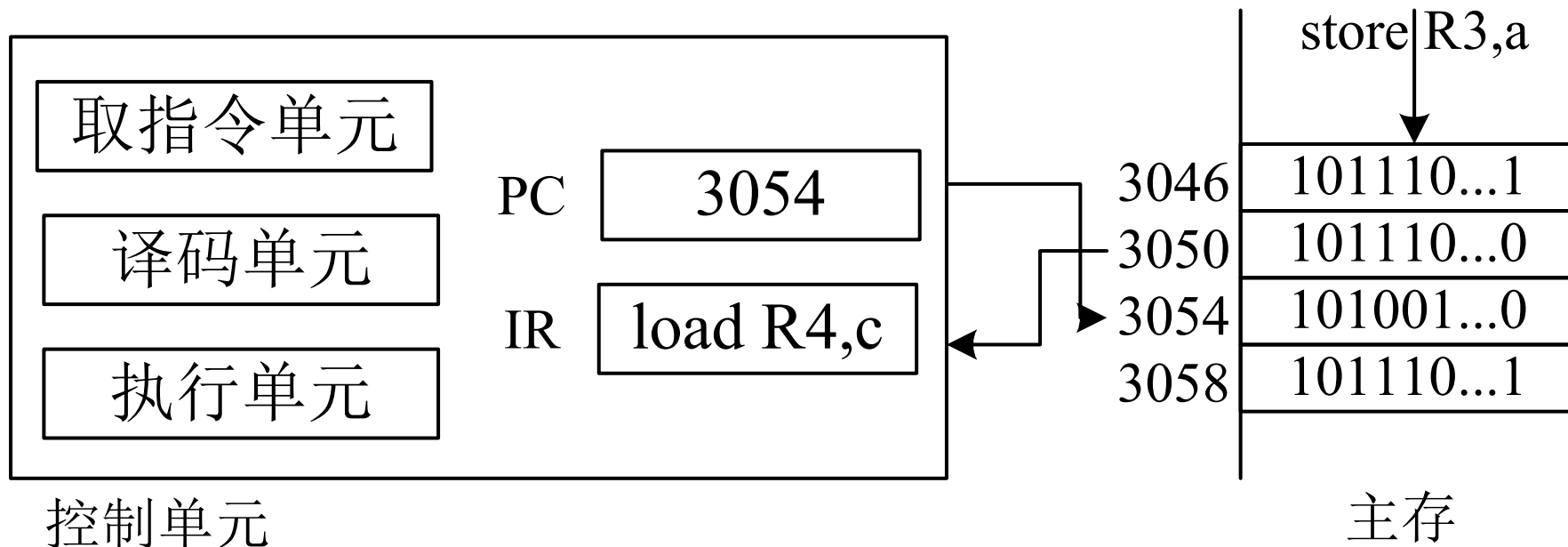
load R4, =100

subtract R3,R4

store R3,d

控制单元 CU 1/2

- 程序计数器 Program Counter, PC
- 指令寄存器 Instruction Register, IR



程序计数器、指令寄存器和主存



控制单元 CU 2/2

- 硬件进程 hardware process
- 硬件取指 - 执行周期

PC=<machine start address>;

IR=memory[PC];

haltFlag=CLEAR;

while(haltFlag not SET during execution){

PC=PC+1;

execute(IR);

IR=memory[PC];

};

- 时钟周期（晶体管、同步数字逻辑）
 - 时钟频率是决定 ALU 执行速度的重要因素
 - 6MHz: 0.167×10^{-6} 秒, 即 0.167 微秒, 167ns
 - 2GHz: 0.4ns



Registers: example 1/3

- 调试状态下改变指令执行顺序

```
int main(int argc, char* argv[])
```

```
{
```

```
    printf("Hello World!\n");
```

```
    printf("AAA\n");
```

/ 不改变源程序情况下，执行该代码时不输出 **BBB***/*

```
    printf("BBB\n");
```

```
    printf("CCC\n");
```

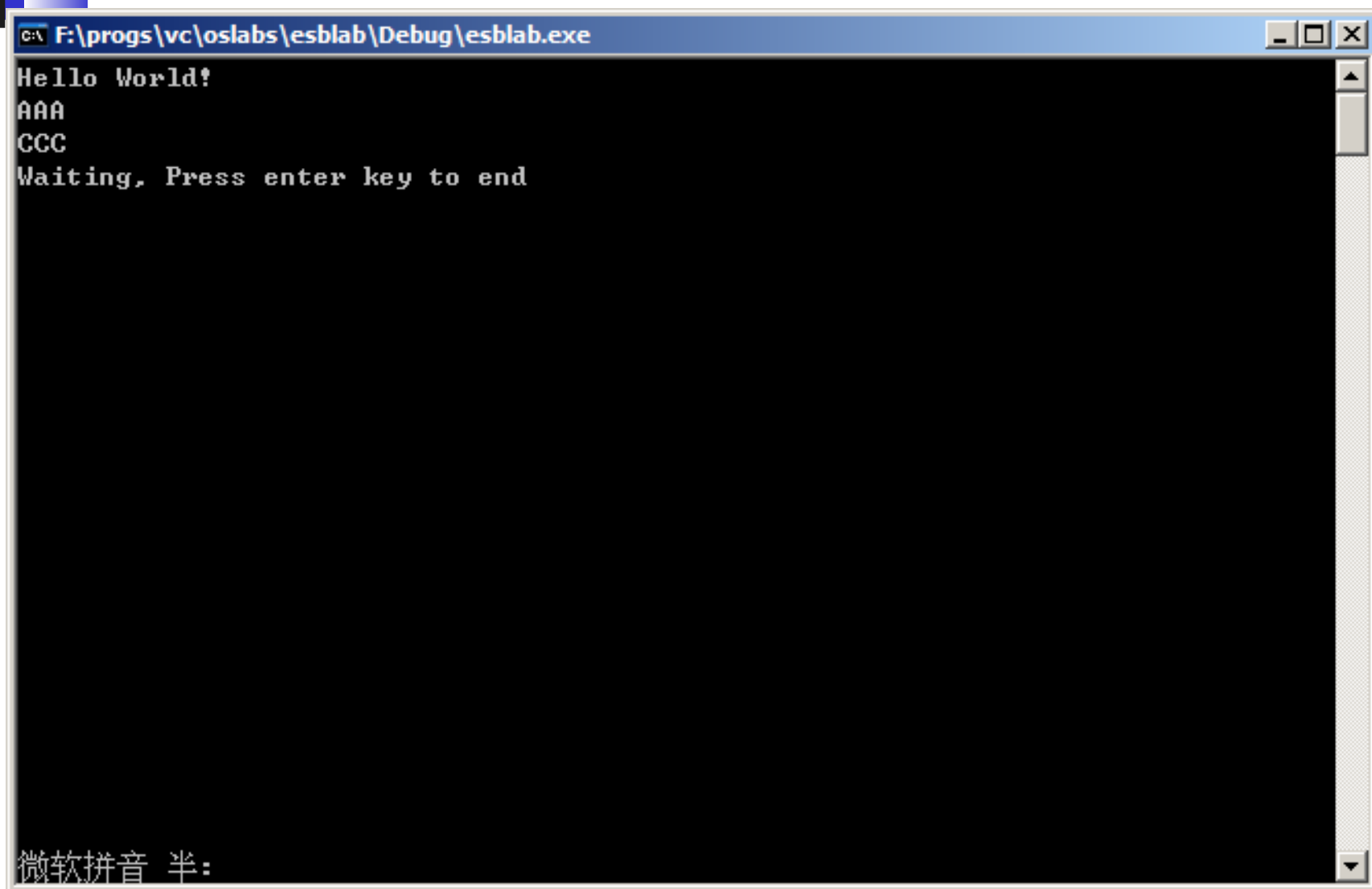
```
    printf("Waiting, Press enter key to end");
```

```
    getchar();
```

```
    return 0;
```

```
}
```

Registers: example 2/3



```
C:\F:\progs\vc\oslabs\esblab\Debug\esblab.exe
Hello World!
AAA
CCC
Waiting, Press enter key to end

微软拼音 半:
```

Registers: example 3/3

eslab - Microsoft Visual C++ [break] - [Disassembly]

File Edit View Insert Project Debug Tools Window Help

[Globals] [All global members] main

```
0040BE81 mov     eax,0CCCCCCCCh
0040BE86 rep stos dword ptr [edi]
8:      printf("Hello World!\n");
0040BE88 push    offset string "Hello World!\n" (00420fc0)
0040BE8D call    printf (00401090)
0040BE92 add     esp,4
9:      printf("AAA\n");
0040BE95 push    offset string "AAA\n" (00420034)
0040BE9A call    printf (00401090)
0040BE9F add     esp,4
10:     printf("BBB\n");
0040BEA2 push    offset string "BBB\n" (0042002c)
0040BEA7 call    printf (00401090)
0040BEAC add     esp,4
11:     printf("CCC\n");
0040BEAF push    offset string "CCC\n" (00420024)
```

