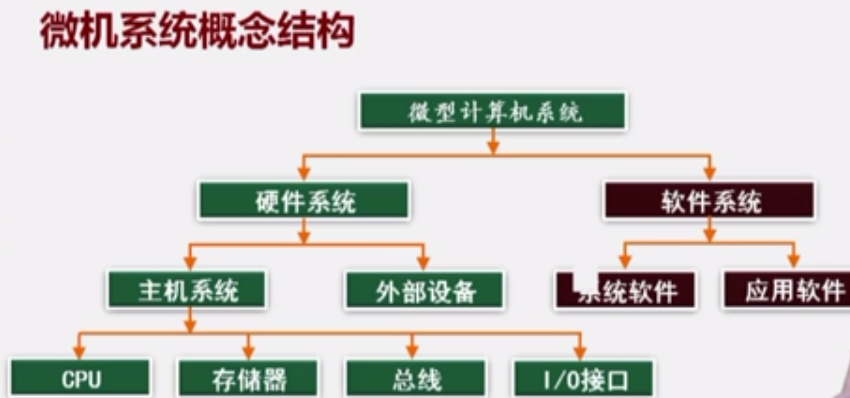
微机原理与接口技术

计算机的主要应用：数值计算、信息处理、**过程控制（本课程研究方向）**。

课程内容：数值信息表示、微型机基本原理、汇编程序设计、半导体存储器及其接口设计、输入输出技术（数字并行接口、模拟量并行接口）

### 一、微型计算机基础概念



1. 计算机系统的组成：硬件系统、软件系统

硬件系统：主机系统、外部设备（不能直接和CPU进行数据交换，需要通过输入输出接口与计算机进行数据交换）

主机系统：CPU、存储器、输入输出系统、总线

a. CPU：运算器、控制器、寄存器组

b. 存储器：内存储器、外存储器（联机外存：硬磁盘、固盘；脱机外存：移动存储设备）【机械硬盘接口：SATA】

内存条上芯片有8片。（内存由内存单元组成，每个单元存放1Byte数据）

内存容量：内存单元的个数。

内存分为：随机存取存储器（RAM）、只读存储器（ROM）

c. 输入/输出（I/O）接口：CPU与外部设备间的桥梁。

功能：数据缓冲寄存、信号电平或类型的转换、实现主机和外设间的运行匹配

d. 总线：一组导线和相关的控制、驱动电路的集合。

地址总线（AB）、数据总线（DB）、控制总线（CB）

【能够与CPU直接进行信息交换的部件属于主机系统，不能够与CPU直接进行信息交换的部件属于外部设备。】

1. 微型计算机的一般工作工程

**取指令** 🡪 **分析指令** 🡪 读取操作数 🡪 **执行指令** 🡪 存放结果

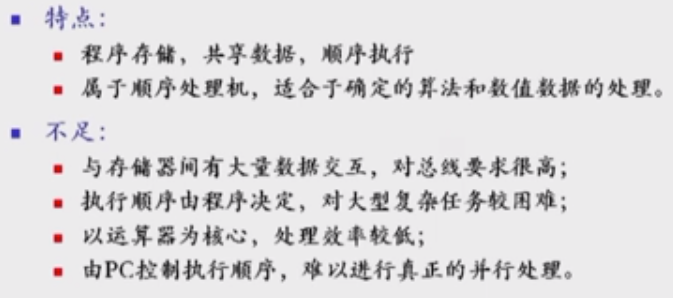
顺序执行、并行执行

加速比S = 顺序执行花费的时间/并行执行花费的时间

1. 冯诺依曼计算机

五个部件：存储器、运算器、控制器、输入设备、输出设备

存储程序工作原理，以运算器为核心。（如今，**大部分以存储器为核心**）



哈佛结构：

指令和数据分别存放在两个独立的存储器模块中。

CPU与存储器间指令和数据传送分别采用两组独立的总线。

1. ASCII码：128个字符（7bit，1Byte中最高位默认为0）

扩展ASCII码：8位（保括一些希腊字母和德语字母）

1. 数值的表示

浮点数，规格化：规格化过程中小数点后第一位必须为1。

规格化结果：正数补码表示为0.1xxx（包括1/2：0.100）；负数补码表示1.0xxx（除-1/2：1.100）。

例：浮点数规范化后，阶码和尾数都用补码表示，都是8位。求：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 阶码 | 尾数 | 值(阶码 尾数) |
| 最大正数 | 最大正数 | 最大正数 |  |
| 最小正数 | 最小负数 | 最小正数 |  |
| 最大负数 | 最小负数 | 最大负数 |  |
| 最小负数 | 最大正数 | 最小负数 |  |

【注：最后两项颇有争议。有人认为最大负数为1.0111111，最小负数为1.0000000】

1. 无符号数加减运算溢出判断：最高位向更高位有进位（或借位）时产生溢出。

有符号数加减运算溢出判断：最高位进位状态⊕次高位进位状态 = 1时产生溢出。

【】

### 二、微处理器与总线

1. 8088/8086（第三代处理器）：8088是inter公司对之前机器的兼容版本，8088和8086内部宽度都是16位，8086对外总线也是16位，而8088对外总线是8位。

【8088/8086地址总线20位，8088数据总线8位，8086数据总线16位】

1. 8088/8086 CPU特点：
2. 采用并行流水线工作方式

CPU内部通过设置指令预取队列实现。

1. 对内存空间实行分段管理（实地址模式下的寻址）

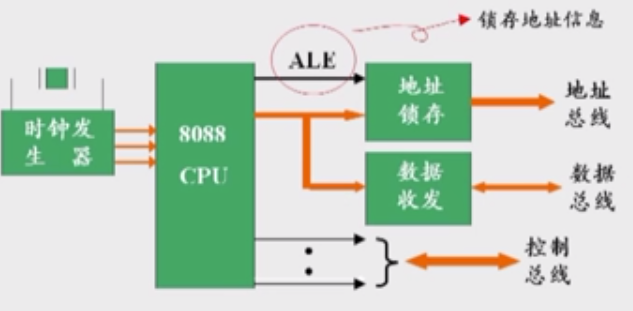
将内存分为4个段并设置地址段寄存器，以实现对1MB空间的寻址。

1. 支持协处理器

主要用来实现浮点数的计算。

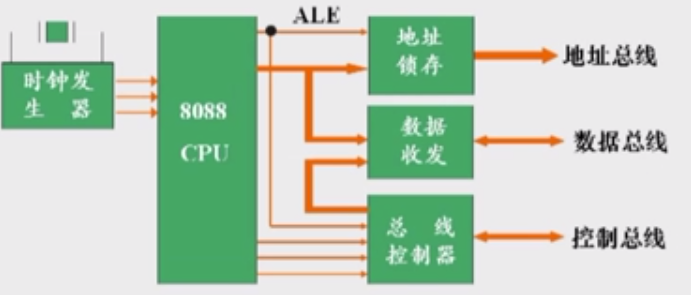
1. 8088/8086 CPU的两种工作模式
2. 最小模式

最小模式为单处理器模式，所有控制信号由微处理器产生。



1. 最大模式

最大模式为多处理器模式，部分控制信号由外部总线控制器产生【用于包含协处理器的情况下】



8088的工作模式是由引线的状态决定。为0工作在最大模式，为1工作在最小模式。

1. 8088的主要引线及内部结构

8088是一个40个引脚的一个双列直插式芯片。

* 8088最小模式下的主要引脚信息

1. 完成一次访问内存或接口所需要的主要信号

地址线和数据线：

* AD0-AD7：低8位地址和低8位数据信号分时复用。在传送地址信号时为单向，传送数据信号时为双向。
* A8-A15：8位地址信号。
* A16-A19：高4位地址信号，与状态信号分时复用。

主要控制信号：【#代表低电平有效】

* #WR：写信号
* #RD：读信号
* IO/#M：为“0”表示访问内存，为“1”表示访问接口
* #DEN：低电平有效，允许进行读/写操作（数据收发器）
* DT/#R：数据收发器的传送方向控制（数据收发器）
* ALE：地址锁存信号
* RESET：复位信号

1. 与外部同步控制信号

READY：外部同步控制信号，CPU访问一次内存或接口时，在第三个时钟周期后检测该信号，若为高电平表示访问正常，否则表示未就绪，需要在第三个周期后插入一个Twait周期，重复以上检测。（正常情况下，完成一次访问需要4个时钟周期）【一个总线周期等于4个时钟周期，每个时钟周期大致】



