

1. 基本组成

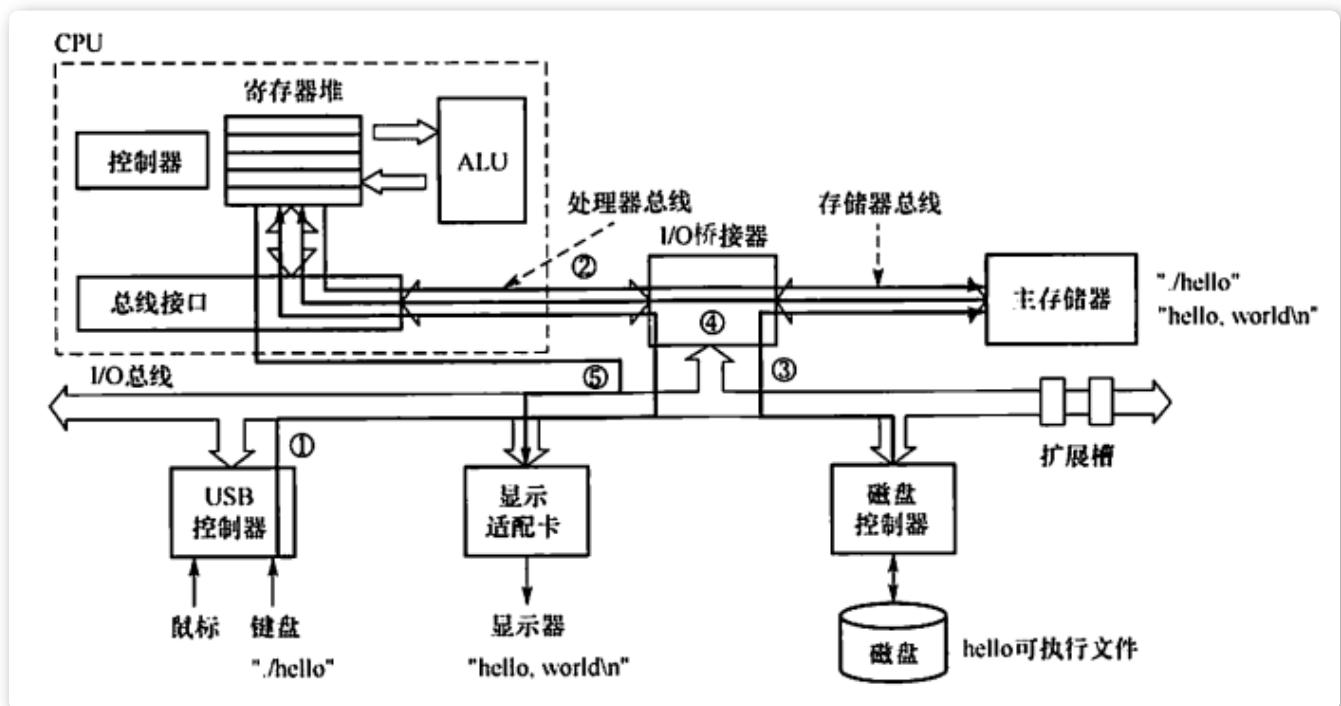
源程序到可执行文件

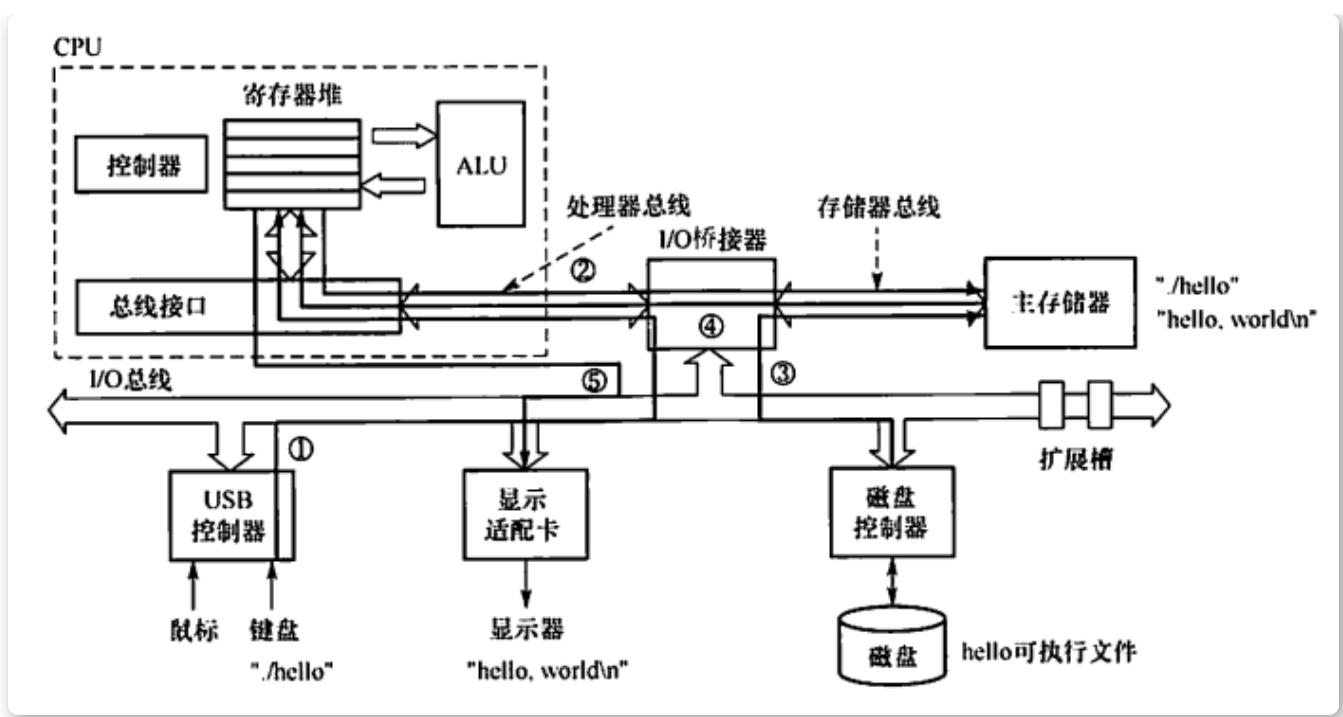
IR指令寄存器

CU控制单元

MAR 地址寄存器 存放访存地址 用于寻址 位数对于存储单元的个数 长度与PC长度相等

MDR 数据寄存器 暂存要从存储器中读或写的信息 位数与存储字长相等





指令和数据以同等地位存放在存储器里面，形式上无差别，只是在程序执行时具有不同的含义

指令按地址访问，数据由指令的地址码指出，除立即寻址外，数据均存在存储器内