Registers

EAX = 00000000 EBX = 7FFD9000
ECX = 00422A60 EDX = 00422A60
ESI = 00000000 EDI = 0012FF80
EIP = 0040BE9A ESP = 0012FF2C
EBP = 0012FF80 EFL = 00000216 CS = 001B
DS = 0023 ES = 0023 SS = 0023 FS = 003B
GS = 0000 OV=0 UP=0 EI=1 PL=0 ZR=0 AC=1
PE=1 CY=0
ST0 = +0.0000000000000000e+0000

计算器

程序员 记忆

4,325,428 存储器中未保存数据

HEX 42 0034
DEC 4,325,428
OCT 20 400 064
BIN 0100 0010 0000 0000 0011 0100

QWORD MS

按位 位移

A	<<	>>	CE	<
B	()	%	÷
C	7	8	9	×
D	4	5	6	-
E	1	2	3	+
F	+/-	0	.	=

Address: 0x0012ff34

0012FF24	80 FF 12 00
0012FF28	92 BE 40 00
0012FF2C	34 00 42 00	4.B.
0012FF30	00 00 00 00
0012FF34	00 00 00 00
0012FF38	00 90 FD 7F
0012FF3C	CC CC CC CC
0012FF40	CC CC CC CC

Name Value

(int*)(esp)	0x0012ff2c
	4325428

Watch1 Watch2 Watch3 Watch4



CPU 的工作模式

- 实模式
 - 8086/8088 模式
 - 1MB 的存储空间 $0x00000 \sim 0xFFFFF$
 - 段地址 + 段内偏移量
- 保护模式
 - 分段模式、分页模式
 - 4GB 的存储空间
 - 选择子所指的基地址 + 偏移量
- 虚模式



- The diagram illustrates the 8086 address calculation and segment mapping. On the left, the 20-bit physical address is calculated by adding the 16-bit segment address (bits 15-0, value 0000) and the 16-bit offset address (bits 19-0, value 0). This results in a 20-bit physical address. On the right, the segment registers (CS, DS, SS, ES) are mapped to 64K memory banks. CS is mapped to 64K code (01500H), DS to 64K data (42000H), SS to 64K stack (1CD00H), and ES to 64K additional data (B0000H). The segment registers are labeled as '段寄存器' (segment registers) and the memory banks as '存储器' (memory).

寄存器	地址	范围
CS	0150H	64K 代码 (01500H)
DS	4200H	64K 数据 (42000H)
SS	1CD0H	64K 堆栈 (1CD00H)
ES	B000H	64K 附加数据 (B0000H)



CPU 指令与寻址方式 1/2

■ 指令的格式

- 操作码 操作数 ... 操作数 ;(op)dw (mod)(reg)(rm)

■ CPU 指令实例

- ADD CL,BH
- 其机器指令为: *00000010 11001111*
- 即 02CFH

■ 寻址方式

- 立即寻址方式 `mov AX,3064H` ; 则 (AX)=3064H
- 寄存器寻址方式 `mov AX,BX` ; 若 (BX)=1234H, 则 (AX)=1234H
- 直接寻址方式
 - `mov AX, [2000H]` ; 若 (DS)=3000H, 则 (AX)= 内存 32000H 的值
- ...



CPU 指令与寻址方式 2/2

■ 寻址方式 (续)

■ 寄存器间接寻址方式

- `mov AX,[BX]` ; 若 $(DS)=2000H$, 则 $(BX)=1000H$, 则 (AX) = 内存 $21000H$ 的值

■ 寄存器相对寻址方式

- `mov AX,COUNT[SI]` ; 若 $(DS)=\mathbf{3000H}$, 则 $(SI)=2000H, COUNT=3000H$
- 内存地址 $\mathbf{30000}+2000+3000=35000H$, (AX) = 内存 $35000H$ 的值

■ 基址变址寻址方式

- `mov AX,[BX][DI]` ; $[16d*(DS)+(BX)+(DI)]$
- `mov AX,ES:[BX][SI]`

■ 相对基址变址寻址方式

- `mov AX,MASK[BX][SI]` ; $[16d*(DS)+(BX)+(SI)+(MASK)]$



CPU 特权指令和非特权指令

- 目的

- 用于多用户或多任务的多道程序设计环境中

- 特权指令

- 在指令系统中那些只能由操作系统使用的指令，例如设置程序状态字、启动某设备、设置中断屏蔽、设置时钟指令、清主存指令、建立存储保护指令等，这些指令不允许一般的用户使用的，否则就有可能使系统陷入混乱。因此必须区分特权指令和非特权指令。

- 非特权指令

- 用户只能使用非特权指令
 - 操作系统既能执行特权指令也能执行非特权指令



非特权状态向特权状态的转换

■ 非特权状态向特权状态的转换

- 如果一个用户程序需要使用特权指令，一般将引起一次处理器状态的切换，这时处理器通过特殊的机制，将处理器状态切换到操作系统运行的特权状态，然后将处理权移交给操作系统中的一段特殊代码。这一过程通常称为陷入 **trap**。
- 如果某计算机的指令系统中不能区分特权指令和非特权指令，那么在这样的硬件环境下要设计出一个具有多道程序运行的 **OS** 非常困难。



CPU 保护模式下的状态

■ 处理器的状态

- 管态 (supervisor mode)
 - 也称特权态、系统态、核心态；操作系统管理程序运行状态
- 目态 (user mode)
 - 也称普通态、用户态；用户程序和操作系统部分程序的运行状态

■ Intel CPU x86

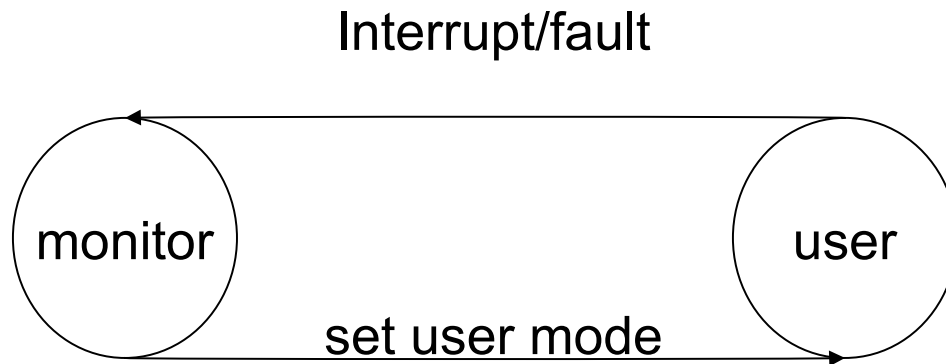
- 80386, 80486, Pentium Pro, Pentium II, III, IV
- 4 个特权级别 (特权环 , Privilege Levels)
 - R0, R1, R2, R3 数字越大级别越小

■ CPU 状态的转换

- 目态到管态的转换
 - 唯一的途径：中断，将 CPU 的状态位标志为管态 (trap 为软中断)
- 管态到目态的转换
 - 通过设置 PSW 指令 (修改程序状态字)

Dual-Mode Operation

- *Mode bit* added to computer hardware to indicate the current mode: monitor (0) or user (1)
- When an interrupt or fault occurs hardware switches to monitor mode



Privileged instructions can be issued only in monitor mode



内存寻址方法

- 逻辑地址
 - 段 (或选择子): 偏移量
- 线性地址
 - 32 位平坦地址空间
 - 若无分页机制, 则线性地址就等于物理地址
 - 若有分页机制, 则线性地址需经过 **MMU** 转换为物理地址
- 物理地址
 - 系统内存的真正地址



内存数据编码

- 内存中存放多于一个字节的数据 (int) 时，如何存放
 - 小端编码
 - 将地位的字节放在低地址，高位的字节放到高地址
 - **x86 系列**
 - 大端编码
 - 反之， **Ultra Sparc** 等芯片
- 网络通信时的数据编码？