1. 中断请求和响应信号

INTR：外部可屏蔽中断请求输入端

NMI：外部非屏蔽中断请求输入端

#INTA：中断相应输出端

1. 总线保持和响应信号（CPU工作在直接存储器存取的时候：内存可以和外设直接进行数据交互）

HOLD：总线保持请求信号输入端。当CPU以外的其他设备要求占用总线时，通过该引脚向CPU发出请求。

HLDA：总线保持相应信号输出端。CPU对HOLD信号的响应信号。

* 8088和8086引线功能的比较

1. 数据总线宽度不同

8088的外部总线宽度是8位，8086是16位。

1. 访问存储器和输入输出控制信号含义不同

8088——IO/M=0表示访问内存。

8086——IO/M=1表示访问内存。

1. 其他部分引线功能的区别

* 8088内部结构（所有CPU都包含运算器、控制器、寄存器组三个部分）

1. 执行单元（EU）

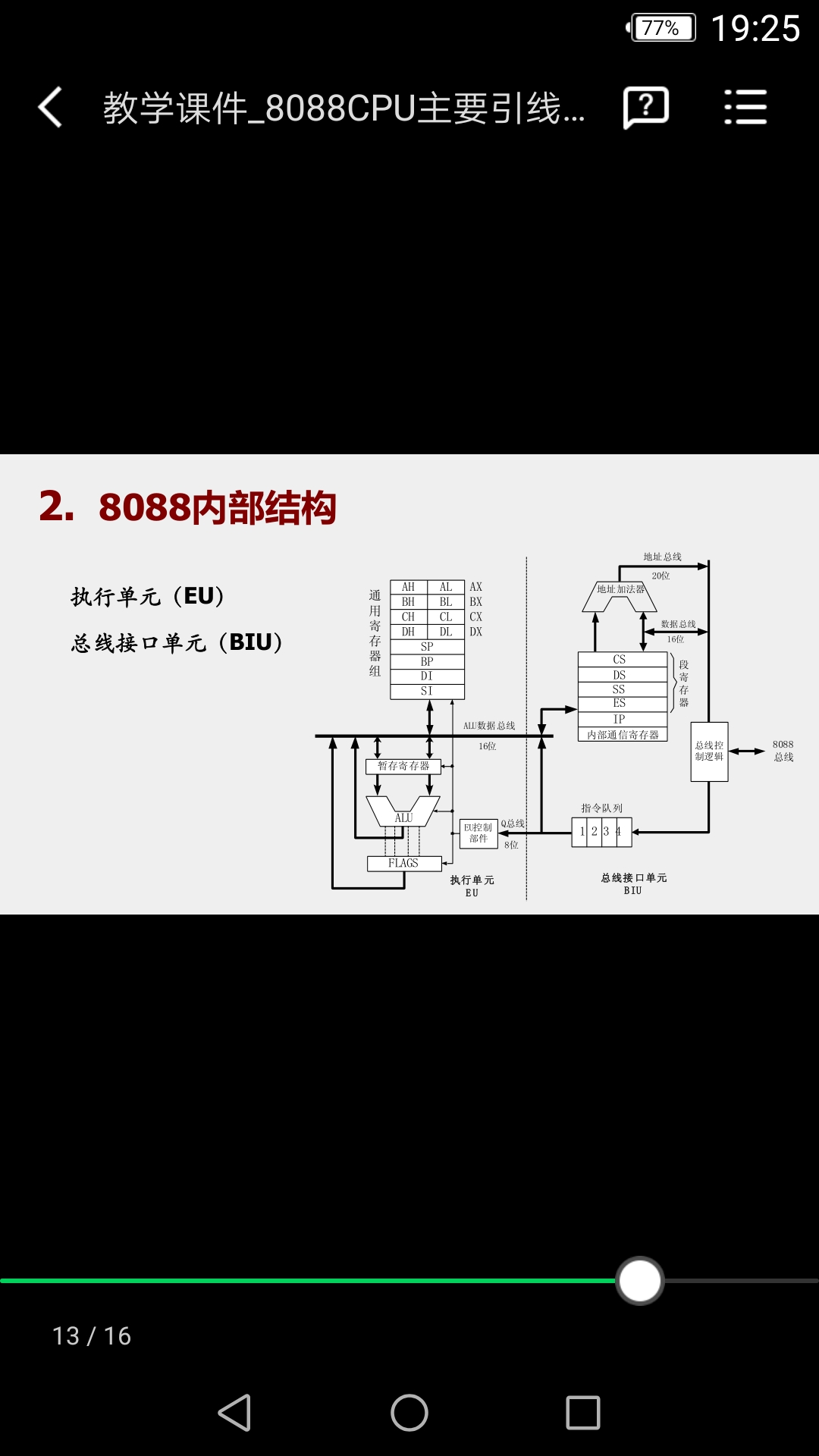
运算器

8个通用寄存器

1个标志寄存器FLAGS（标志两个数相加，有无进位、有位溢出等）

EU部分控制电路

1. 总线控制单元（BIU）



1. 内部寄存器

含14个16位寄存器，按功能分为三类：

1. 8个通用寄存器

* 数据寄存器（每个可拆分为两个8位寄存器）：AX、BX、CX、DX

AX：累加器，性能略高，普遍用来存放中间运算结果。所有I/O指令都通过AX与接口传送信息。

BX：基址寄存器，在间接寻址中用于存放基地址。

CX：计数寄存器，用于在循环或串操作指令终存放计数值。

DX：数据寄存器，在间接寻址的I/O指令中存放I/O端口地址；在32位乘除法运算时，存放高16位数。

- 地址指针寄存器：SP、BP

SP：堆栈指针寄存器，内容为栈顶的偏移地址。（有堆栈时，该寄存器即专用寄存器）

BP：基址指针寄存器，常用于在访问内存时存放内存单元的偏移地址。

【BX vs BP：作为通用寄存器，二者均可用于存放数据；作为基址寄存器，用BX表示所寻找的数据在数据段，用BP则表示数据在堆栈段。】

- 变址寄存器：SI、DI

SI：源变址寄存器

DI：目的变址寄存器

【变址寄存器在指令中常用于存放数据在内存中的地址。】

1. 4个段寄存器（存放相应逻辑段的段基地址，**段寄存器的值表明相应逻辑段在内存中的位置**）

逻辑段类型：

代码段：存放指令代码 CS

数据段：存放操作的数据 DS

附加段：存放操作的数据 ES

堆栈段：存放暂时不用但需保存的数据 SS

【在一个程序模块中，一种逻辑段只能有一个】

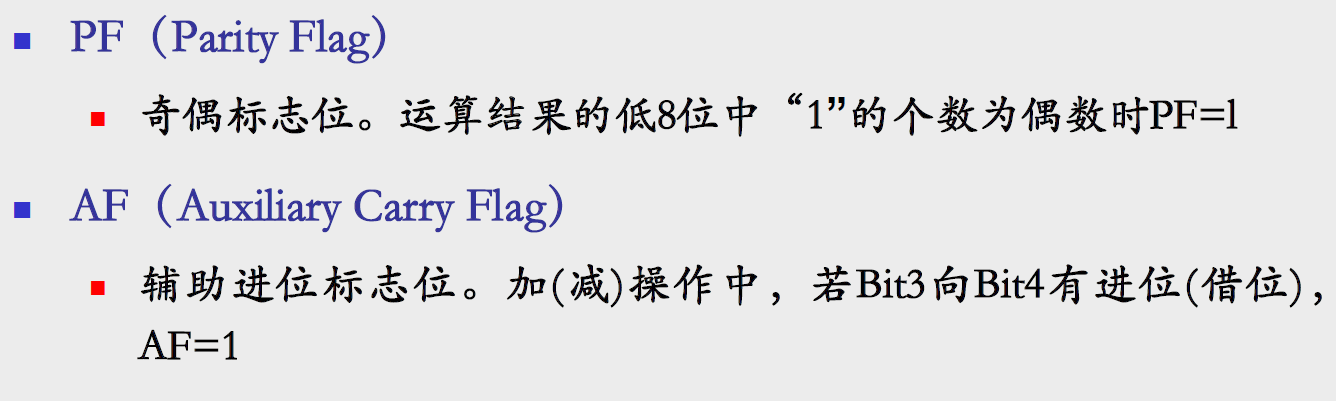
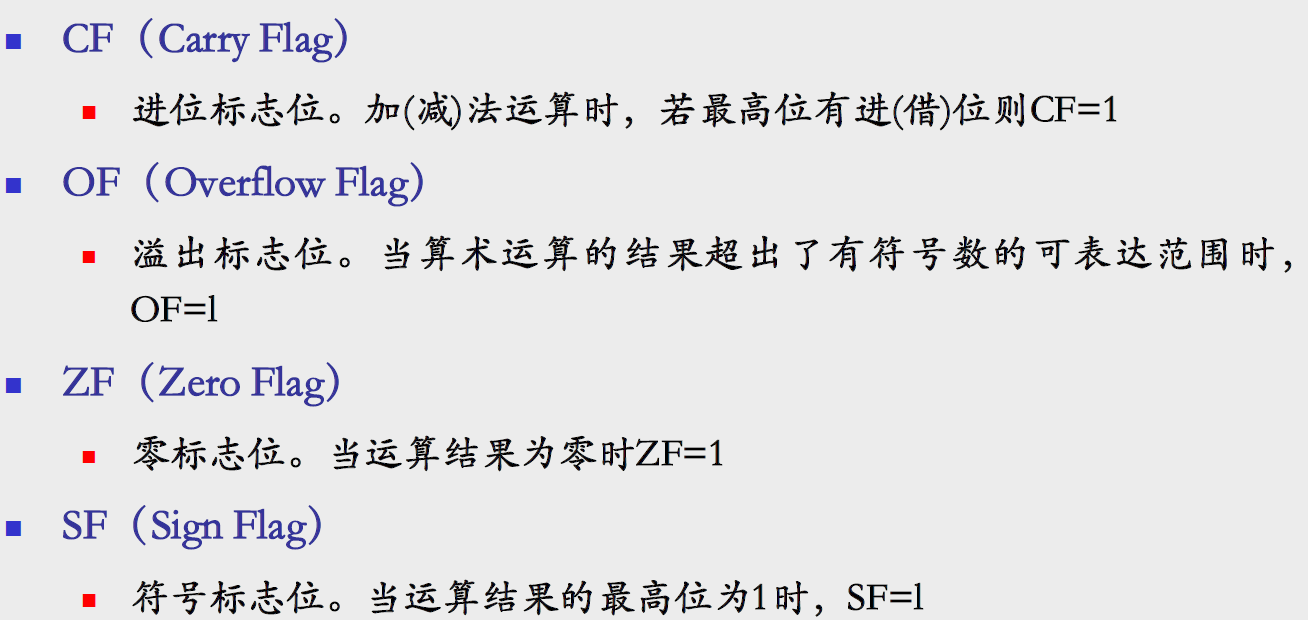
1. 2个控制寄存器

IP：指令指针寄存器，其内容为下一条要取的指令的偏移地址。

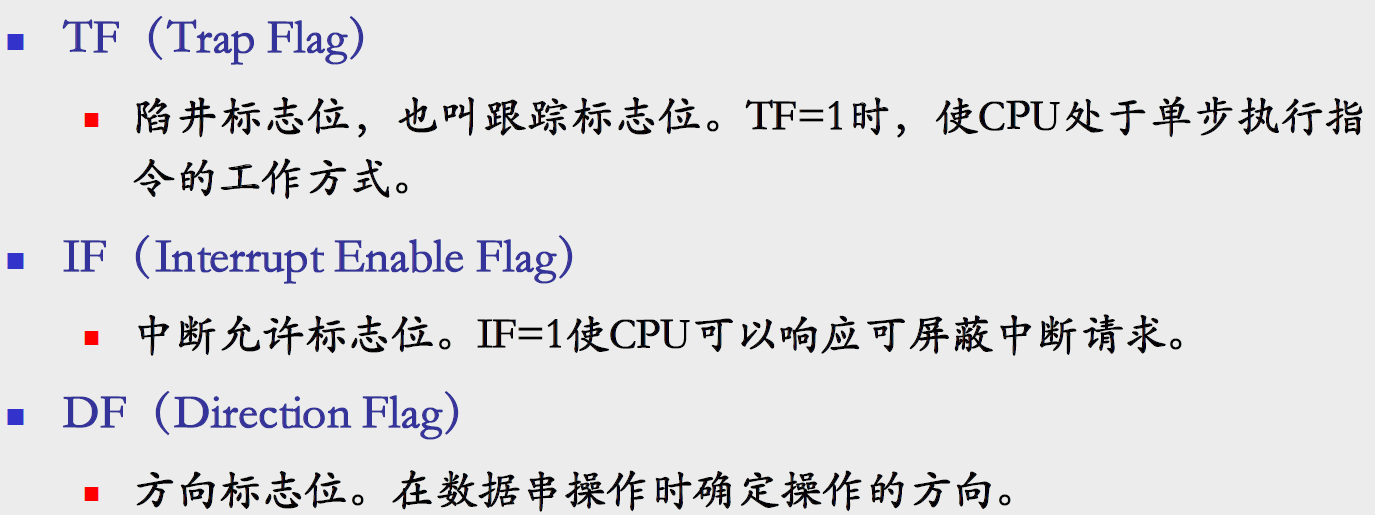
FLAGS：标志寄存器，存放运算结果的特征。（其中只有9位是有意义的）

6个状态标志位：表示运算结果的特征，CF、OF、ZF、SF、AF、PF

3个控制标志位：表示CPU当前的工作状态，IF、TF、DF



【注意：CF（无符号数溢出）、OF（有符号数溢出）、ZF、SF针对8位和16位都有效；PF、AF只针对低8位有效，AF表示第3位向第4位的进位，从0开始数】



【注意：TF=1表示中断】

1. 内存储器分段管理

* 内存单元的编址

内存每个单元的地址在逻辑上都由两部分组成：

* 段（基）地址

指示存储单元在整个内存空间中处于哪个段。

* 段内地址（相对地址/偏移地址）

指示存储单元在段中的相对位置（与段中第1个单元的距离）

* 实地址模式下的内存地址转换

物理地址 = 段基地址 \* 10H + 偏移地址

注意：1. 内存分段时，段首都是以节为边界起始的，即段首地址都必须能被16整除

2. 理论上，可以产生64K个逻辑段，但存放段地址的只有4个段寄存器。

3. 内存的分段是逻辑分段，不是物理分段。各个逻辑段在地址上可以相连，可以部分重合，也可以完全重合。

4. 每个内存单元具有唯一的物理地址，但可能有多个逻辑地址。

* 堆栈段

栈首即段首。若栈顶(SP为栈顶偏移地址) = 栈底，表示空栈；若栈顶 = 栈首，表示满栈。

1. 系统总线

* 总线时序

CPU工作时序：CPU各引脚信号在时间上的关系。

总线周期：CPU完成一次访问内存（或接口）操作所需要的时间。一个总线周期至少包括4个时钟周期。

* 总线
* 是一组导线和相关的控制、驱动电路的集合。是计算机系统中各部件之间传输地址、数据和控制信息的通道。
* 按层次结构分类：CPU总线、系统总线（连接CPU和接口间的总线，常见的两种标准为PCI总线和PCIE总线）、外部总线（接口和外设间的总线，常见的是USB总线）
* 今天的微机中大部分都是多总线结构。
* 总线的基本功能：