存储系统

存储器的类型

- 读写型存储器 RAM(Random Access Memory)
- 只读型存储器 ROM(Read-Only Memory), PROM, EPROM

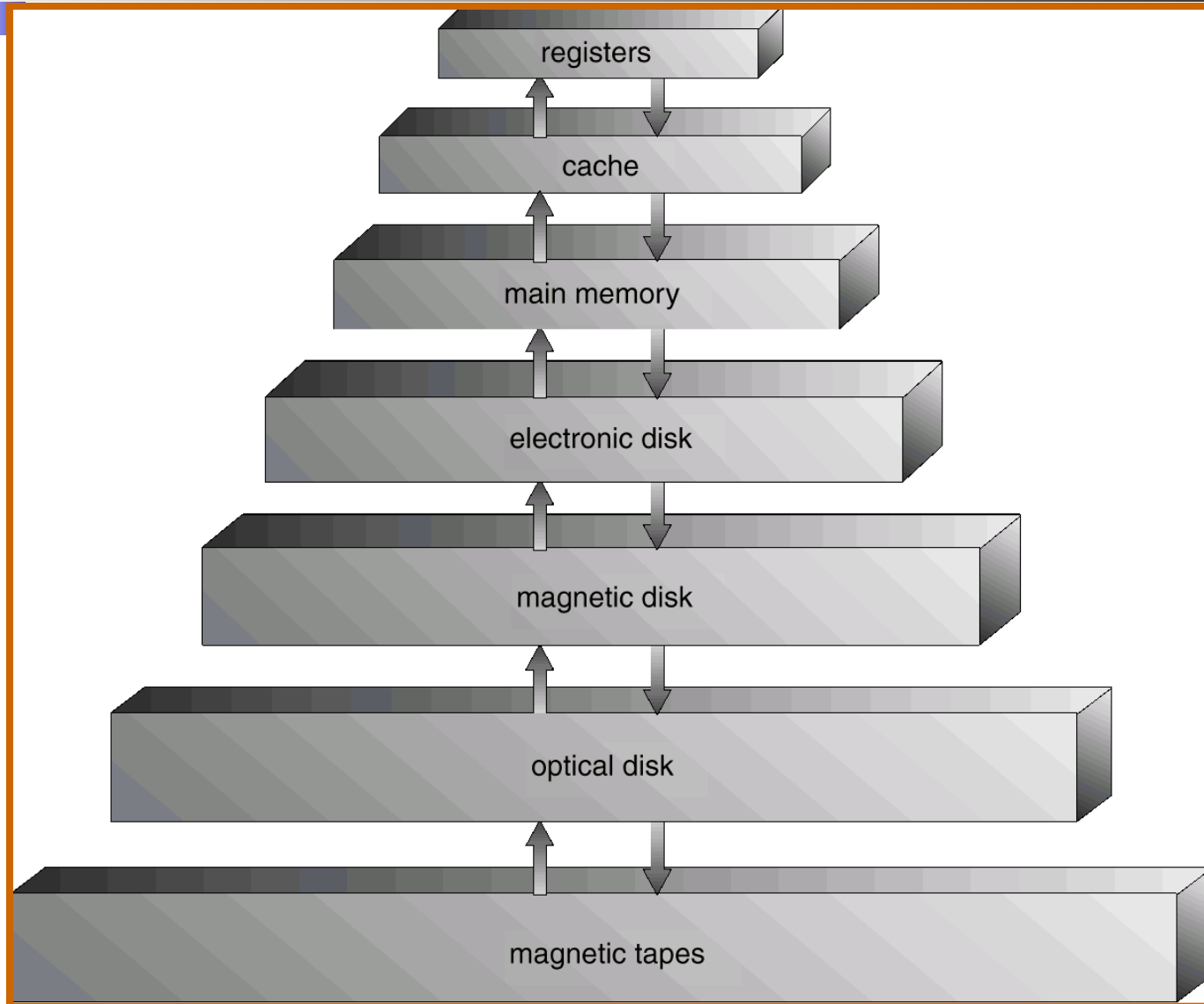
■ 存储分块

- 存储最小单位 Bit :0,1
- 存储器的最小编址单位 Byte:8bit
- 1KB=1024Byte, 1MB=1024KB, 1GB=1024MB
- 为用户分配最小主存单位：块 (Block), 或页 (Page): 大小不定
- Server:512M~4GB

■ 存储器的层次结构

- 存储系统的设计主要考虑三个问题：容量、速度、成本
- 存储体系：寄存器、高速缓存、主存储器、硬盘存储器、磁带机以及光盘存储器 *
- 提高存储系统效能的关键：程序的存储访问局部性原理