1. 数据传送：同步、半同步（类似于READY接口）、异步（需要接受到应答信号）
2. 仲裁控制
3. 出错处理
4. 总线驱动

* 总线主要性能指标

总线带宽（B/s）：单位时间内总线上可传送的数据量。总线带宽=位宽 \* 工作频率

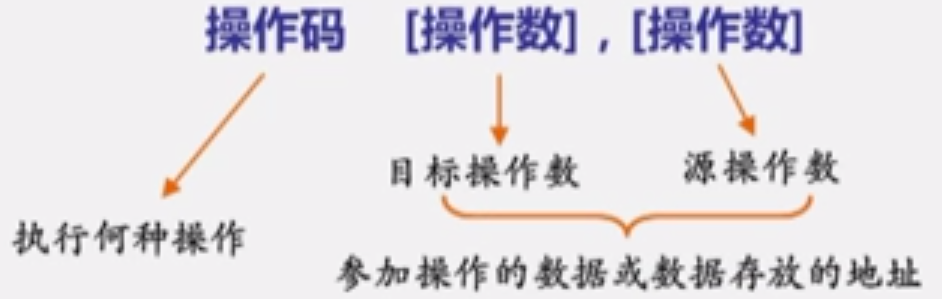
1. 指令系统
2. 基本概念

指令：控制计算机完成某种操作的命令。

指令系统：处理器所能识别的所有指令的集合。

指令的兼容性：同一系列机的指令都是兼容的。

1. 指令格式



零操作数指令：操作码

单操作数指令：操作码 操作数

双操作数指令：操作码 操作数 操作数

多操作数指令：三操作数及以上

1. 指令中的操作数

立即数、寄存器、存储器

【立即数无法作为目标操作数】

1. 寻址方式

* 操作数由指令直接给出

立即寻址 MOV AX,1200H（**立即数1200H在编译时被置于代码段**）

* 存放于寄存器中的寻址方式

寄存器寻址 MOV AX,BX

* 存放于存储器中的寻址方式

方括号中是偏移地址，逻辑段的段基地址通过默认或重设方式给出。

【存储器操作数的字长本身不确定，其字长取决于指令中另一个寄存器操作数，或通过其他方式指定字长】

1. 直接寻址 MOV AX,[1200H]

直接寻址方式下，操作数默认在数据段，但可以重设（MOV AX,ES:[1200H]）

1. 寄存器间接寻址 MOV AX,[BX]

但此处仅有4个通用寄存器可用于存放偏移地址（间址寄存器）：BX、BP、SI、DI

操作数的默认段地址取决于使用的间址寄存器（可以段重设）：

BX、SI、DI → 默认在数据段

BP → 默认在堆栈段

1. 寄存器相对寻址

MOV AX,[BX + DATA]（DATA可以为8位也可以为16位）

MOV AX,[BX]DATA 或者 MOV AX, DATA [BX] 也表示相同含义

相对寻址主要用于一位数组的操作。常将位移量作为“表头”地址，间址寄存器的值作为表内相对地址。

1. 基址变址寻址

操作数的偏移地址为 一个基址寄存器（BX/BP）的内容 + 一个变址寄存器（SI/DI）的内容。

操作数的段地址由选择的基址寄存器决定：基址寄存器为BX，默认为数据段；基址寄存器为BP，默认在堆栈段。

1. 相对基址变址寻址

基址寄存器 + 变址寄存器 + 位移量

操作数的段地址由选择的基址寄存器决定。

基址变址相对寻址主要用于二维表格操作。

1. 隐含寻址

MUL BL（AX = AL\* BL）

1. 8086指令集（共96条）

* 数据传送

1. 通用数据传送指令

* 一般数据传送指令MOV

注意：a. 两个操作数字长必须相同。

b. 两操作数不允许同时为存储器操作数。

c. 两操作数不允许同时为段寄存器。

d. 在源操作数是立即数时，目标操作数不能是段寄存器。

e. IP和CS不作为目标操作数，FLAGE一般也不作为操作数在指令中出现。

* 堆栈操作指令【先进后出、以字为单位】

压栈指令：PUSH OPRD

指令执行过程：a. SP -2 → SP（下面为高地址，上面为低地址）【先减后压】

b. 操作数高字节 → SP + 1

c. 操作数低字节 → SP

出栈指令：POP OPRD

指令执行过程：a. SP → 操作数低字节

b. SP + 1 → 操作数高字节

c. SP +2 → SP

不能使用POP给CS赋值

【注意：对于单操作数的指令，操作数不能是立即数；如果操作数是内存单元，要显式地给出操作数的字长；单操作数指令的操作数都是16位。】

* 交换指令 XCHG REG,REG

要求：两个操作数中不能出现立即数；两个操作数至少有一个是寄存器；不允许使用段寄存器。

* 查表指令 XLAT

说明：用BX的内容代表表格首地址，AL内容为表内位移量，将结果送入到AL。

* 字位扩展指令

说明：该指令均针对的是有符号数的扩展，有符号数扩展将符号位扩展到高位。

CBW 将AL内容扩展到AX

CWD 将AX内容扩展到DX AX

1. 输入输出指令

I/O端口：I/O接口中用于存储数据、可以直接被CPU访问的寄存器。

计算机输入输出系统中可以包含若干接口控制电路（芯片），每个接口中都包含了1个或多个端口。

* IN acc,PORT
* OUT PORT,acc

【acc代表AL/AX，但是不能是AH，PORT是端口地址】

* 直接寻址

端口地址为8位时，指令中直接给出8位端口地址，最大可寻址256个接口。

* 间接寻址

端口地址为16位时，指令中的端口地址必须由DX指定，最大可寻址64K个接口。

1. 地址传送指令

* 取偏移地址指令LEA REG,MEM（取近地址指针）

将变量的16位偏移地址写入到目标寄存器。（**源操作数一定是存储器操作数**）

LEA针对的变量是一个静地址指针，即在本数据段内，不考虑段基地址。

* LDS指令（取远地址指针，多模块编程使用）

LDS REG,MEM 将源操作数的偏移地址送到目标寄存器，将源操作数的段地址送到DS

* LES指令（取远地址指针，多模块编程使用）

LES REG,MEM 将源操作数的偏移地址送到目标寄存器，将源操作数的段地址送到ES

1. 标志传送指令

* LAHF：将FLAGES的低8位装入到AH

SAHF：将AH的写入到FLAGES的低8位

隐含操作数AH

* PUSHF

POPF

隐含操作数FLAGES

【除标志传送指令外，其他指令的执行都对标志位不产生影响。】

* 算术运算

1. 加法指令

* 普通加法指令ADD
* 带进位的加法指令ADC（**多用于多字节数的相加，使用前要将CF清零**。）

ADD、ADC的执行会对全部6个标志位产生影响。

* 加1指令INC

操作数不能是段寄存器和立即数，**指令执行不会对CF造成影响**。

1. 减法指令

* 普通减法指令SUB

SUB OPRD1, OPRD2：OPRD1 - OPRD2 → OPRD1

* 考虑借位的减法指令SBB
* 减1指令DEC（标志位影响同INC）
* 比较指令CMP

CMP OPRD1, OPRD2：OPRD1 - OPRD2

指令执行的结果不影响目标操作数，仅影响标志位。主要用于比较两个数的大小。

CMP AX, BX

* + 两个无符号数的比较：

若AX ≥ BX → CF = 0

若AX < BX → CF = 1

若AX = BX → CF = 1, ZF = 1

* + 两个带符号数的比较：

两个数的大小由OF和SF共同决定。

OF和SF状态相同时，AX ≥ BX

OF和SF状态不同时，AX < BX

* 求补指令NEG

NEG OPRD：0 – OPRD → OPRD

在绝大多数情况下，CF = 1，除了OPRD=0。

当操作数为80H(-128)或者8000H(-32768)，则执行后，结果不变，但OF置1，其他情况下OF均置0。

对一个负数取补码就相当于用零减去此数。

1. 乘法运算指令

* 无符号的乘法指令MUL

MUL OPRD：OPRD不能是立即数。

OPRD为字节数：AL × OPRD → AX

OPRD为16位数：AX × OPRD → DXAX

单操作数指令要求：a. 操作数不能是立即数。

b. 操作数如果是存储器操作数，必须要用属性运算符PTR显示给出字长。

* 带符号的乘法指令IMUL

执行原理：a. 将两个操作数取补码；

b. 做乘法运算；

c. 将乘积取补码。

乘法指令采用隐含寻址，隐含了存放被乘数的累加器AL或AX，以及存放结果的AX或DX、AX。

1. 除法运算指令

* 无符号除法DIV

DIV OPRD

若 OPRD是字节数，执行AX/OPRD → AL=商，AH=余数

若OPRD是双字节数，执行DXAX/OPRD → AX=商，DX=余数

* 有符号出发IDIV

【除法指令要求被除数是除数的双倍字长。】

* 逻辑运算和移位

1. 逻辑运算指令

逻辑运算指令总体上速度大于算术运算指令。

除“非”以外，其他逻辑运算指令都会对除AF状态标志位产生影响。

无论运算结果如何都会使CF和OF清零。“非”运算指令不会对任何标志位产生影响。

* 与运算AND

应用：实现与运算

某些位实现清零：AND AL,0FH

在操作数不变的情况下，使CF和OF清零：AND AX,AX

* 或运算OR

应用：实现或运算

某些位置1：OR CL,0FH（二进制数转ASCII码：OR AL，30H）

在操作数不变的情况下，使CF和OF清零：OR AX,AX

* 非运算NOT
* 异或XOR

应用：实现异或运算

最高位取反，其他位不变：XOR BL,80H

清零：XOR AX,AX

* 测试指令TEST

TEST OPRD1,OPRD2：执行与运算，但将操作结果舍弃

1. 移位操作指令

指令格式在形式上是双操作数，本质上为单操作数，故目标操作数为存储器操作数时，需要说明其字长。当移动1位时，可以由指令直接给出，移动两位及以上位数必须由CL指定。

* 线性移位

【**算术针对有符号数，逻辑针对无符号数**】

算术左移指令：SAL OPRD,CL

逻辑左移指令：SHL OPRD,CL

左移均是最高位移到CF，低位补零。

算术右移指令：SAR OPRD,CL

最低位移入CF，最高位会将复制原符号位

逻辑右移指令：SHR OPRD,CL

最低位移入CF，最高位补0.