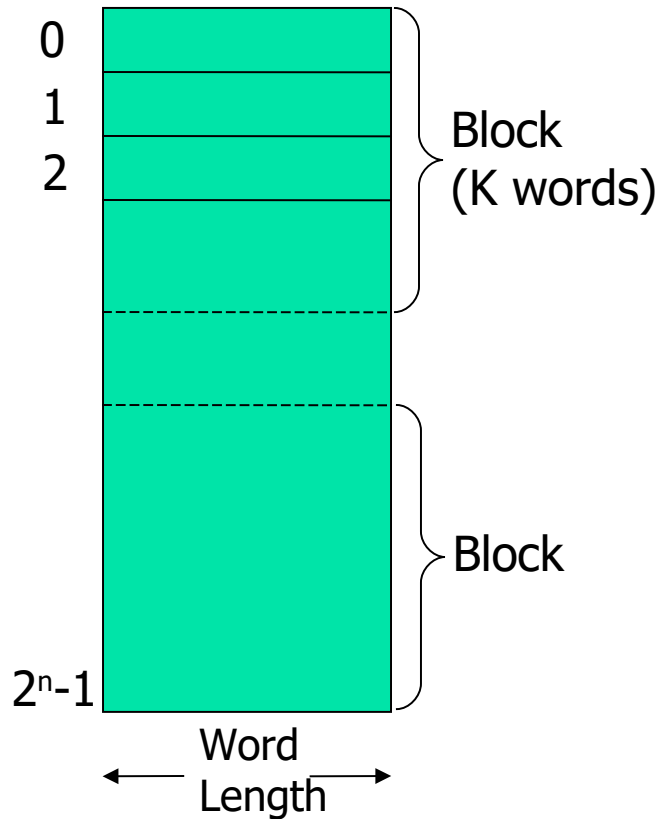
存储设备层次



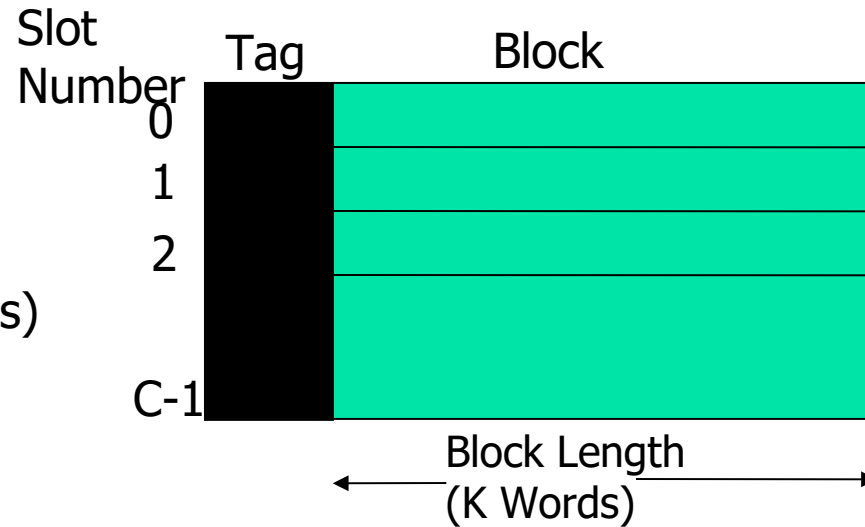
缓存 (cache) 与主存的结构

(a) Main Memory

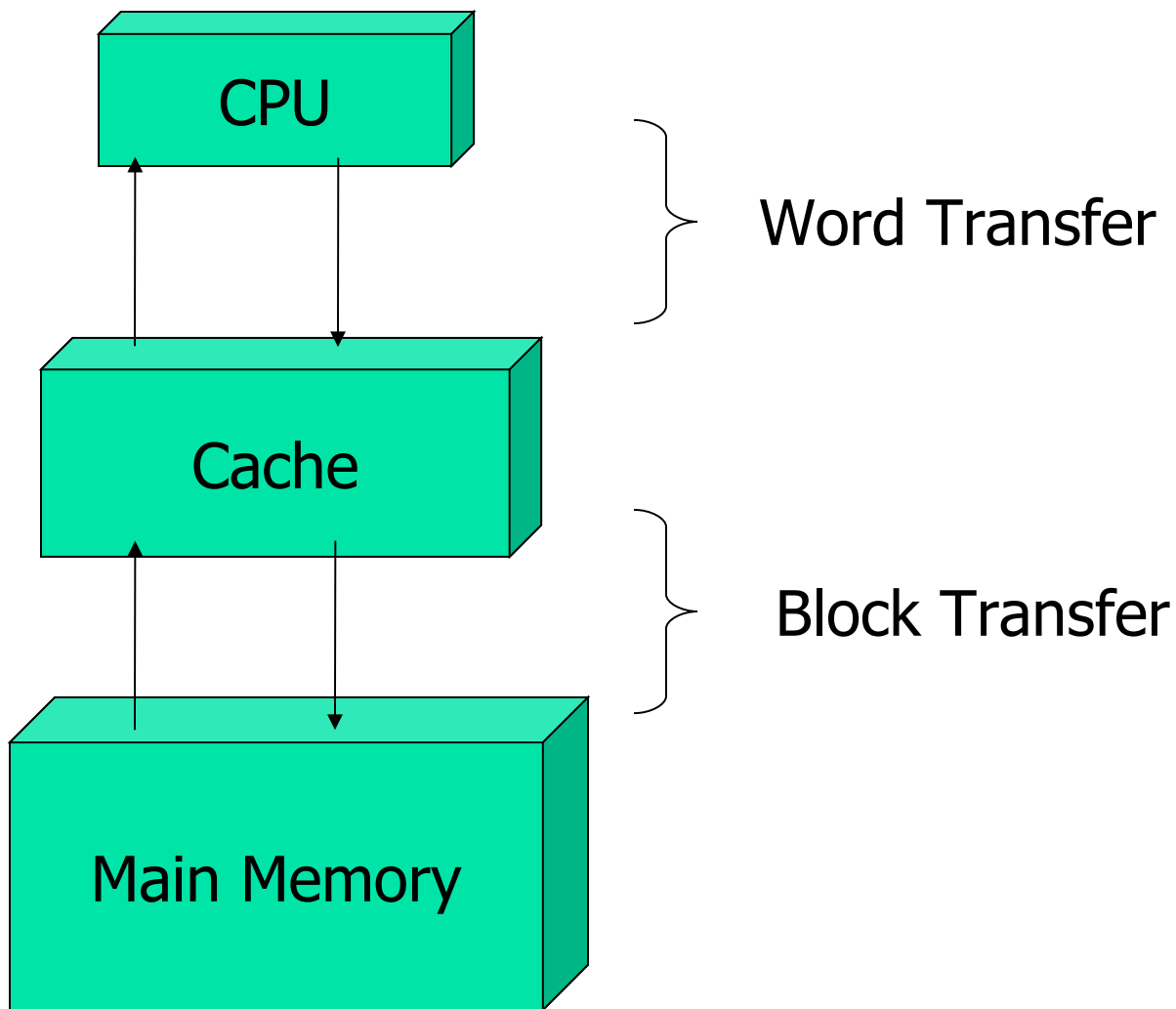
Memory address



(b) Cache



缓存 (cache) 与主存的数据交互





缓存 (cache) 与主存的地址映射

■ 缓存 (cache) 与主存的地址映射

- 直接映像、全相联映像、组相联映像
- 直接映像
- 主存与 Cache 的划分：将主存根据 Cache 的大小分成若干分区；Cache 也分成若干个相等的块，主存的每个分区也分成与 Cache 相等的块
- 主存与 Cache 的映像：主存中的每一个分区由于大小相同，可以与整个 Cache 相像，其中的每一块正好配对。编号不一致的块是不能相互映像的
- 优点：地址变换简单；缺点：每块相互对应，不够灵活。
- 主存与 Cache 的地址组成：主存：区号 + 块号 + 块内地址；Cache：块号 + 块内地址



缓存 (cache) 与主存的地址映射

■ 全相联映像

- 主存与 Cache 的划分：将主存与 Cache 划分成若干个大小相等的块
- 主存与 Cache 的映像：主存中每一块都可以调到 Cache 中的每一块
- 优点：访问灵活，冲突率低，只有 Cache 满时才会出现在冲突
- 缺点：地址变换比较复杂，速度相对慢
- 主存与 Cache 的地址组成：
- 主存：块号 + 块内地址
- Cache：块号 + 块内地址



缓存 (cache) 与主存的地址映射

■ 组相联映像

■ 主存与 Cache 的划分

- 主存：主存根据 **Cache** 大小划分成若干个区，每个区内划分成若干个组，每个组再划分成若干个块
- **Cache**：划分成若干个组，每个组划分成若干个块

■ 主存与 **Cache** 的映像

- 主存的每个分区之内的组与 **Cache** 之组采用直接映像，主存的每个组之内块与 **cache** 之块采用全相联映像
- 融合前两种映像方式，实现容易命中率与全相联映像接近
- 主存与 **Cache** 的地址组成
 - 主存：区号 + 组号 + 块号 + 块内地址
 - **Cache**：组号 + 块号 + 块内地址

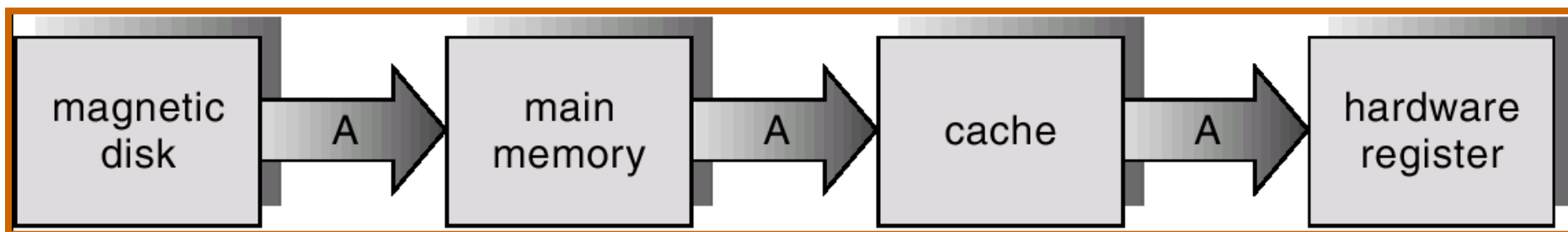


实例：缓存与主存的地址映射

- 容量为 64 块的 Cache 采用组相联方式映像，字块大小为 128 字节，每 4 块为一组，若主容量为 4096 块，且以字编址，则主存地址为多少位，主存区号为多少位？
 - 组相联的地址构成为：区号 + 组号 + 块号 + 块内地址
 - 主存的每个分区大小与整个 Cache 大小相等，故此主存需要分的区数为： $4096/64=64$ ，因为 $2^6 = 64$ ，因此需要 6 位来表示区号
 - 每 4 块为一组，故共有组数 $64/4 = 16$ ，因为 $2^4 = 16$ ，因此需要 4 位表示组号每组 4 块，故表示块号需要 2 位
 - 块内地址共 128 字节， $2^7 = 128$ ，故块内地需要 7 位表示
 - 所以：主存地址的位数 = $6+4+2+7 = 19$
 - 主存区号的位数 = 6



Move “A” From Disk to Register



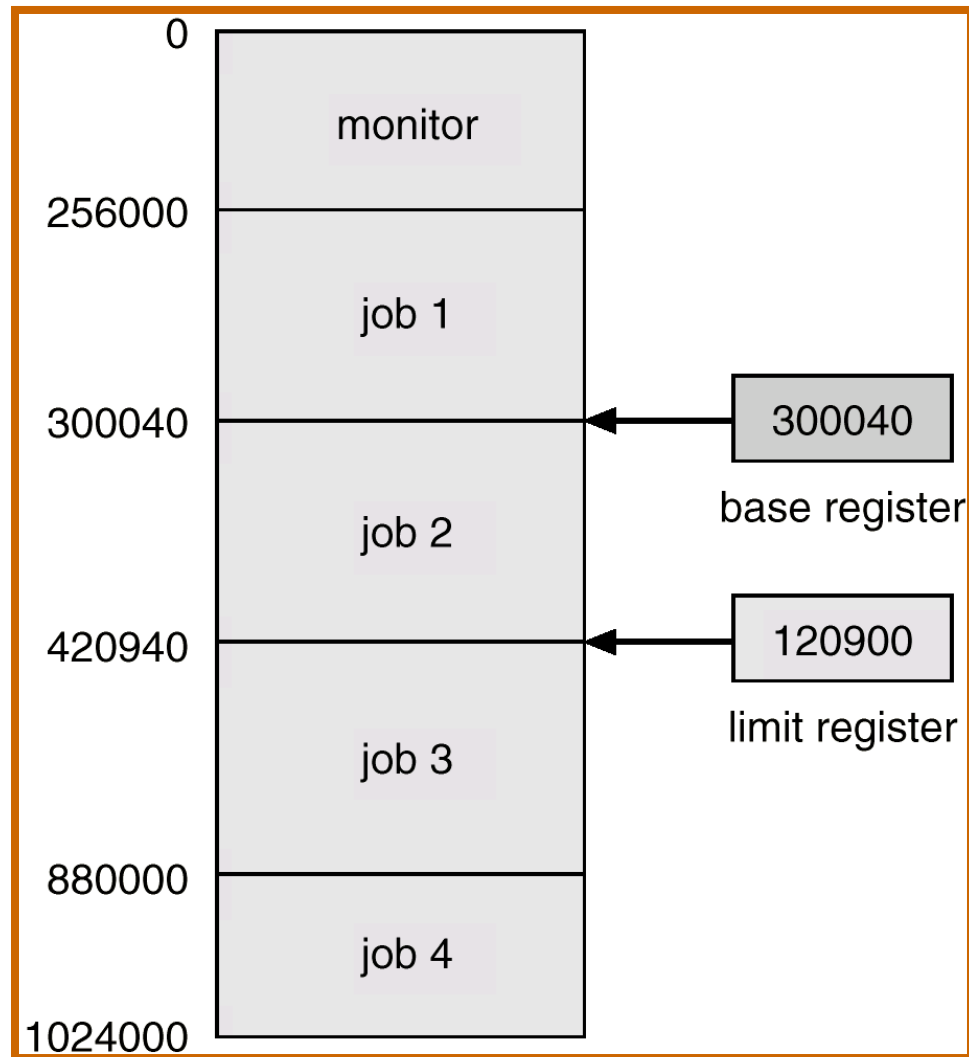


Memory Protection

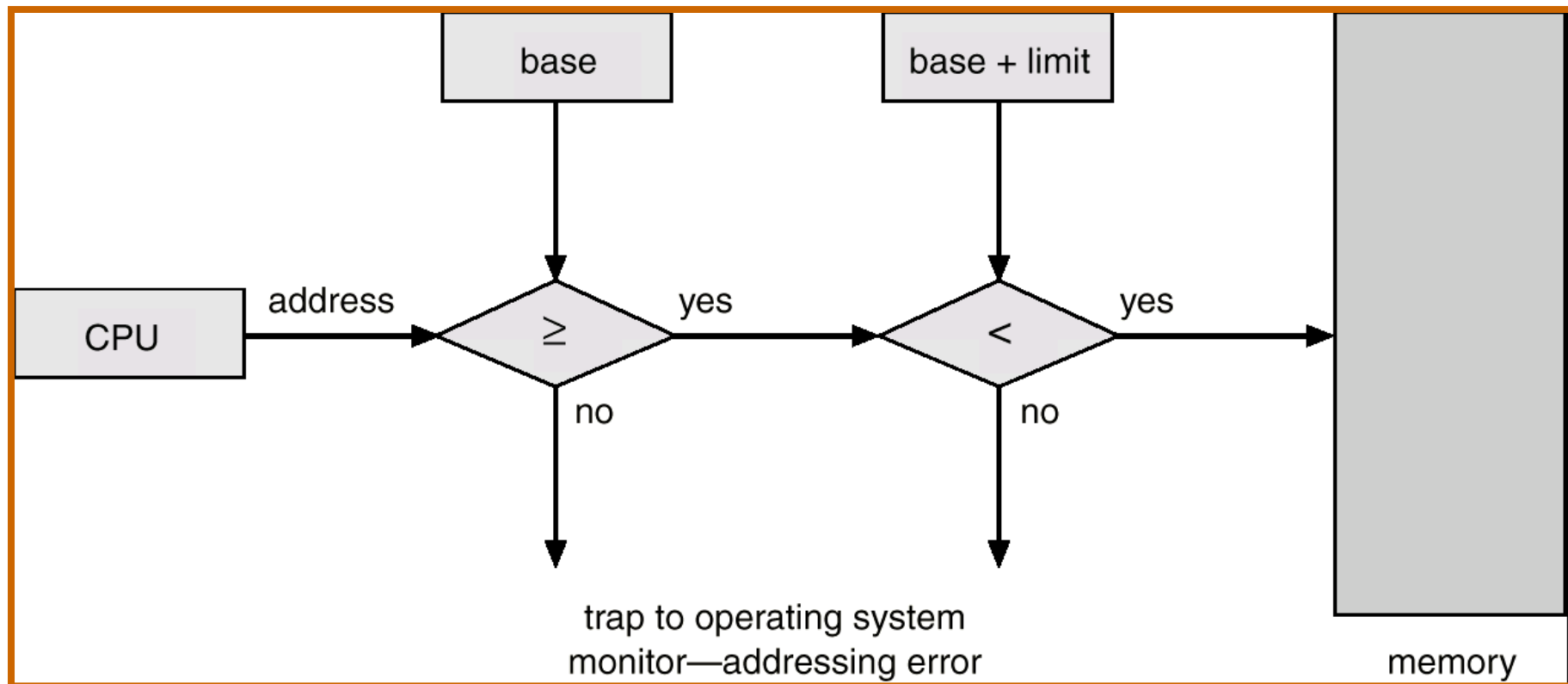
■ 存储保护

- 界地址寄存器（界限寄存器）：上下限寄存器
- 存储键：每个主存的存储块都有一个存储保护键（不同的进程其键值不同）
- Must provide memory protection at least for the interrupt vector and the interrupt service routines
- In order to have memory protection, at a minimum add two registers that determine the range of legal addresses a program may access:
 - **Base register** – holds the smallest legal physical memory address
 - **Limit register** – contains the size of the range
- Memory outside the defined range is protected

Use of A Base and Limit Register



Hardware Address Protection

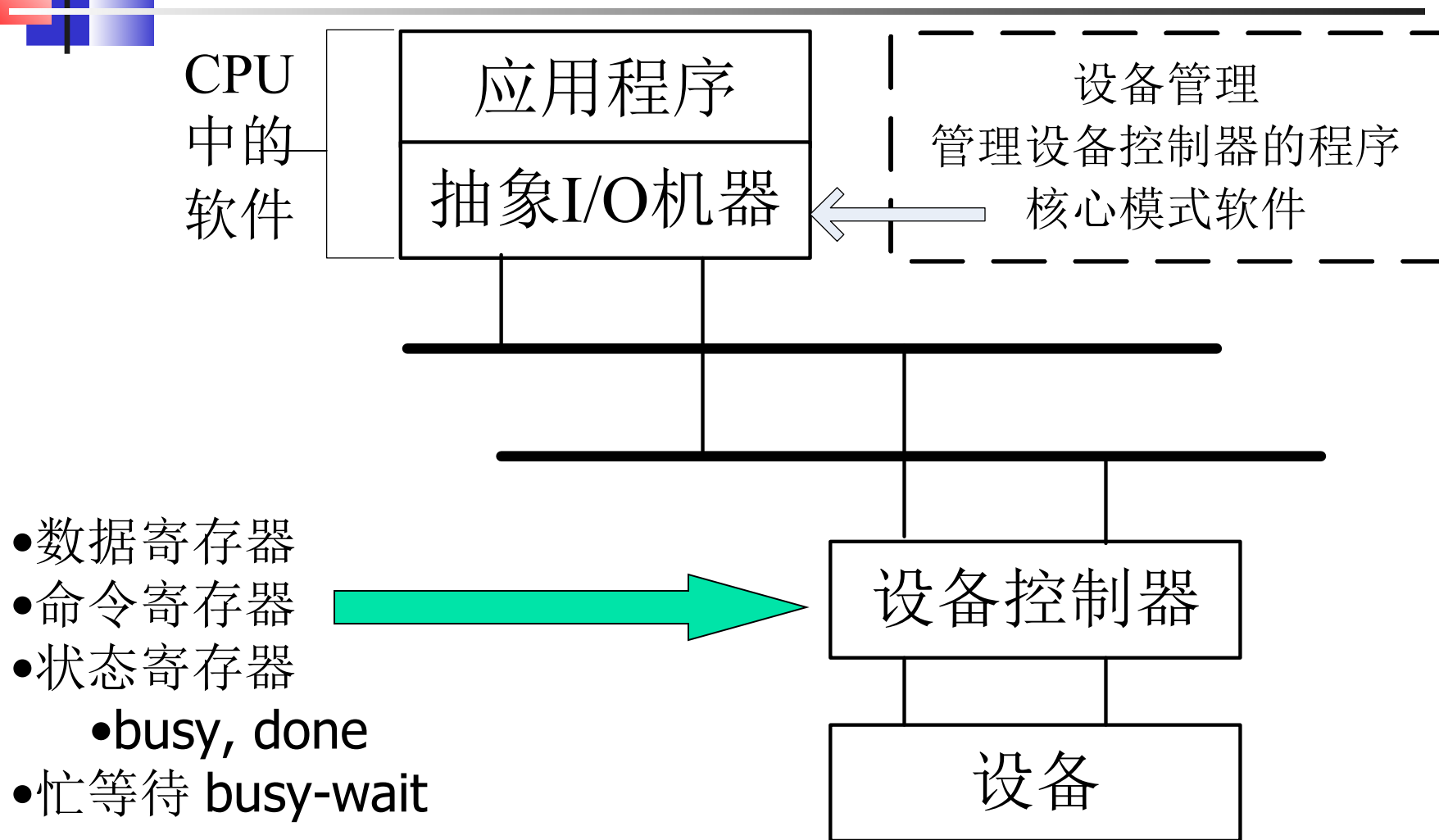




Hardware Protection

- When executing in monitor mode, the operating system has unrestricted access to both monitor and user's memory
- The load instructions for the *base* and *limit* registers are privileged instructions

I/O 设备



设备、控制器、软件之间的关系



I/O 设备接口

■ 外部设备接口

- 是外部设备与主机 (CPU 和存储器) 的通信是通过外设接口完成的
- 三类寄存器
 - 数据寄存器
 - 用来存放要在外设和主机间传送的数据
 - 缓冲作用
 - 状态寄存器
 - 用来保存外部设备或接口的状态信息
 - 命令寄存器
 - CPU 给外设或接口的控制命令通过此寄存器送给外部设备



I/O 访问技术

- 外设的各种技术

- 轮询（忙等待）

- I/O 结构：完全由 CPU 直接发出一条条指令来实现。（效率低）

- 通道（中断）

- 通道独立于 CPU, 专门负责数据 I/O 传输工作的处理单元。通道对外部设备实行统一的管理, 它代替 CPU 对 I/O 操作进行控制, 从而使 CPU 和外部设备可以并行工作, 故又称为 I/O 处理机。
- 由 CPU 发布启动 I/O 命令, 然后由通道控制, 外部设备工作结束后产生一个“输入输出操作结束”的 I/O 中断事件, 然后由 CPU 处理此中断
- 通道技术一般用于大型机系统和 I/O 处理能力要求高的系统中