* 循环移位

不带进位位CF的循环移位

左移ROL：最高位移到CF，最高位同时移动到最低位

右移ROR：最低位移到CF，最低位同时移动到最高位

带进位位CF的循环移位

左移RCL：最高位移到CF，同时CF移到最低位

右移RCR：最低位移到CF，同时CF移到最高位

应用：a. 用于对某些位状态的测试；

b. 高位部分和低位部分的交换；

c. 与非循环移位指令一起组成32位或更长字长数的移位。

* 串操作

操作对象为一个数据块或者一个字符串。

串操作指令需要确定：串所在的区域、原串和目的串起始地址、串长度、串的操作方向

1. 串操作指令的要求：

a. 源串一般存放在数据段，偏移地址必须由SI指定。**允许段重设**。

b. 目标串必须在附加段，偏移地址由DI指定。

c. 串长度必须由CX指定。

d. 串的操作方向由DF标志位决定。指令根据DF状态自动修改地址指针：

DF = 0 → 增地址方向

DF = 1 → 减地址方向

1. 串操作指令前边可以增加**重复前缀**，可以实现对CX值的自动修改，实现循环。

重复前缀：

**无条件重复**

* REP

当CX ≠ 0时，REP后的指针将继续重复执行。

常用于传送类指令前，未传完则继续传送。

**条件重复**

* 相等（为零）重复 REPE/PEPZ：

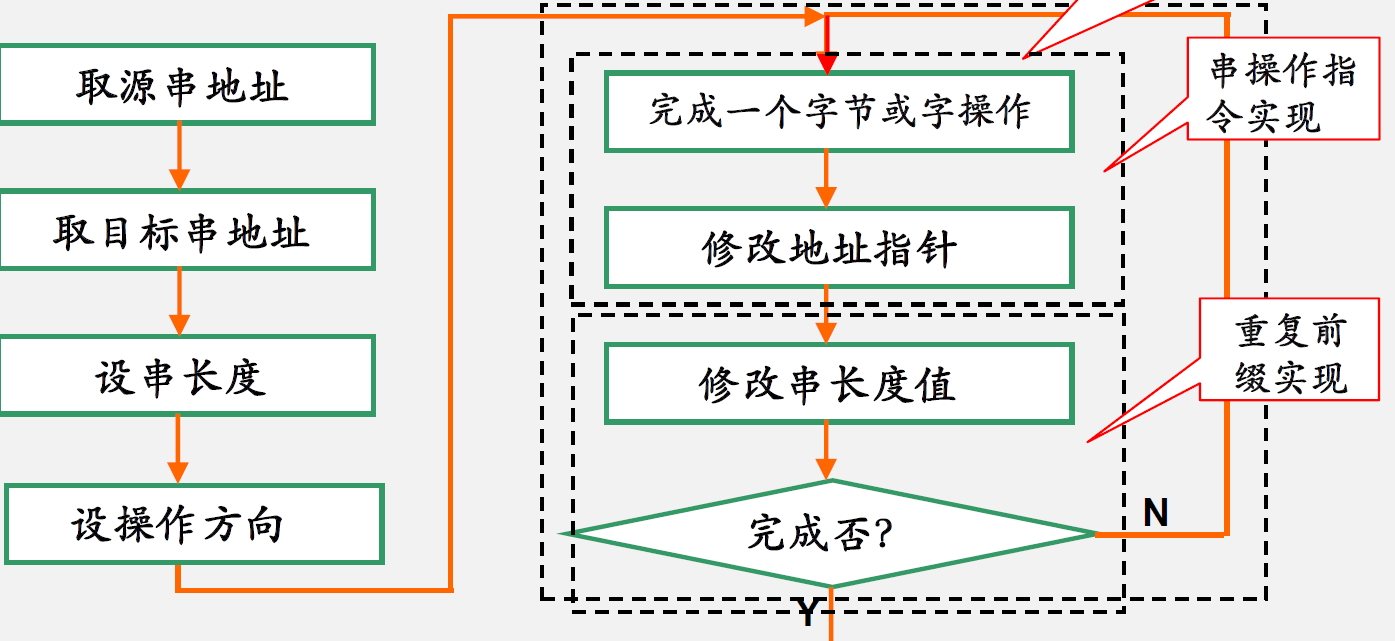
CX ≠ 0 且 ZF = 1，则前缀后的指令将继续重复执行

* 不相等（不为零）重复 REPNE/REPNZ：

CX ≠ 0 且 ZF = 0，则前缀后的指令将继续重复执行

前缀操作对标志位不会产生影响。

1. 执行细节



则若按增地址方向操作，串操作结束时，

串传送指令：指针将指向串尾+1

串比较类指令：指针将指向结束位+1

若按减地址方向操作，串操作结束时，

传传送指令：指针将指向串尾-1

传比较指令：指针将指向结束位-1

1. 具体指令

* 串传送MOVS

MOVS OPRD1,OPRD2 通常用于源操作数需要段重设的情况下

MOVSB 按字节传送

MOVSW 按字传送

* 串比较CMPS（指令同MOVS）
* 串扫描SCAS：常用于在指定存储区域中寻找某个关键字（关键字位于累加器）

SCAS OPRD（目标操作数）

SCASB 源操作数是AL

SCASW 源操作数是AX

* 串装入LODS

常用于将内存某个区域的数据串依次装入累加器，以便显示或输出到接口。

**LODS指令一般不加重复前缀**。

LODS OPRD（源操作数）

LODSB：[DS:SI] → AL

LODSW：[DS:SI] → AX

* 串送存STOS

常用于将内存某个区域置同样的值。

STOS OPRD（目标操作数）

STOSB：AL → [ES:DI]

STOSW：AX → [ES:DI]

* 程序控制

1. 程序控制类指令的本质是：控制程序的执行方向。

决定程序执行方向的因素：CS、IP

1. 转移指令

* 无条件转移指令

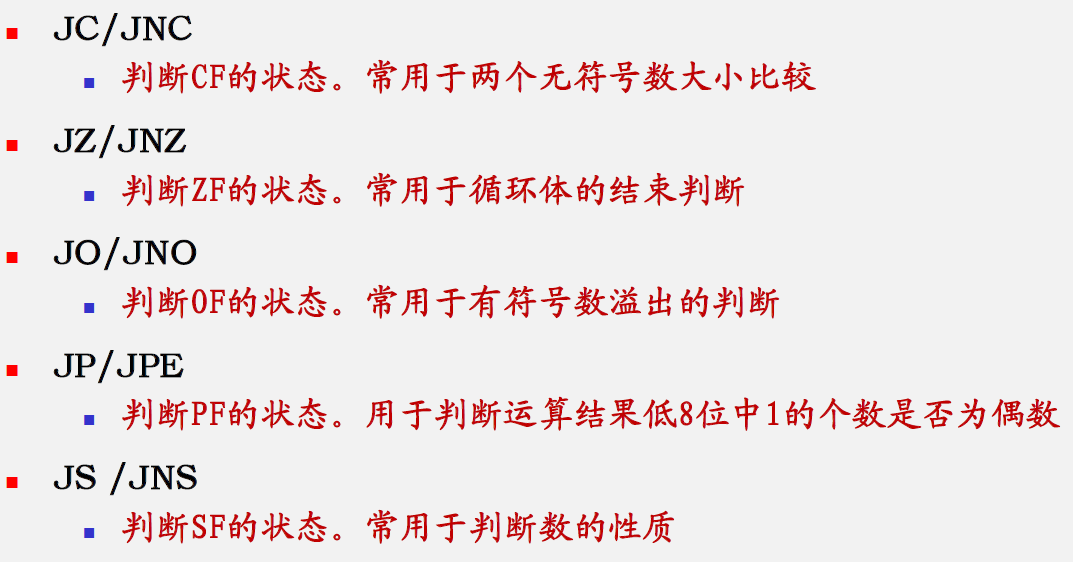
JMP OPRD

OPRD如果和JMP在同一个代码段，称为**段内转移**；如果与JMP不在同一个代码段，称为**段间转移**。

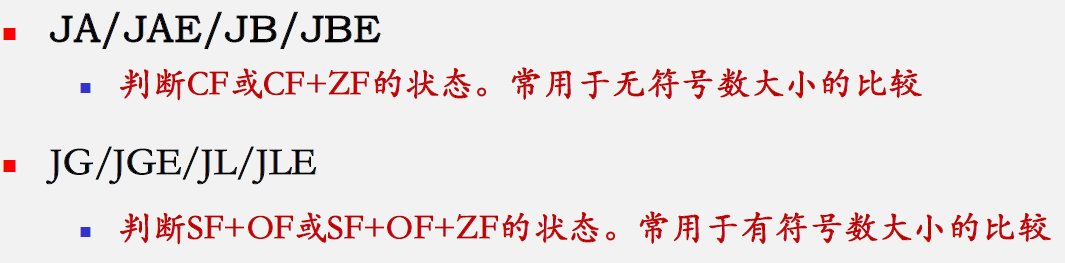
* + 段内转移：目标地址是16位偏移地址。可以由指令中直接给出目标地址，称为**段内直接转移**（JMP Label，Label为近地址标号）；也可由指令中的寄存器或存储器操作数指出目标地址，称为**段内间接转移**（JMP WORD PTR[BX]）。
  + 段间转移：目标地址是32位，包括段地址和偏移地址。可以由指令中直接给出目标地址，称为**段间直接转移**（JMP FAR Label，Label为近地址标号）；也可由指令中32位存储器操作数指出目标地址，称为**段间间接转移**（JMP DWORD PTR[BX]：低地址单元为偏移地址IP，高地址单元为段基地址CS）。
* 条件转移指令

条件转移指令均为**段内短转移**，转移范围：-128 ~ +127。

* 基于1个标志位状态实现转移的指令



* 基于2个或3个标志位状态实现转移的指令



* 基于CX内容转移的指令



1. 循环控制指令

循环范围：以当前IP为中心的-128 ~ +127范围内循环。

循环次数必须由CX寄存器指定。

* 无条件循环指令 LOOP Label

当CX≠0时进行循环。

操作相当于：DEC CX;JNZ 符号地址

* 条件循环指令

LOOPZ Label → 继续循环的条件：CX≠0，且ZF=1

LOOPNZ Label → 继续循环的条件：CX≠0，且ZF=0

1. 过程调用指令：用来调用一个子程序

过程调用必须保护返回地址。（断点：过程调用下一条指令的地址）

过程调用执行过程：

* 保护断点：将调用指令的下一条指令的地址（断点）压入堆栈；
* 获取子程序入口地址；
* 执行子程序：功能实现，参数的保存及恢复；
* 恢复断点：将断点偏移地址由堆栈弹出。
* 段内调用

段内直接调用 CALL TIMER

段内间接调用 CALL WORD PTR[SI]

* 段间调用（子程序与原调用程序不在同一代码段）

保护段点压栈时先将断点的CS压栈，再压入IP。

段间直接调用 CALL FAR TIMER

段间间接调用 CALL DWORD PRT[SI]

* 返回指令 RET

从堆栈中弹出断点地址，返回源程序。子程序的最后一条指令必须是RET。

1. 中断控制指令

* 中断：某种异常或随机**事件**（中断源）使CPU暂时停止正在运行的程序，转去执行一段特殊处理程序，并在处理结束后返回原程序被中断处理继续执行的过程。
* 中断与过程调用比较：

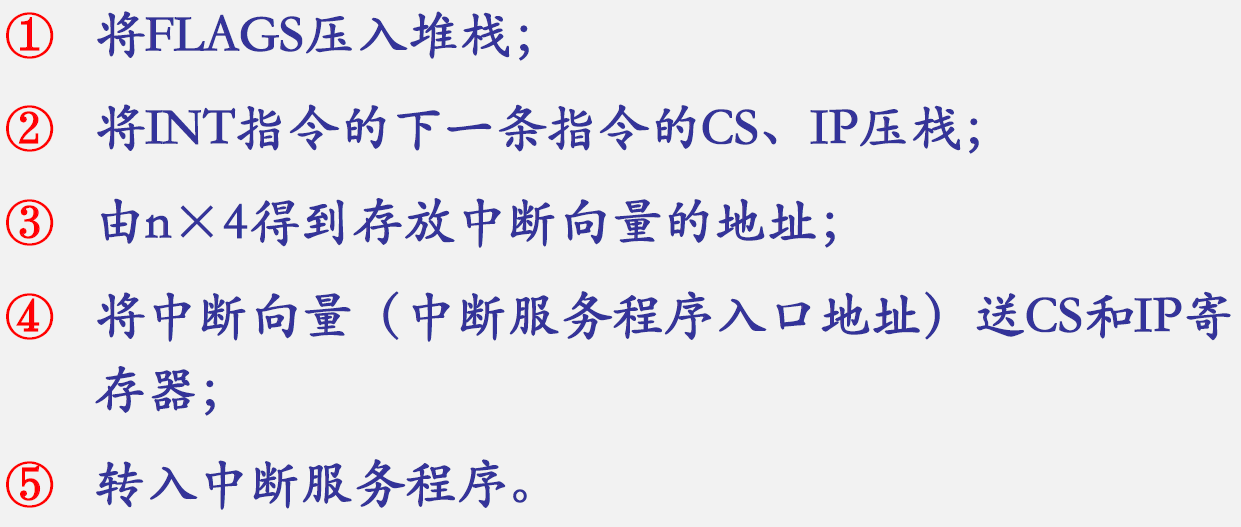
a. 相似点：从一个正在执行的过程转向另一个过程（处理程序），并在执行完后返回原程序继续执行。

b. 区别

* 中断是随机事件或异常事件引起，调用是事先已在程序中安排好。
* 调用指令在指令中直接给出子程序入口地址，中断指令只给出中断向量码，入口地址则在向量码指向的内存单元里。
* 调用可以是近过程调用或远过程调用，中断处理程序均为远过程
* 响应中断请求不仅要保护断点地址，还要保护FLAGS内容。
* 中断指令 INT n（n为中断类型码0 ~ 255）