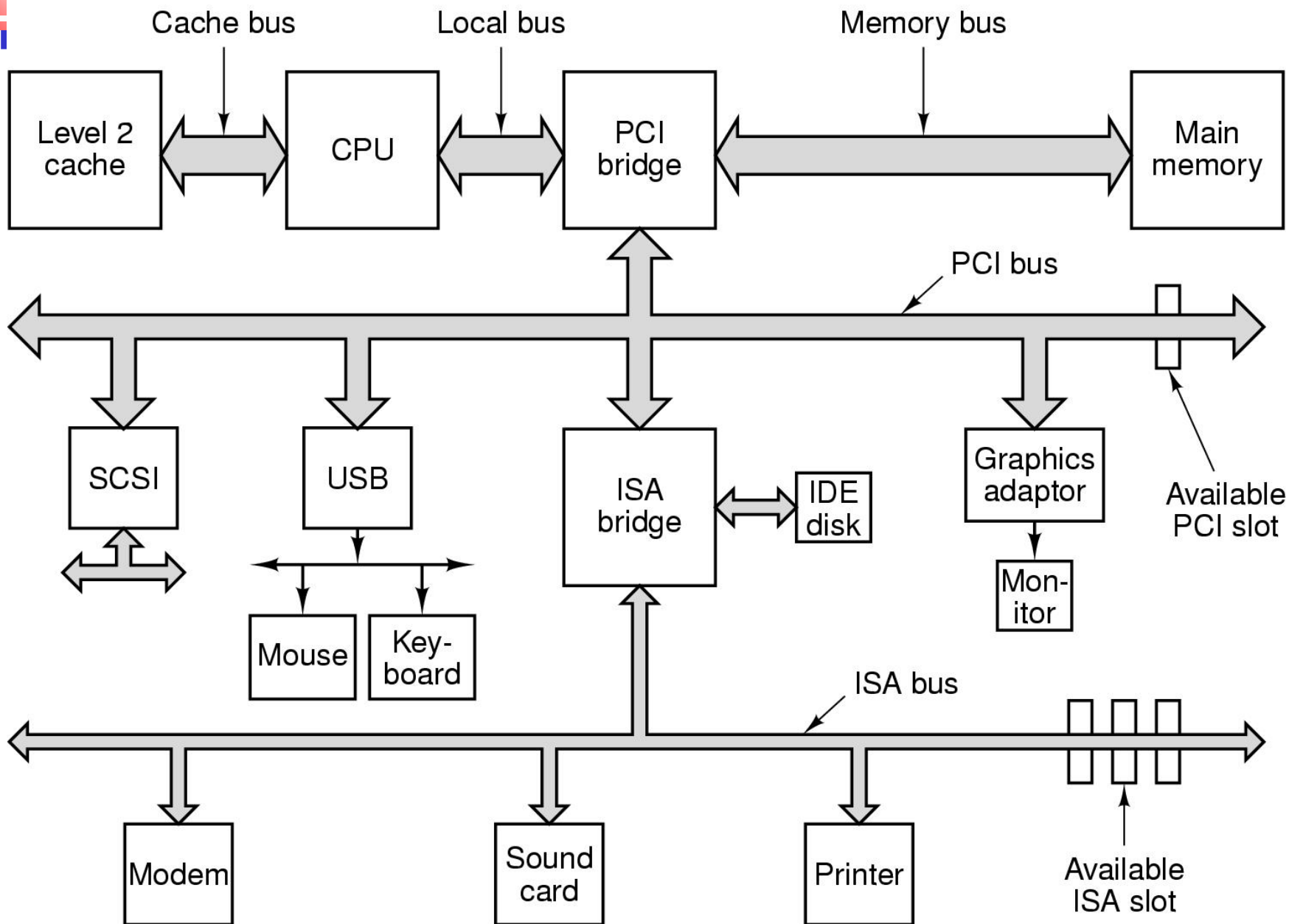
- DMA 技术 (Direct Memory Access) (中断)

- 直接存储器访问技术通过系统总线中的一个独立控制单元 DMA 控制器, 自动地控制成块数据在内存和 I/O 单元之间的传送。（大量数据）

- 缓冲技术（本质上是解决 CPU 处理数据速度与外设不匹配）

- 缓冲技术是用于外部设备与其它硬件部件之间的一种数据暂存技术, 利用存储器件在外部设备中设置了数据的一个存储区域, 称为缓冲区。

总线 BUS: 实例

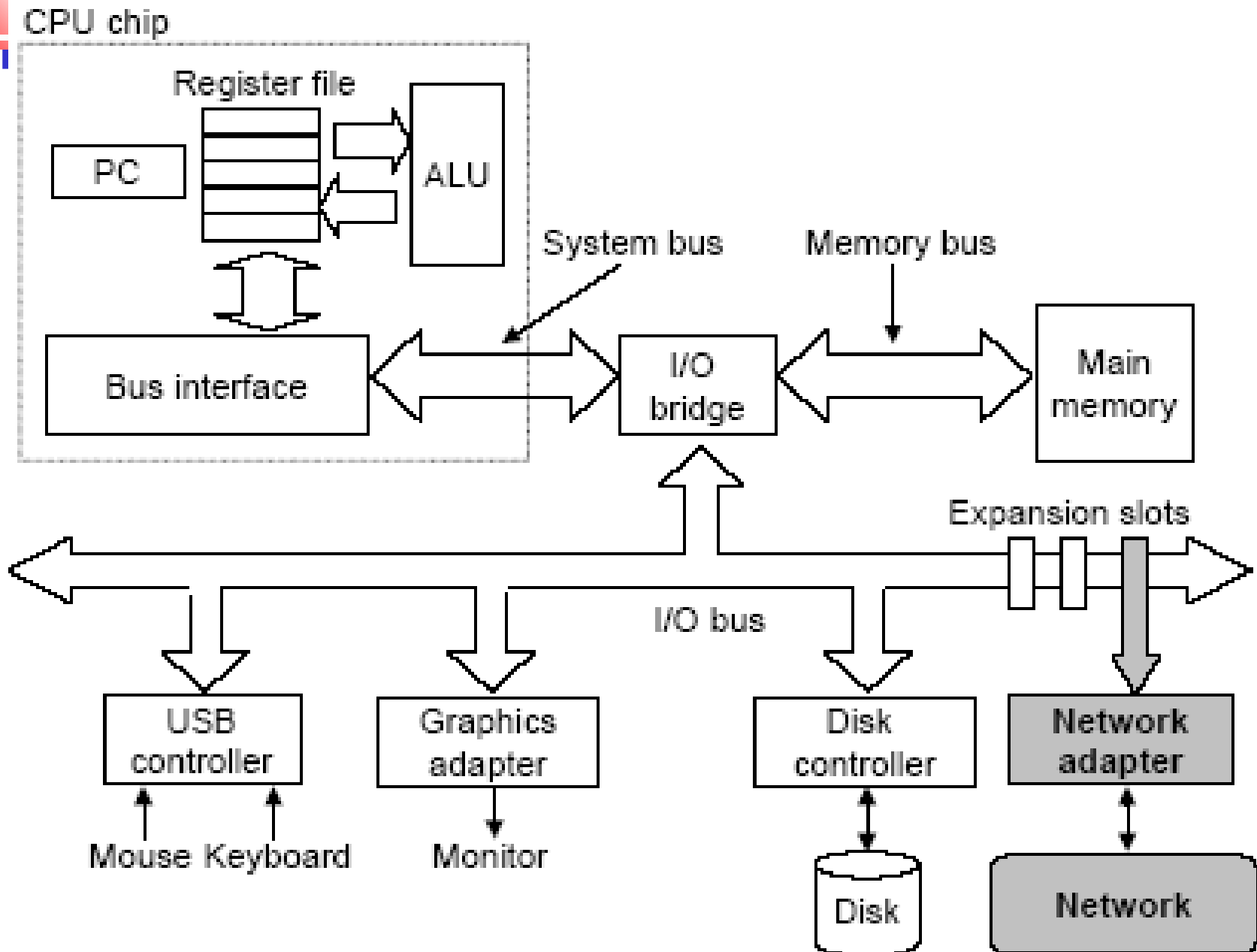




总线 bus

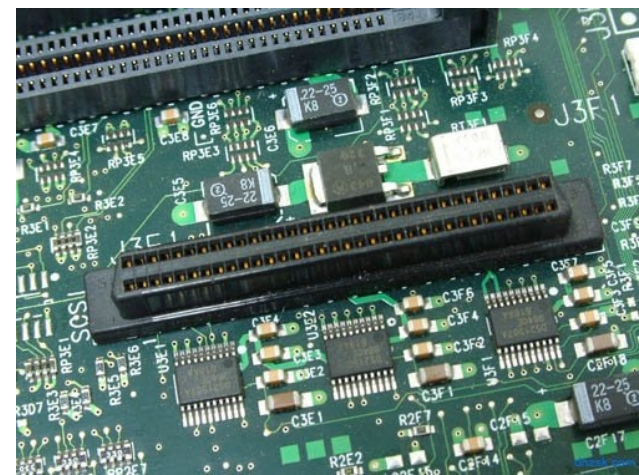
- cache bus
- local bus
- memory bus
- PCI:Peripheral Component Interconnect
 - Intel, 66MHz, 528Mb/s, 并行传 8Bytes
- SCSI:Small Computer System Interface
- USB:Universal Serial Bus
- IDE
- ISA:Industry Standard Architecture
 - IBM PC/AT 总线, 8.33MHz, 16.67Mb/s, 并行传 2Bytes