说明：遇到中断指令后，系统首先会n \* 4，得到存放中断服务子程序入口地址单元的偏移地址。（数据段内存中0 ~ 003FFH为中断向量表，共1K个字节单元，每4个单元构成了一个中断子程序的入口地址，该区域单元在数据段，段地址为DS）

中断指令执行过程：



* 中断返回指令IRET（中断程序最后一条指令必须为中断返回指令）
* 处理器控制

该类指令用来对CPU进行控制，如修改标志寄存器，使CPU暂停，使CPU与外部设备同步等。

特点：处理器控制指令的控制对象为CPU，均为零操作数格式指令。

* 对标志位的操作



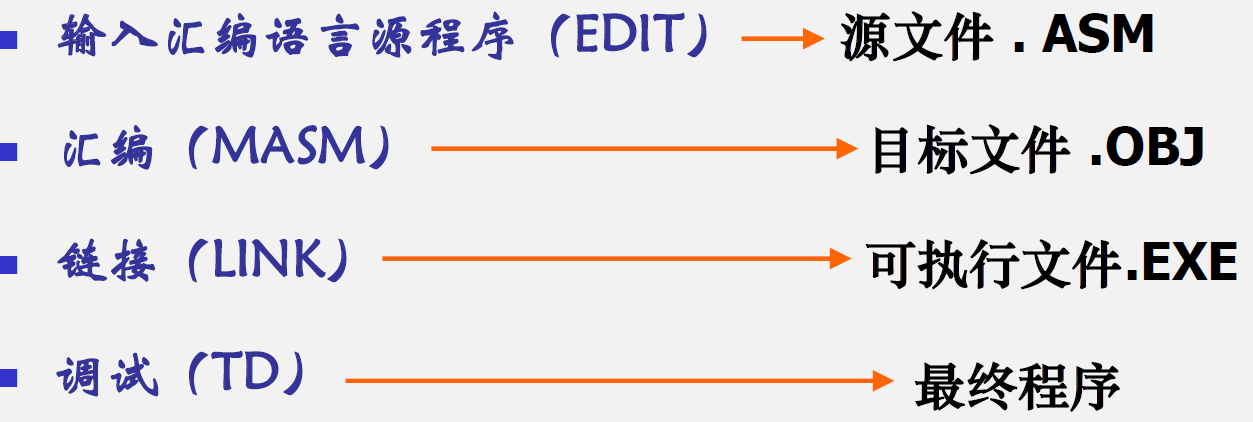
* 与外部设备的同步

使CPU暂停 HLT

1. 汇编语言程序设计
2. 汇编语言源程序：用助记符编写的程序

汇编程序：源程序的编译程序。

1. 执行过程



1. 汇编语言语句类型

指令性语句 → CPU执行的语句，能够生成目标代码



指示性语句 → CPU不执行，而由汇编程序执行的语句，不生成目标代码



指示性语句中至少有一个操作数，操作数个数最大为逻辑段容量64K。

1. 汇编语言语句中的操作数

寄存器、存储单元、常量（数字常量、字符串常量）、变量或标号、表达式

取值运算符：用于分析存储器操作数的属性

* OFFSET → 取得其后变量或标号的偏移地址【MOV BX, OFFSET DATA】
* SEG → 取得其后变量或标号的段地址

属性运算符：用于指定其后存储器操作数的类型

* PTR

1. 伪指令：不会被CPU执行

作用：定义变量、分配存储区、定义逻辑段、指示程序开始和结束、定义过程等。

* 数据定义伪指令

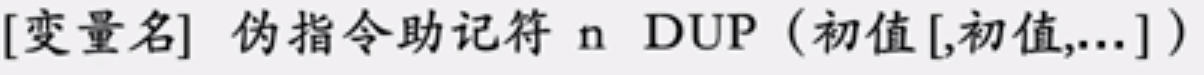
用于定义数据区中变量的类型及其所占内存空间大小。

* 常用类型伪指令：

定义字符串必须用DB伪指令。

例如：DATA DB 11H,12H,13H

* 重复操作符：**常用来声明一个数据区**



例如：M1 DB 10 DUP (0)

* “?”的作用：表示随机值，用来预留空间

M1 DB 34H, ’A’, ?

* 符号定义伪指令
* 符号名 EQU 表达式

用符号名取代后边的表达式，不可重复定义

* 段定义伪指令

说明逻辑段的起始和结束；说明不同程序模块中同类逻辑段之间的联系形态。

* 格式

段名 SEGMENT [定位类型][组合类型][‘类别’]

…

段名 ENDS

段名表示逻辑段的段基地址；定位类型说明逻辑段的起点，通常是以节边界作为起始（一节是16个内存单元）；组合类型为装入内存时各逻辑段的组合方式，默认为不组合；“类别”：链接时不同程序模块中的同类逻辑段将被装入连续存储区。

* 设定段寄存器微指令：说明所定义逻辑段的性质
* ASSUME 段寄存器:段名[, 段寄存器:段名]
* 结束伪指令：表示汇编语言源程序的结束
* END [标号]
* 过程定义伪指令
* 格式

过程名 PROC [NEAR/FAR]

参数保护

…

参数恢复

RET

过程名 ENDP

过程名为过程入口的符号地址。后参数若为近过程体，NEAR可以省略。

* 调用格式：CALL 过程名
* 宏命令伪指令

源程序中由汇编程序识别的具有独立功能的一段程序代码。

当源程序中需要多次使用同一个程序段时，可以将该程序段定义为一个宏。

宏与子程序的区别：宏不能被编译链接为一个独立的程序。

* 定义格式

宏命令名 MACRO <形式参数>

…

ENDM

* 调用格式：宏命令名 <实际参数>

汇编后源程序中会进行宏展开（将实际参数代入取代原形式参数）。

并不会减少编译后程序代码的长度，但会减少编程代码的长度。

* 调整偏移量伪指令

规定程序或变量在逻辑段中的起始地址。默认情况下，程序或变量在逻辑段中的起始编译地址为0。

* ORG 表达式

表达式结果必须为非负常数。

1. 系统功能调用

DOS/BIOS功能调用是调用系统内核子程序，均是通过中断方式调用。

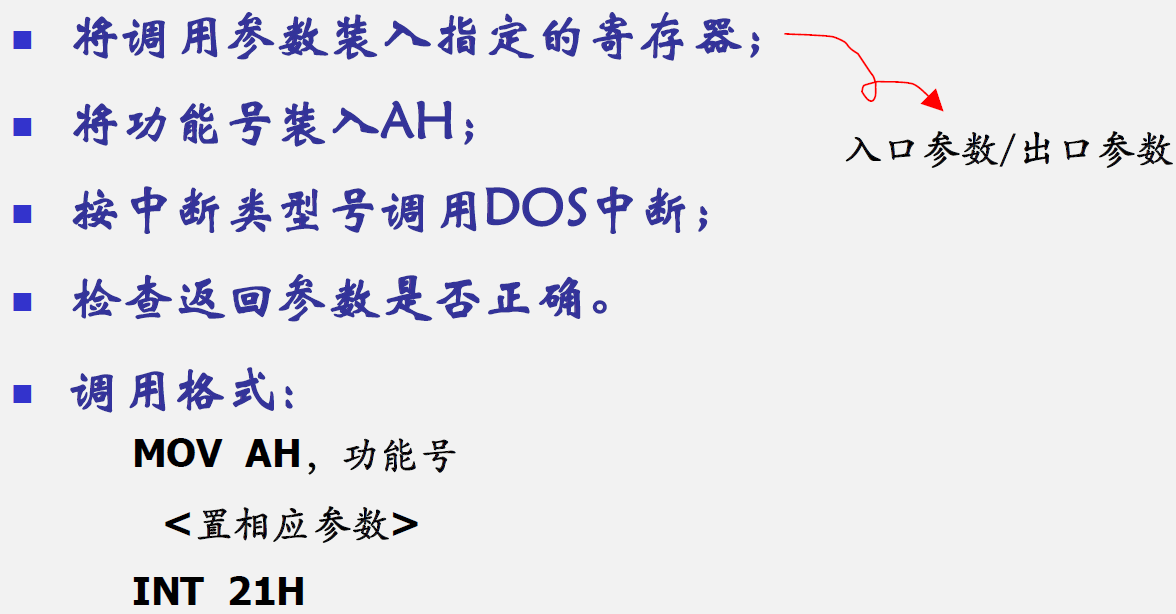
* BIOS：驻留在ROM中的基本输入/输出系统。

功能：加电自检、装入引导、主要I/O设备处理程序及接口控制。

* DOS：磁盘操作系统

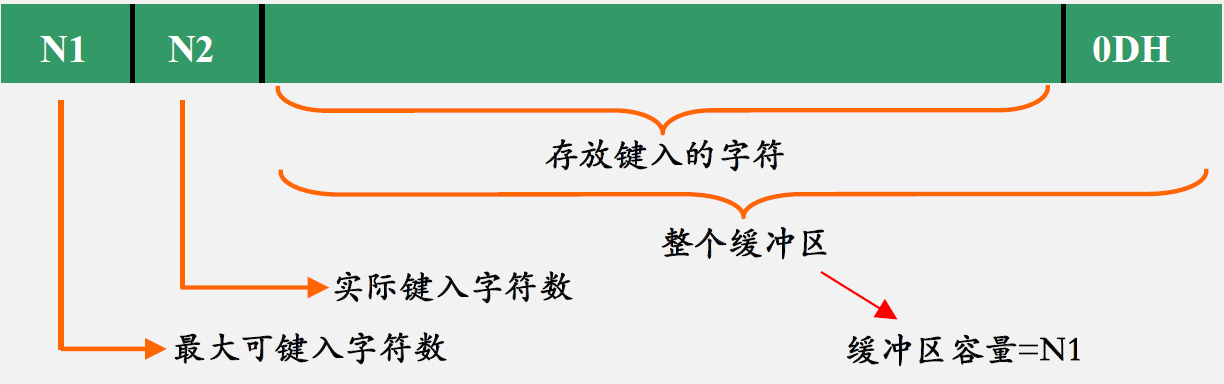
DOS中断包括：设备管理、目录管理、文件管理、其他。用中断类型码区分。

DOS功能调用的基本步骤：



* 单字符出入 功能号为01，输入字符的ASCII码会被保存在AL中。
* 字符串输入

接收由键盘输入的一串字符，输入的字符串存储在内存指定区域里（字符输入缓冲区），字符输入缓存区的格式：



【缓冲区最大只能是255】【eg. DAT1 DB 20, ?, 20 DUP(?)】

字符串输入功能号为10，缓存区必须定义在数据段，整个字符输入首地址必须给DS:DX。

* 单字符显示输出

功能号为2，要显示字符需要交给DL

* 字符串显示输出

功能号为9，待输出字符串的偏移地址给DS:DX

注意：被显示的字符串必须以“$”结束；

所显示的内容不应出现非可见的ASCII码；

若考虑输出格式美观，可以定义字符串后加入回车符(0DH)和换行符(0AH)，回车符能保证光标出现在最后一个输出字符之后。

* 返回操作系统功能

功能号：4CH

程序执行完后能正常返回OS，常位于程序结尾处。

### 三、半导体存储器

1. 基本概念
2. 半导体存储器

半导体存储器由能够表示二进制数“0”和“1”的、具有记忆功能的半导体器件组成。

内存条一般由8块芯片组成。

1. 能存放一位二进制数的半导体器件称为一个存储元。

每个存储单元由8个存储元构成。

1. 分类

内存储器：随机存取存储器（RAM）、只读存储器（ROM）

RAM分类：静态存储器（SRAM），存储元为双稳态电路。速度快，稳定，但价格昂贵，被用来做高速缓冲存储器。 动态存储器（DRAM），存储元为电容，需要复杂的外围控制电路。

ROM：掩模ROM、一次性可写ROM、EPROM、EEPROM

1. 主要技术指标

存储容量：存储单元个数 × 每单元的二进制数位数

存取时间：实现一次读/写所需要的时间

存取周期：连续启动两次独立的存储器操作所需间隔的最小时间

可靠性、功耗

1. 微机中的存储器
2. 分类

内存储器：主内存、高速缓冲存储器

外存储器：联机外存、脱机外存

1. 存储器系统

将两个或两个以上速度、容量和价格各不相同的存储器用硬件、软件或软硬件结合的方法连接起来，使整个系统的存储速度接近最快的存储器，容量接近最大的存储器，价格接近最便宜的存储器。

1. 微型计算机中的存储器系统主要有：

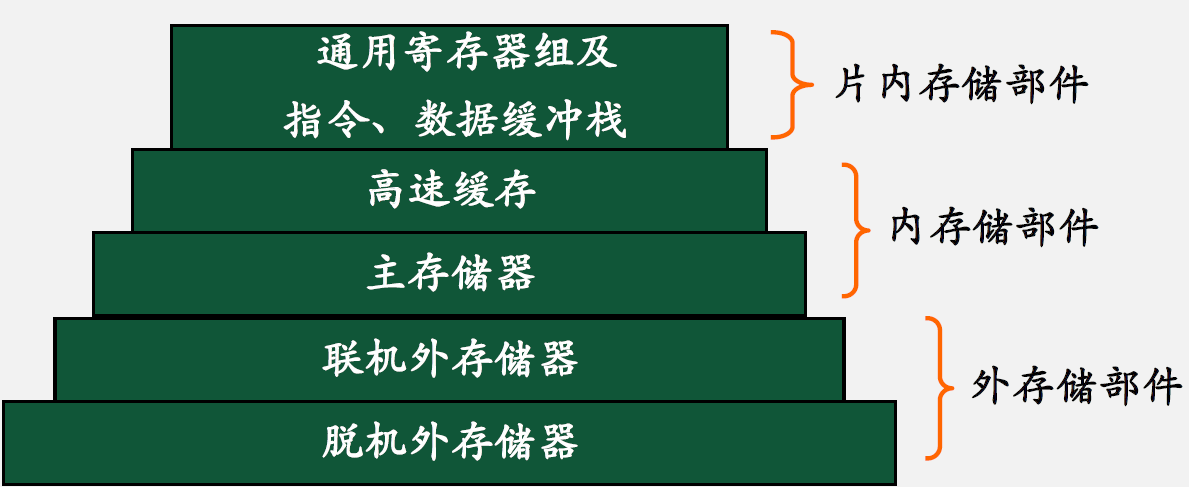
Cache存储器系统：由高速缓冲存储器（Cache）和主内存构成，由硬件系统负责管理。

目标：提高整个系统的存取时间。

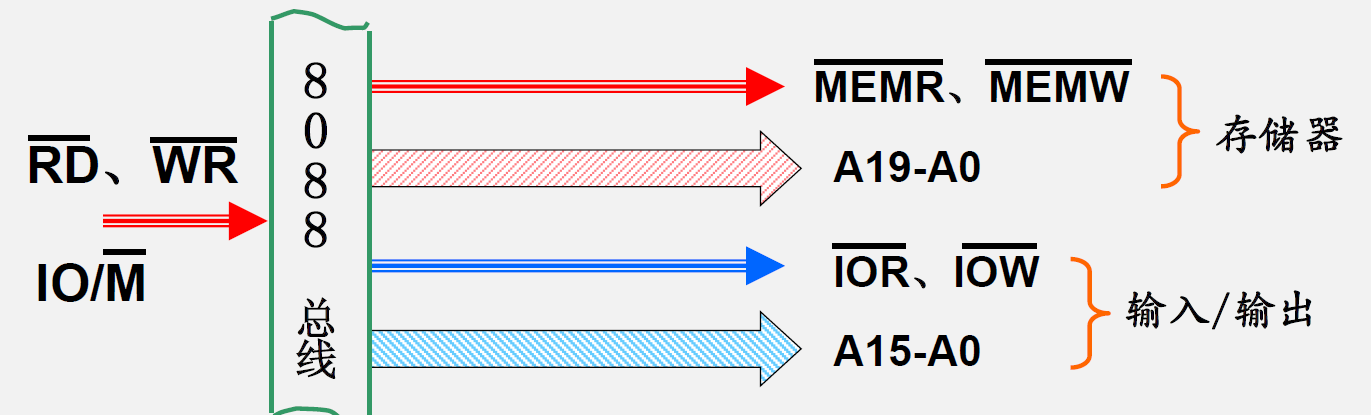
系统存取时间 = 命中率 \* Cache存取时间 + 不命中率 \* 主存存取时间

虚拟存储器系统：由主内存和部分硬磁盘构成，主要由操作系统管理。

目标：扩大存储容量。



1. 随机存取存储器
2. 8088总线信号



1. 存储单元的编址：高位地址和地位地址

高位地址又称片选地址，用于选择芯片。

每个芯片上有一个片选信号，通常低电平有效（因为干扰信号多为尖峰信号）。

1. 译码电路

将输入的一组高位地址信号通过变换，产生一个有效的输出信号，用于选中某一个存储器芯片，从而确定了该存储器芯片在内存中的地址范围。

1. DRAM

存储元由电容构成。故需要定时刷新（定时对存储元进行读或写操作）

1. SRAM

存储元由双稳电路构成，存储信息稳定。

* **示例芯片：SRAM6264**。

容量：8K \* 8b

地址线：A0 -- A12，213=8K

数据线：D0 – D7，8b

输出允许信号：#OE

写允许信号：#WE

片选信号：#CS1，CS2（两个片选信号都要有效，芯片才能正常工作）

1. 译码方式

* 全地址译码（微机系统通常都用全地址译码）

用全部的高位地址信号作为译码信号，使得存储器芯片的每一个单元都占据一个唯一的内存地址。

* 部分地址译码（通常用于嵌入式系统设计）

用部分高位地址信号作为译码信号，使得被选中存储器芯片占有几组不同的地址范围。

1. 只读存储器ROM
2. EPROM

可多次编程写入、掉电后内容不丢失、内容的擦擦需用紫外线擦除器。【擦除后每个单元都是FFH，方可去重新写入】

EPROM芯片因其较高的稳定性，使用时常用做程序存储器，存放相应的控制程序，一般不存放数据。

* **示例：EPROM 2764**

容量：8K \* 8b

地址线：A0 -- A12，213=8K

数据线：D0 – D7，8b

输出允许信号：#OE

片选信号：#CE

编程脉冲输入：#PGM

其引脚与SRAM6264完全兼容。

* 工作方式

数据读出：可在线随机读取

编程写入：不可再线写操作；需专用编程写环境。在编程写脉冲和写电压控制下完成写操作（每一个写脉冲写入1字节数据）

擦除：紫外线擦除。

1. EEPROM

可在线编程写入、掉电后内容不丢失、电可擦除

* 工作方式

数据读出：可在线随机读取

编程写入：可按字节写入，每次写入一个字节；也可自动页写入，每次写入一页（1~32字节）

擦除：字节擦除、片擦除

* **示例：EEPROM 98C64A**

容量：8K \* 8b

地址线：A0 -- A12，213=8K

数据线：D0 – D7，8b

输出允许信号：#OE

写允许信号：#WE

片选信号：#CE

状态输出端：READY/#BUSY，高电平才允许写入，只有该端需要通过接口和系统连接。

* 应用注意

可通过程序实现对芯片的读写；

仅当READY/#BUSY = 1才能进行“写”操作；

“写”操作的方法：

* 根据参数定时写入
* 通过判断READY/#BUSY端的状态进行写入：仅当该端为高电平时才写入下一个字节。
* 中断控制方法：当READY/#BUSY端为高电平时，该高电平可作为中断请求信号。

1. 闪速存储器Flash

U盘、数码相机、手机存储卡。

通过向内部控制器写入命令的方法来控制芯片的工作方式。（也属于一种EEPROM，但是由于EEPROM中READY/#BUSY信号需要通过接口读入速度较慢，而Flash是通过读取内部状态寄存器来判断是否可以进行读写）