Pentium 机系统结构



SCSI

- Small Computer System Interface
- 小型计算机系统接口，是种较为特殊的接口总线，具备与多种类型的外设进行通信。
- SCSI 采用 **ASPI**（高级 SCSI 编程接口）的标准软件接口使驱动器和计算机内部安装的 SCSI 适配器进行通信。
- SCSI 接口是一种广泛应用于小型机上的高速数据传输技术。
- SCSI 接口具有应用范围广、多任务、带宽大、CPU 占用率低，以及热插拔等优点。
- 其缺点就在于昂贵的价格。





时钟 clock

■ clock

- 又称定时器 **timer**，负责维护时间，支持进程调度等，在计算机系统中十分必要。
- 硬件时钟、软件时钟

■ 硬件时钟 * 的原理 8284/82C284

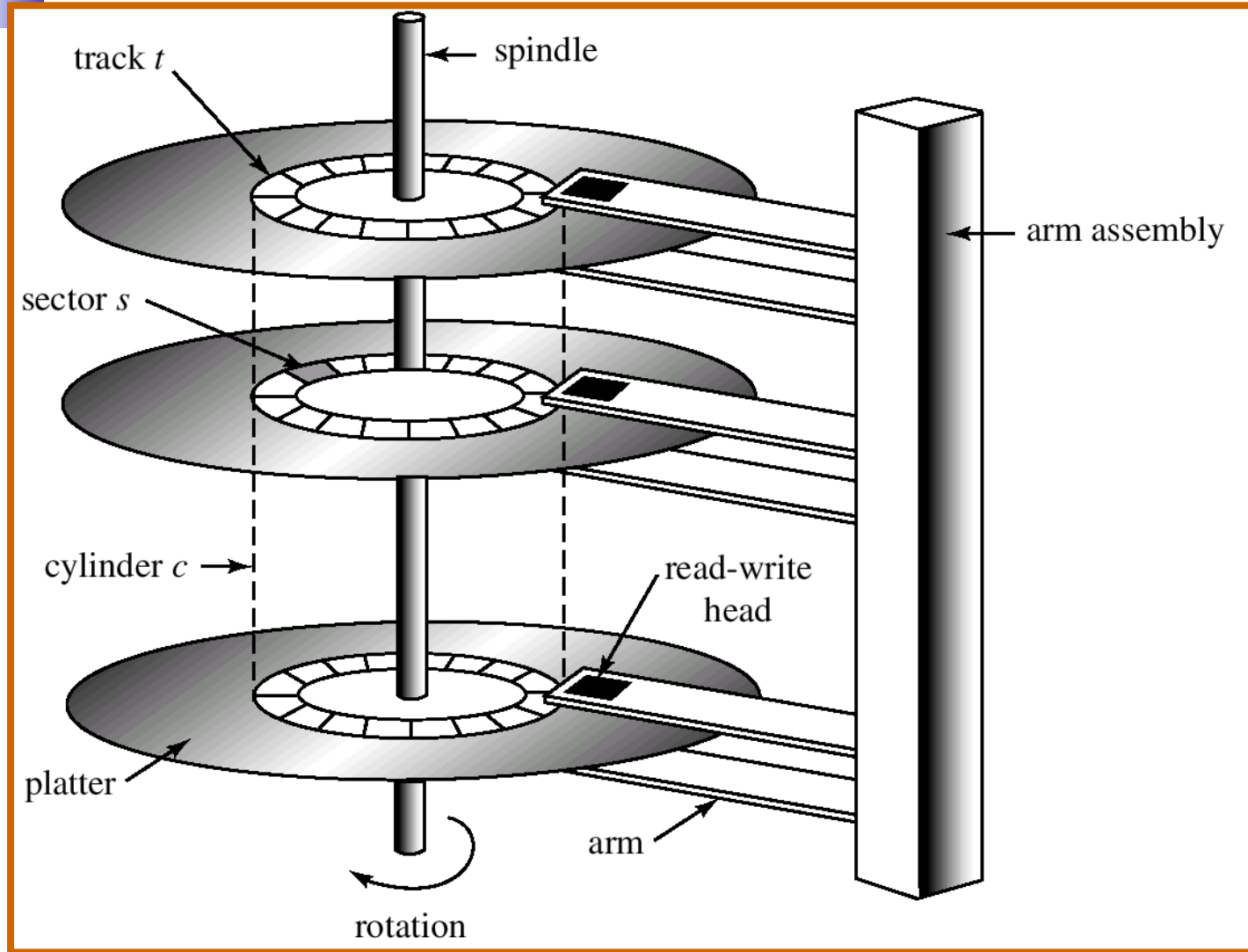
- 硬件时钟的全部工作是根据已知的时间间隔产生中断
- 1 晶体振荡器：每隔一定间隔产生固定的脉冲频率
- 2 计数器：晶体振荡器的每一次脉冲都将计数器减 1，当计数器变为 0 时，产生一个 **CPU** 中断
- 3 存储寄存器
- 操作模式
 - 1) 一次完成模式 **one-shot mode**: 计数器被减至 0 则停止工作
 - 2) 方波模式 **square-wave mode**: 计数器被减至 0 且产生中断后，存储寄存器的值自动复制到计数器中，无限期重复
- 时钟滴答 **clock tick**: 方波模式下产生
- 其中断频率可以由软件控制：可编程时钟



通过时钟 CPU 完成的工作

- 在多道程序运行的环境中，时钟可以为系统发现一个陷入死循环的作业，从而防止机时的浪费；
- 在分时系统中，用时钟间隔来实现各个作业按时间片轮转运行；
- 在实时系统中，按要求的时间间隔输出正确的时间信号给相关的实时控制设备；
- 定时唤醒那些要求按照事先给定的时间执行的各个外部事件，如定时器 *
- 记录用户使用各种设备的时间和记录某外部事件发生的时间间隔
- 记录用户和系统所需要的绝对的时间，年月日

硬盘结构





硬盘操作

- 硬盘能被操作系统使用前的三个操作
 - 低级格式化
 - 扇区的物理地址：磁头号 H、柱面号 C、扇区号 S
 - 使用 **FDISK** 命令进行分区 (PartitionMagic)
 - Primary DOS Partition (Can be Actived)
 - Extended DOS Partition
 - Non-DOS Partition
 - 使用 **FORMAT** 命令进行高级格式化
 - Format d: [/S] [/V:label]
- 硬盘的结构
 - 物理第一扇区：0 柱 0 头 1 扇
 - 主引导程序 bootstrap(446B)*
 - 硬盘的各个分区 (1BE~1FDh, 64B=16X4)*
 - (0x1fe0x1ff):0xaa55 有效标志
 - 驱动器参数块 DPB(Disk Parameter Block)

硬盘主分区表

	00h	01h	02h	03h	04h	05h	06h	07h	08h~0Bh	0Ch~0Fh
01BEh	BI	Hs	Ss	Cs	SI	He	Se	Ce	HS	N
01CEh	BI	Hs	Ss	Cs	SI	He	Se	Ce	HS	N
01DEh	BI	Hs	Ss	Cs	SI	He	Se	Ce	HS	N
01EEh	BI	Hs	Ss	Cs	SI	He	Se	Ce	HS	N

BI 引导标志 80h 激活分区,00h 未激活

Hs 起始磁头号,Ss 起始扇区号,Cs 起始柱面号

SI 系统标志: 00h 未定义,01hDos 分区 (12 位),02hXENIX 系统,
04hDos 分区 (16),05h 扩展分区,06Dos 分区 (32)

He,Se,Ce

HS 隐藏扇区数,本分区前的扇区数

N 本分区占的扇区数



DOS 硬盘分区信息

HS		RS	NF*FS	32*DS/ SS	(最大簇号 -1)*AU
物理 第 1 扇区	...	DOS 引导 扇区	FAT 区	根目录区	文件区
本盘卷之前空间			1 个 DOS 盘卷		

RS: 保留扇区数

FAT: 文件分配表

NF:FAT 数目 (2)

FS: 每个 FAT 的扇区数

DS: 根目录项数

SS: 每扇区字节数

AU: 每簇扇区数



1.44M 软盘组织结构

逻辑扇区	内容
0	引导扇区
1	(第一个) FAT 表的第一个扇区
...	
10	(第二个) FAT 表的第一个扇区 (如果有的话, 查看引导扇区的地址 0x10 处)
19	磁盘根目录的第一个扇区
xx	根目录的最后一个扇区 (查看引导扇区的地址 0x11 处)
xx+1	磁盘数据区的开始处



1.44M 软盘 DOS 引导扇区

0x00	0x02	< 跳转到地址 0x1e 处的指令 >
0x03	0x0a	计算机制造厂商名称
0x0b	0x0c	每扇区的字节
0x0d	0x0d	每柱面的扇区
0x0e	0x0f	引导记录的预留扇区
0x10	0x10	FAT 表数
0x11	0x12	根目录表项的数目
0x13	0x14	逻辑扇区数目
0x15	0x15	介质描述符字节 (只用于老版本的 MS-DOS)
0x16	0x17	每个 FAT 表的扇区
0x18	0x19	每个磁道的扇区
0x1a	0x1b	磁头数
0x1c	0x1d	隐藏扇区数
0x1e	...	引导程序