* 与SRAM的区别

在进行写入和擦除操作时需要12V编程电压。

* 与普通EEPROM的区别

通过读状态寄存器的内容确定是否可继续写入。

* 工作方式

数据读出：读单元内容、读内部状态寄存器内容、读芯片厂家及器件标记

编程写入：数据写入、写软件保护

擦除：字节擦除、块擦除、片擦除、擦除挂起

1. 存储器扩展技术

用已有的存储器芯片构造一个需要的存储空间。

存储器芯片的存储容量 = 单元数 × 每单元的位数

1. 位扩展：扩展字长

**实例芯片：DRAM2164A**

64K \* 1bit，采用行地址和列地址来确定一个单元，为了减小芯片体积。行列地址分时传送，共用一组地址信号线，所以只需要8根片内地址信号线。

位扩展：确保所有芯片具有完全相同的地址范围。

1. 字扩展：扩展单元数

确保所有芯片具有完全不同的地址范围。

【**设计时要读写控制信号要作为译码控制电路的输入信号，当有一个信号有效时后面的译码电路才应该工作。**】

1. 字位扩展：既扩展字长也扩展单元数
2. 译码器 74LS138

### 四、输入输出技术

1. I/O接口
2. 接口解决的问题：

速度匹配 → 数据的缓冲与暂存

信号的驱动能力 → 信号驱动

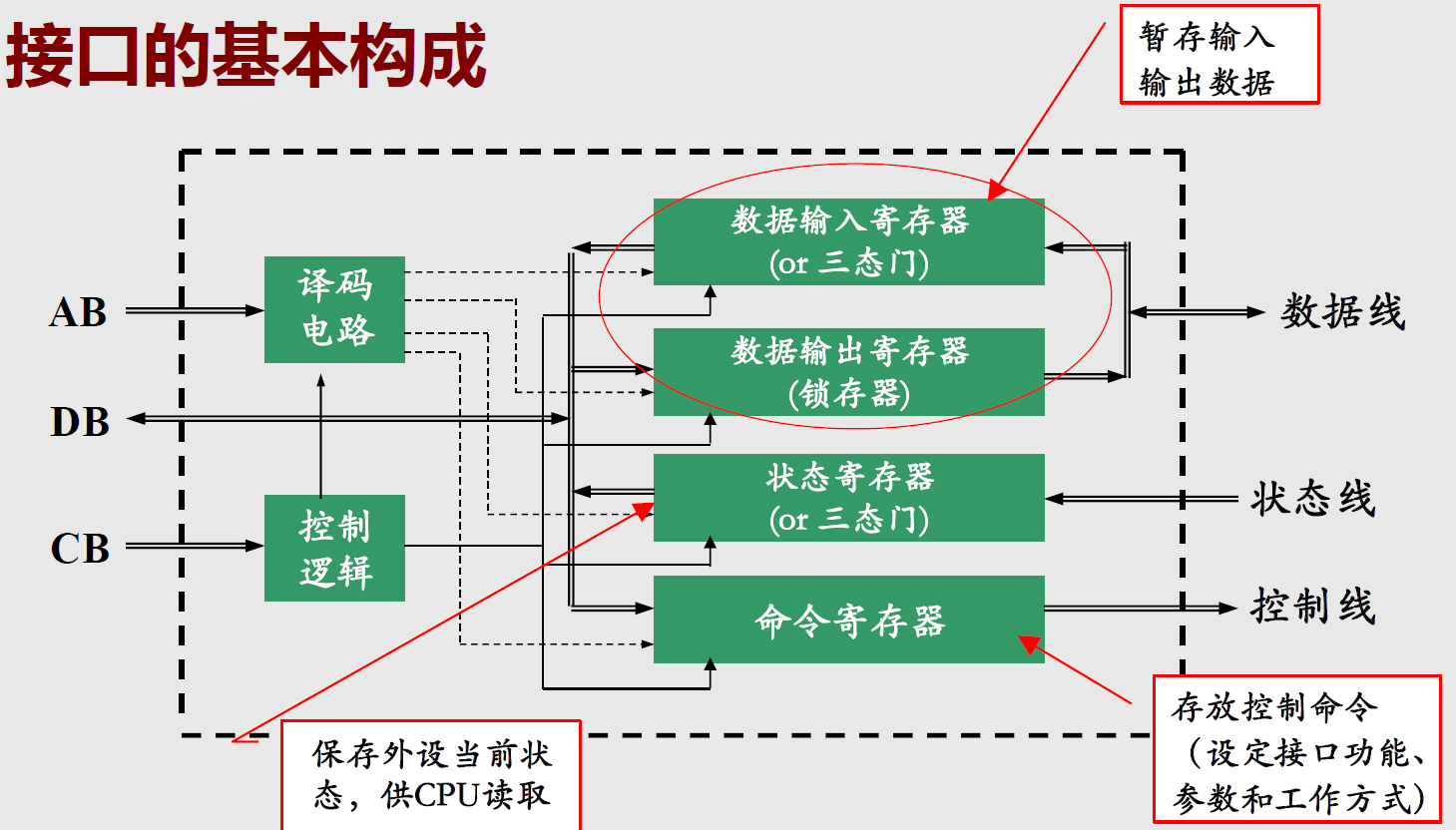
信号形式和点平的匹配 → 信号类型转换

信息格式 → 信号格式转换

时序匹配（定时关系）

总线隔离 → 三态门

1. 接口基本构成



1. 端口

数据端口（双向）、状态端口（输入端口）、控制端口（输出端口）

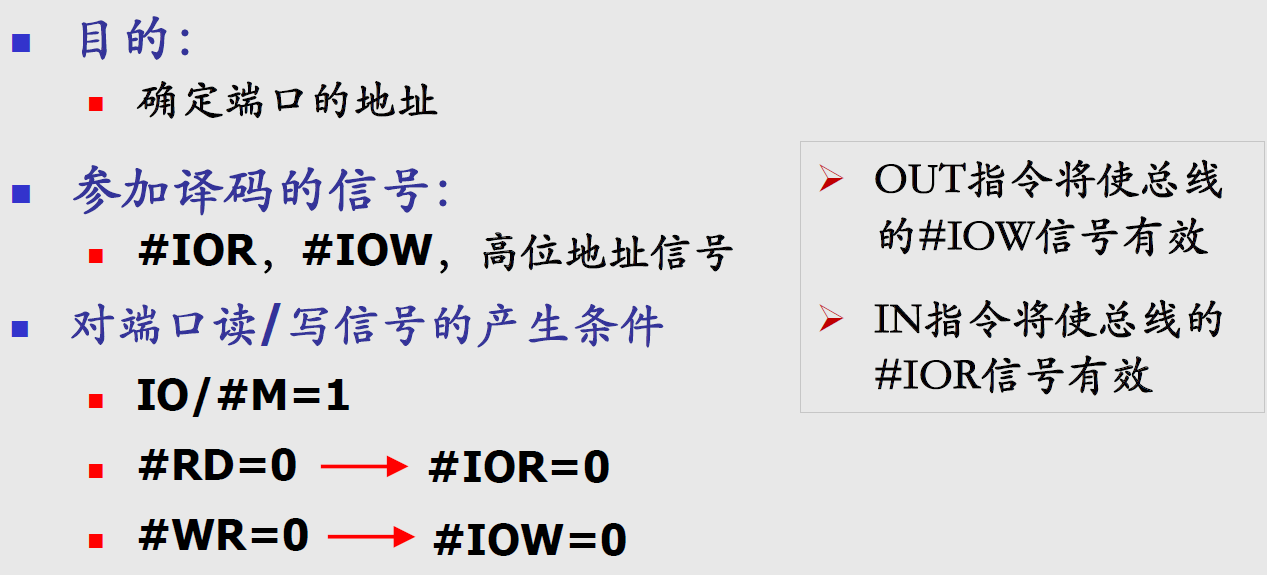
每个端口地址 = 片选地址（高位地址）+ 片内地址

端口的编址方式：与内存统一编址、独立编址

* 8088/8086寻址端口的能力：64K个端口

采用独立编址，通过IO/#M信号状态区分，当为低电平时只有A15~A0起作用

8088CPU(IBMPC)只适用了1024个I/O地址（0~3FFH）



【由于端口地址资源丰富，为了简化电路，接口地址译码通常采用部分地址译码。】

1. 简单接口电路
2. 分类

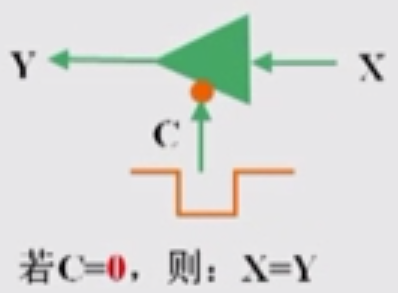
按传输信息的方向分类：输入接口、输出接口

按传输信息的类型分类：数字接口、模拟接口

按传输信息的方式分类：并行接口（硬盘和内存之间的传输）、串行接口

1. 输入和输出接口

* 输入接口：要求对数据具有控制能力。常用三态门实现。



三态门接口：示例 74LS244

* 输出接口：要求对数据具有锁存能力，重用锁存器实现，如D触发器

锁存器：

示例：74LS273：8D触发器，不具备数据的控制能力，上升沿有效【只能做输出接口】

74LS373：含三态的8D触发器，具有对数据的控制能力【可以做输出/输入接口】

1. 基本输入/输出方法

* 无条件传送

要求外设总是处于准备好状态。

优点：软件及接口硬件简单

缺点：只适用于简单外设，适用范围较窄。比如：开关

* 查询式传送

仅当条件满足时才能进行数据传送，每满足一次条件只能进行一次数据传送。