活动分区或磁盘的引导程序

Bootstrap 引导程序

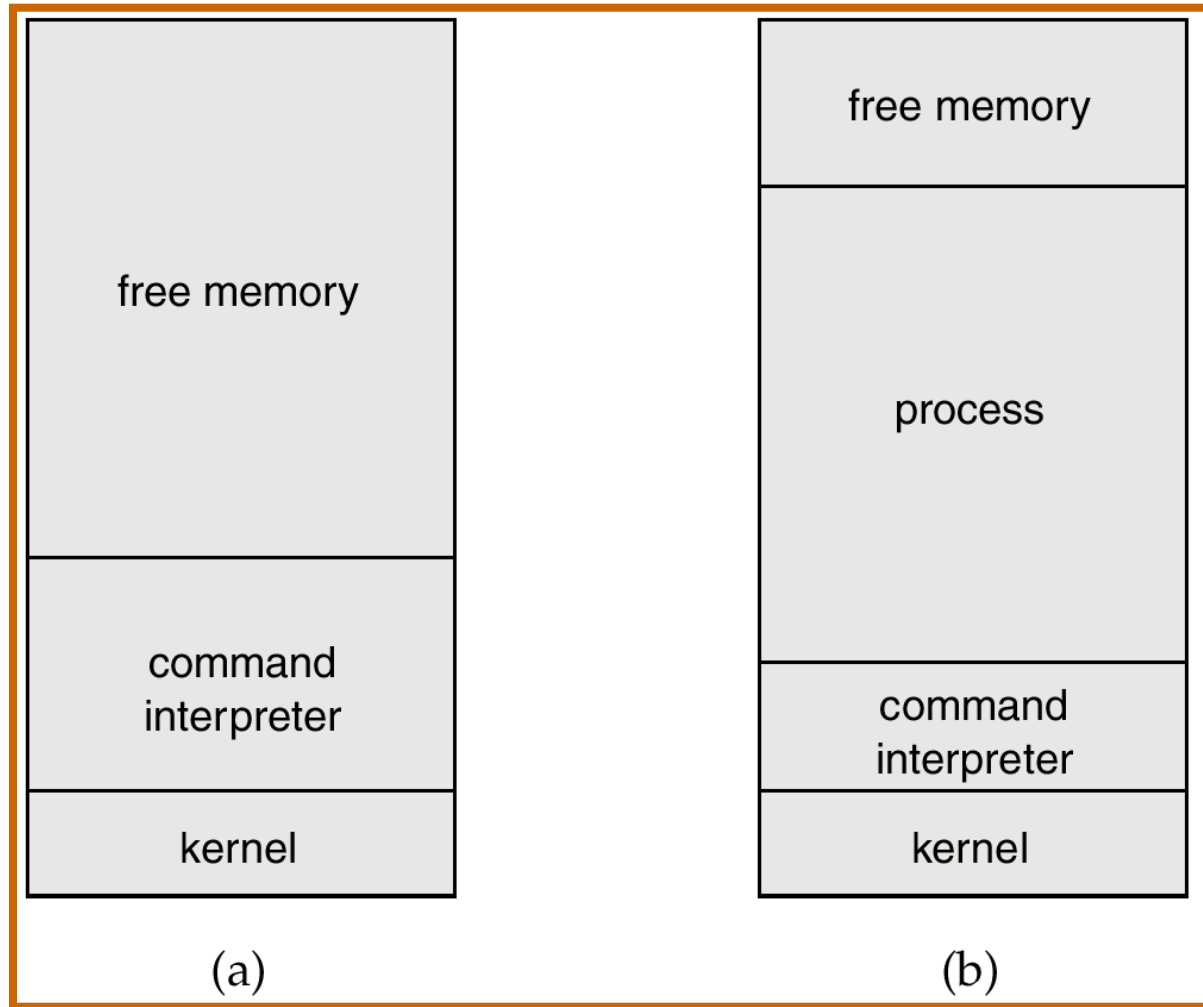
0x000~0x002	<A jump instruction to 0x <i>ttt</i> >
0x003~...	Disk parameters(used by BIOS)
0x <i>ttt</i> ~0x1fd	Bootstrap program
0x1ff~0x1fe	0xaa55



计算机的引导过程

- 1 硬件系统加电
- 2ROM BIOS 自举程序
 - 2.1 硬件自检 POST
 - 2.2 若软盘启动则执行 2.4, 若硬盘启动则执行 2.5
 - 2.3 提示无启动盘
 - 2.4 将软盘物理第 1 扇区读入内存 07C0:0000 处, 并将控制权转移到内存 07C0:0000, 从内存 07C0:0000 开始执行 4
 - 2.5 将硬盘物理第 1 扇区读入内存 07C0:0000 处, 并将控制权转移到内存 07C0:0000, 从内存 07C0:0000 开始执行 3
- 3 执行硬盘主引导程序, 寻找到激活分区, 将该分区首扇区读入 07C0:0000, 原硬盘主引导程序相应下移, 从内存 07C0:0000 开始执行 4
- 4 执行引导程序 (bootstrap), 装载相应盘上的 OS 内核并执行内核初始化程序等指令, 从而将 OS 引导成功。

MS-DOS Execution



At System Start-up

Running a Program

Metric Units: 1000 ? 1024?

Exp.	Explicit	Prefix	Exp.	Explicit	Prefix
10^{-3}	0.001	milli	10^3	1,000	Kilo
10^{-6}	0.000001	micro	10^6	1,000,000	Mega
10^{-9}	0.000000001	nano	10^9	1,000,000,000	Giga
10^{-12}	0.000000000001	pico	10^{12}	1,000,000,000,000	Tera
10^{-15}	0.000000000000001	femto	10^{15}	1,000,000,000,000,000	Peta
10^{-18}	0.000000000000000001	atto	10^{18}	1,000,000,000,000,000,000	Exa
10^{-21}	0.000000000000000000001	zepto	10^{21}	1,000,000,000,000,000,000,000	Zetta
10^{-24}	0.000000000000000000000001	yocto	10^{24}	1,000,000,000,000,000,000,000,000	Yotta

The metric prefixes



Metric Units- 中文表达

字节 B (byte)

千字节 KB (kilobyte)

兆字节 MB (megabyte)

吉字节 GB (gigabyte)

太字节 TB (terabyte)

拍字节 PB (petabyte)

艾字节 EB (exabyte)

泽字节 ZB (zerabyte)

尧字节 YB (yattabyte)

BB (Brontobyte)

NB (Nonabyte)

DB (Doggabyte)

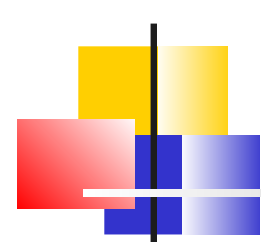


Summary

- 计算机系统组成
- 中央处理器 (CPU)
- 存储系统
- I/O 技术
- 中断机制
- 总线
- 时钟
- 操作系统的启动



Q&A



A word cloud featuring the word "thank you" in various languages and scripts. The central word "thank you" is the largest and most prominent, rendered in red. Surrounding it are numerous other expressions of gratitude in different colors and sizes, including:

- danke** (German)
- 謝謝** (Chinese)
- ngiyabonga** (Xhosa)
- tesekkür ederim** (Turkish)
- gracias** (Spanish)
- tapadh leat** (Irish Gaelic)
- obrigado** (Portuguese)
- dziękuję** (Polish)
- sukriya** (Hindi)
- kop khun krap** (Lao)
- arigatō** (Japanese)
- merci** (French)
- terima kasih** (Indonesian)
- 감사합니다** (Korean)
- xiexie** (Mandarin)
- euχαριστώ** (Greek)
- shukriya** (Urdu)
- merce** (Catalan)
- merci** (Italian)
- trugarez** (Breton)
- chokrone** (Sinhala)
- manana** (Swahili)
- asante** (Kisumu)
- hvala** (Slovene)
- obrigada** (Portuguese)
- tenki** (Japanese)
- mamun** (Arabic)
- moichchakkeram** (Tamil)
- go raibh maith agat** (Irish Gaelic)
- taiku** (Lao)
- grazie** (Italian)
- tanemirt** (Finnish)
- rahmet** (Turkmen)
- najis tuke** (Pashto)
- kam sah hamida** (Pashto)
- did madloba** (Pashto)
- mes** (Pashto)
- dekui** (Pashto)
- sobodi** (Pashto)
- hvala** (Slovene)
- gracie** (Polish)
- hvala** (Slovene)
- mauriuru** (Maori)
- koszonom** (Hungarian)
- hvala** (Slovene)
- dhanyavad** (Sinhala)
- kiitos** (Finnish)
- nandri** (Sinhala)
- nanni** (Sinhala)
- enkosi** (Zulu)
- bedankt** (Dutch)
- spas** (Sinhala)
- tack** (Swedish)
- welalin** (Sinhala)
- barka** (Sinhala)
- kia ora** (Maori)
- blagodaram** (Sinhala)
- vinaka** (Sinhala)
- спасиби** (Sinhala)
- спасибо** (Russian)
- Баярлалаа** (Mongolian)
- рахмат** (Tajik)