工作条件：外设应该提供设备状态信息，接口应该具备状态端口。

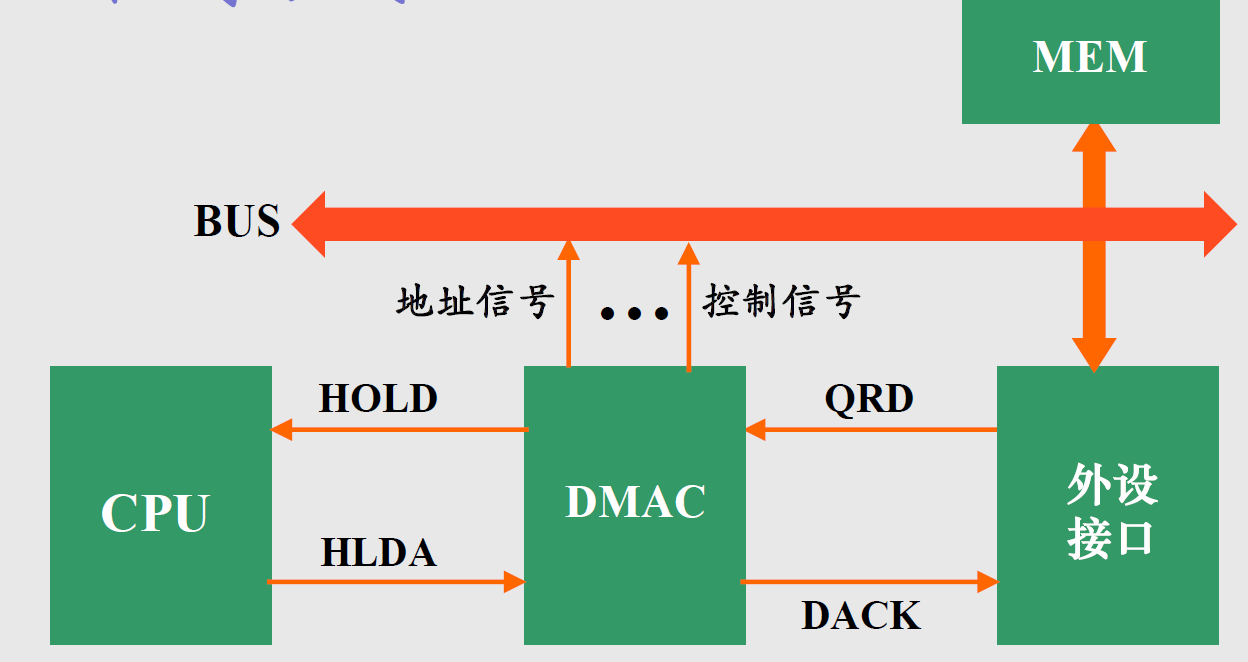
* 中断方式传送

特点：外设在需要时向CPU提出请求，CPU再去为它服务。服务结束后或外设不需要时，CPU可执行自己的程序。

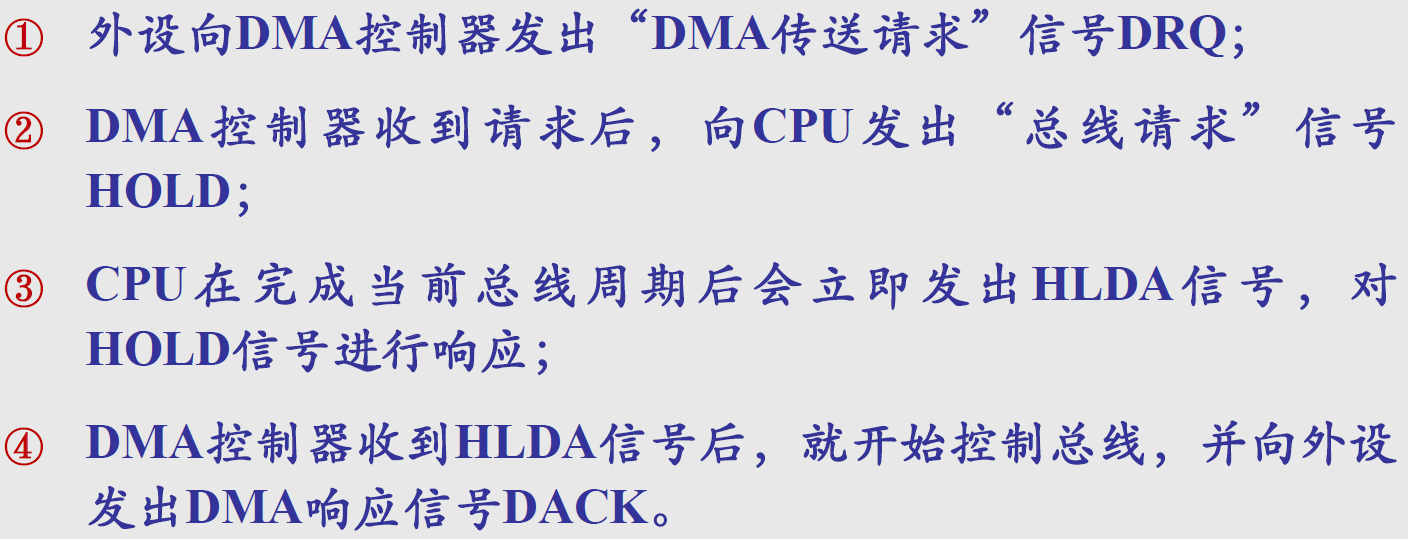
* 直接存储器存取（DMA）

外设通过接口直接与存储器进行数据交换，CPU不再担当数据传输的中介。

总线由DMA控制器进行控制（CPU临时放弃了总线控制权）。

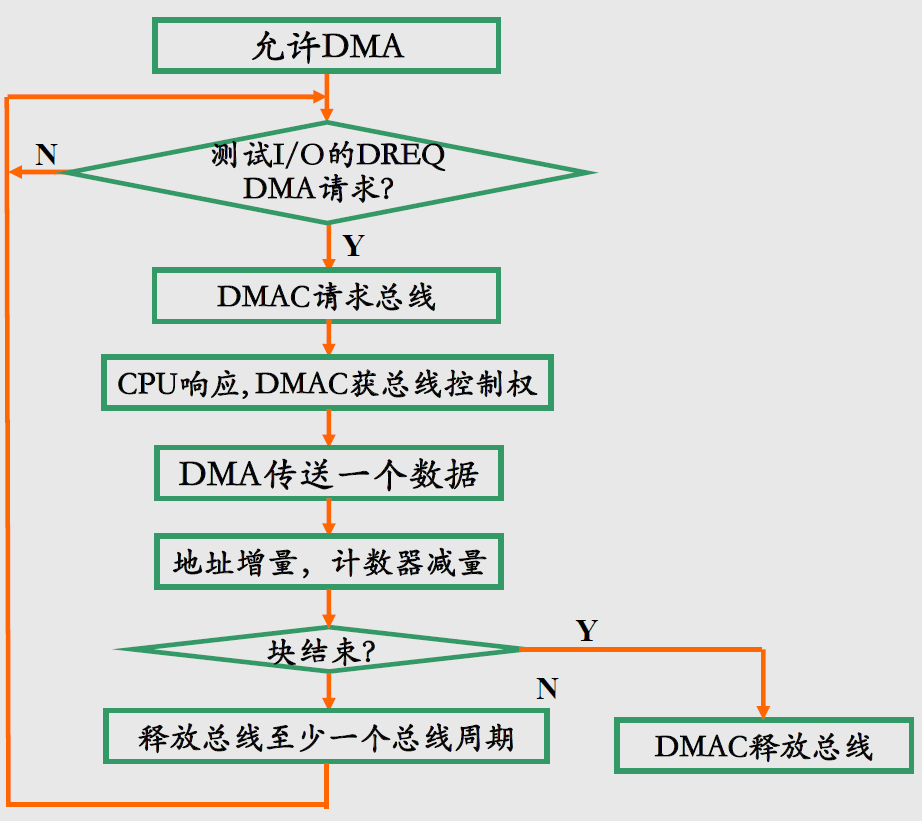


* DMA控制方式工作过程



* DMA工作方式

周期窃取：每个DMA周期只传送一个字节或一个字就立即释放总线。



数据块传送：DMAC在申请到总线后，将一块数据传送完后才释放总线，而不管中间DREQ是否有效。

直接存取方式：DMA的数据传送请求直接发到主存储器，在得到相应后，整个工作过程在DMA控制器中由硬件完成。

1. 中断技术
2. 引入中断的原因

提高对外设请求的响应实时性；提高CPU的利用率：避免了CPU不断检测外设状态的过程。

1. 分类

根据中断请求的来源分为：

内部中断：异常中断 → 异常事件引起

软件中断 → 中断指令引起（INT）

外部中断：可屏蔽中断 → INTR中断

非屏蔽中断 → NMI中断（上升沿有效）

1. 外部可屏蔽中断响应的一般过程

* 中断请求

外部可屏蔽中断请求信号：INTR

中断请求信号应保持到中断被处理为止；CPU响应中断后，中断请求信号应及时撤销。

* 中断源识别及中断判优（目前均由硬件实现）

软件查询法（效率较低）

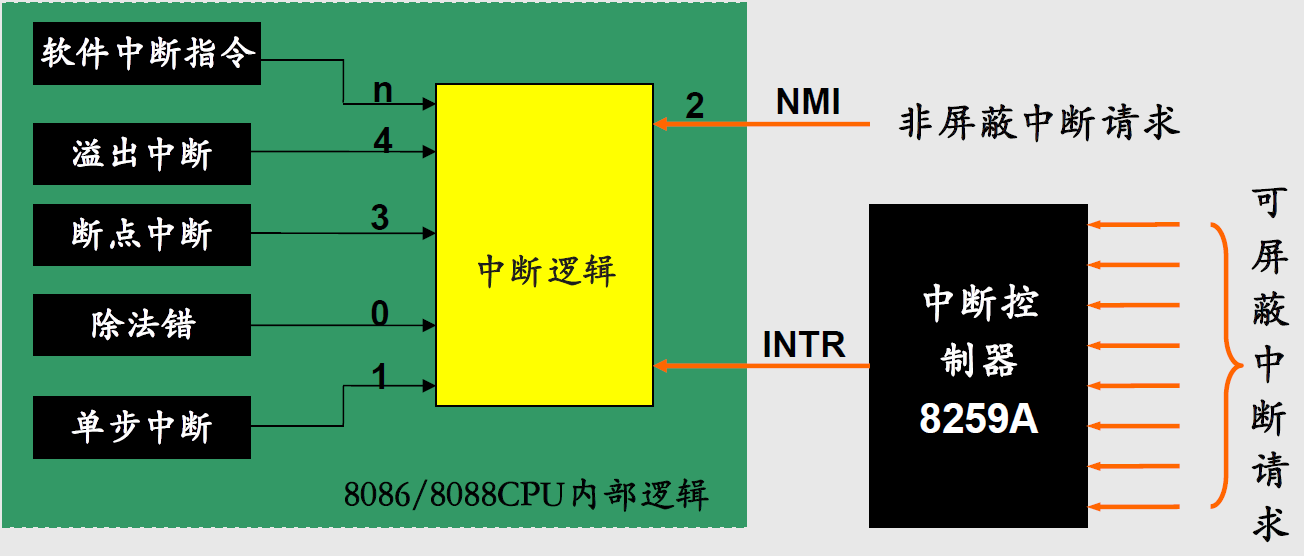
中断矢量法：由中断源提供中断类型号，CPU根据类型确定中断源。

优先级：陷阱中断 > 内部中断 > NMI中断 > 外部可屏蔽中断

中断判优：优先级法则、排队法则

链路法

中断控制器法（通过软件可以改变优先级）



* 中断响应

外部可屏蔽中断响应信号：#INTA

中断响应时会发出两个#INTA负脉冲，第一个负脉冲表示响应中断请求，第二负脉冲接受由中断控制器发来的中断类型码。



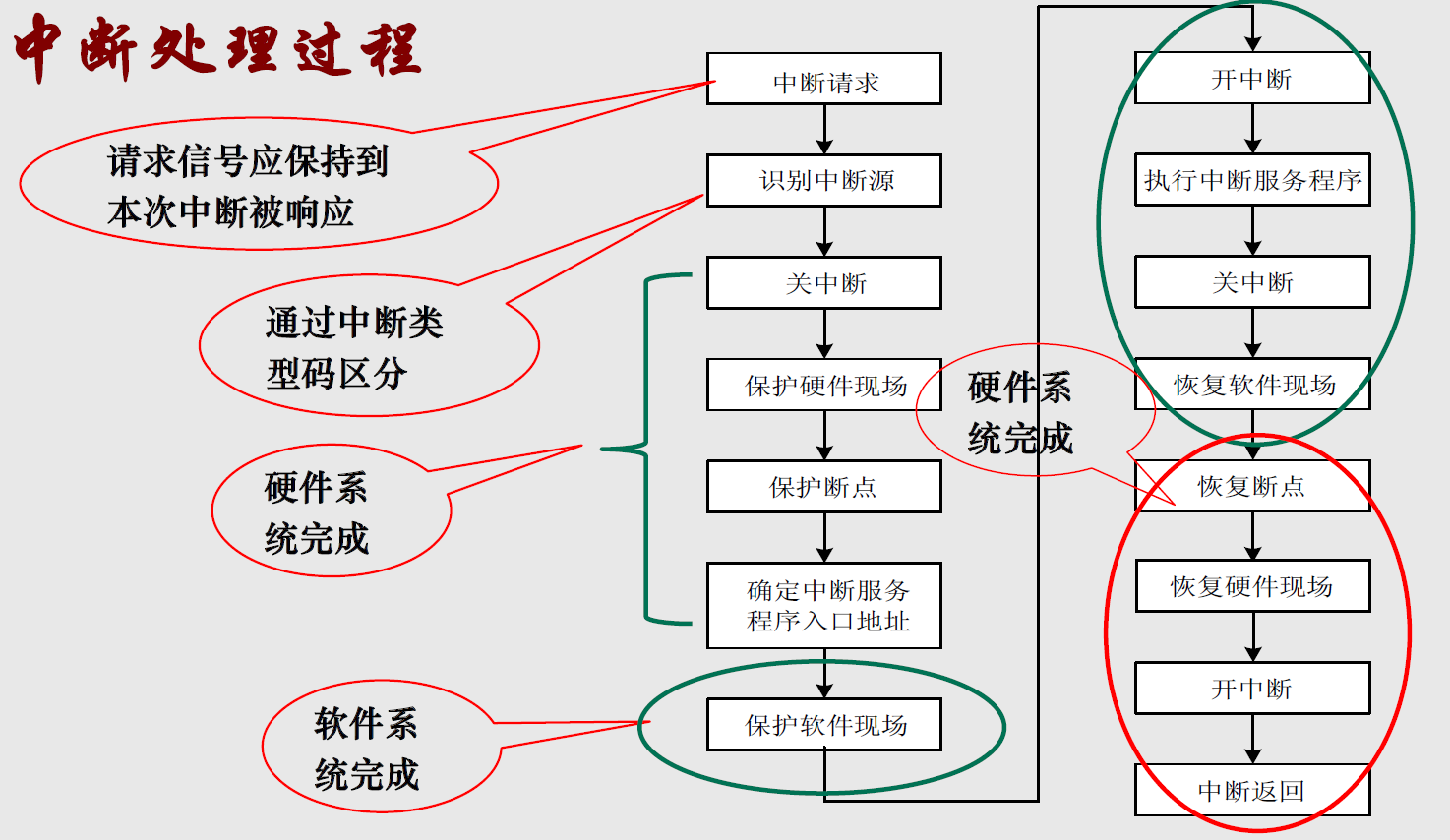
* 中断处理（服务）

执行中断服务子程序，中断服务子程序的特点：为“远地址”，用IRET指令返回。

保护软件现场（参数）→ 开中断（STI）→ 中断处理 → 关中断（CLI）→ 恢复现场 → 中断返回

* 中断返回

执行IRET指令，使IP、CS和FLAGS从堆栈弹出。



1. 中断向量表【实地址模式下内存起始1KB个内存空间】

存放各类中断的中断服务程序的入口地址，位于内存的00000H ~ 003FFH.

每个入口占用4B，低地址为段内偏移，高地址为段基地址。共管理256个中断向量。

1. 8088内部中断与NMI中断

特点：无#INTA周期；中断类型码固定由指令给出。