在程序执行前，指令和数据需要预先存放在存储器中，中央处理器可以从存储器存取代码

计算机多层结构中，上下层是可以分割的，且上层是下层的实现功能实现，上层在下层的基础上实现了更加丰富的功能

编译程序是先完整的编译后运行程序，eg c ,c++ 解释程序是一句一句翻译且边翻译边执行的程序如 js,python

犹豫解释程序要边翻译成机器语言边执行，一般速度较编译程序慢

多处理器的工作方式是多指令多数据流方式

冯诺依曼计算机基本工作方式是控制流驱动方式 本质特征是 存储程序原理

地址译码器是主存的构成部分

层次化，模块化，规则性

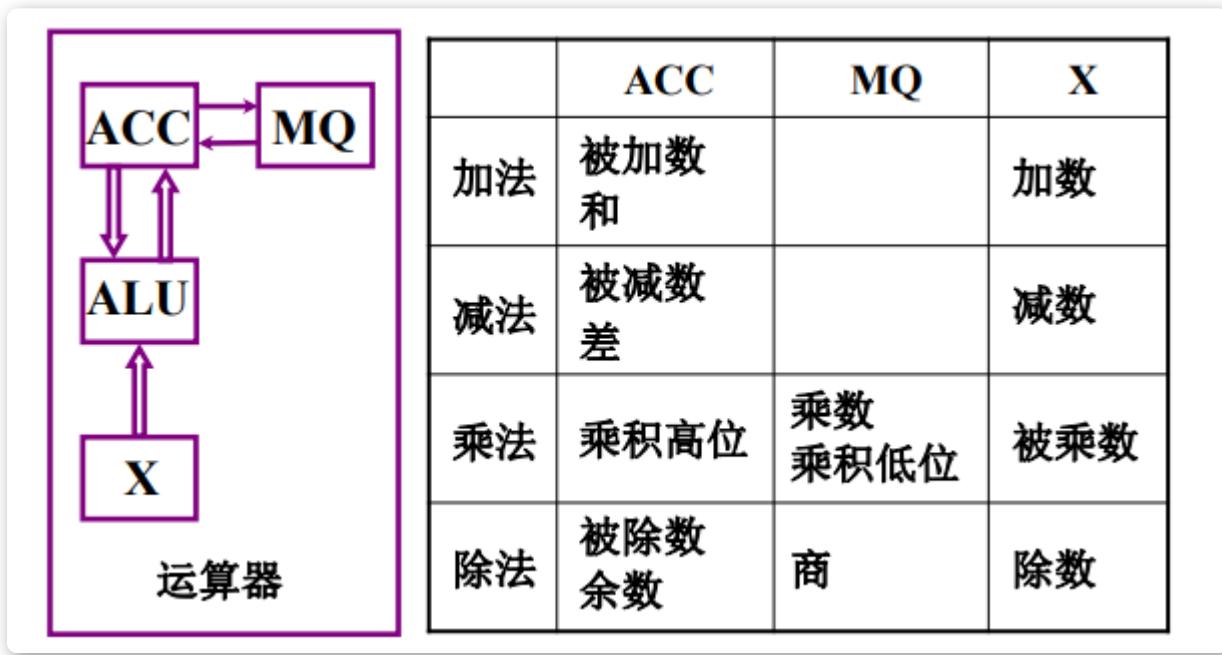
指令和数据都是保存在存储器中的

存储器基本组成

主存储器{存储体， MAR， MAR}

MAR 存储器地址寄存器（反应存储单元的个数）

MDR 存储器数据寄存器（反映存储字长）



存储体-存储单元-存储元件

存储单元 存放一串二进制代码

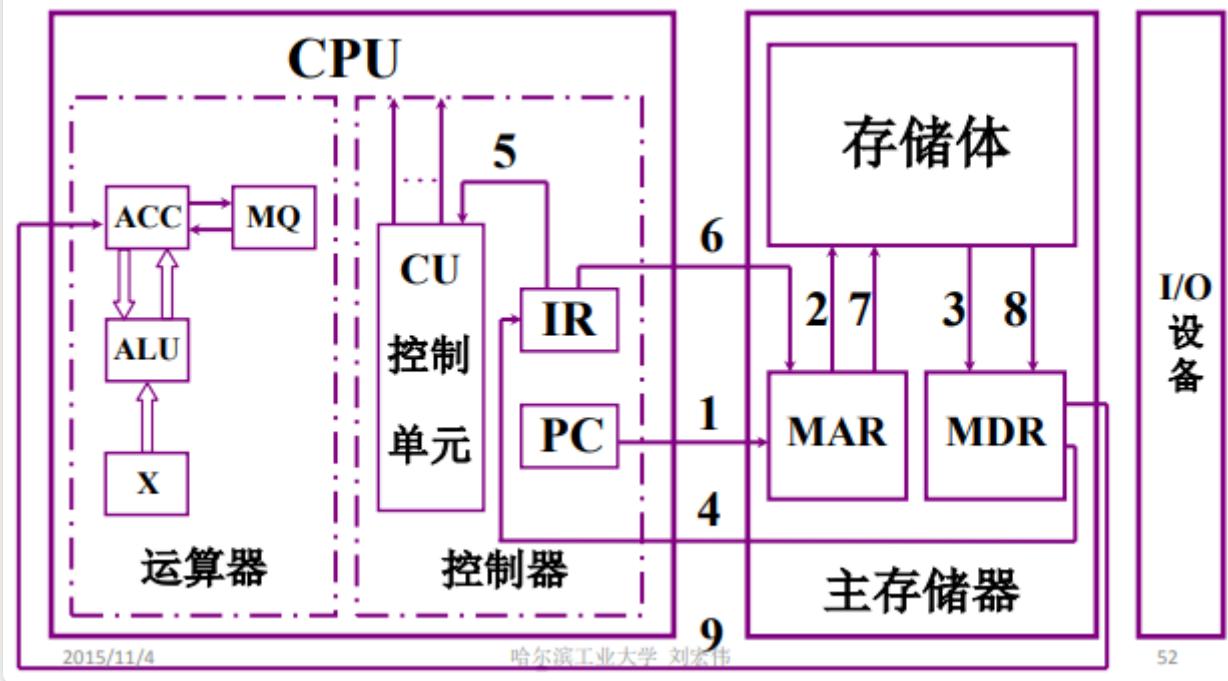
存储字 存储单元中的二进制代码组合

存储字长 存储单元中二进制的位数

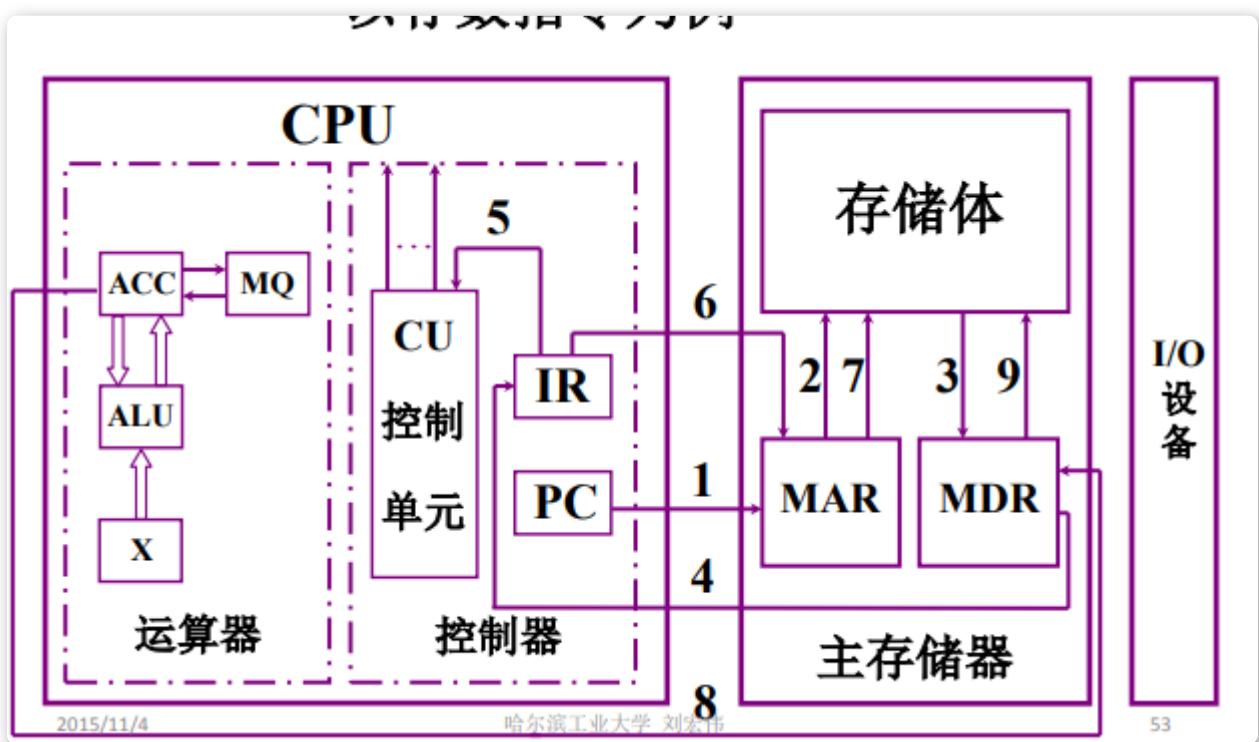
控制器功能：解释指令，保证指令按序执行

控制器基本组成 CU（执行指令） PC（取指令） IR（分析指令）

取数流程



存数流程



机器字长 CPU一次能处理数据的位数与CPU中的寄存器位数有关

运算速度

主频 核数（每个核支持的线程数） 吉普森算法

CPI 执行一条指令所需要的时钟周期数

MIPS 每秒执行百万条指令

FLOPS 每秒浮点运算次数

存储容量 存放二进制信息的总位数

存储容量

主存容量

辅存容量

存储单元个数 × 存储字长
如 MAR MDR 容量
 $10 \quad 8 \quad 1 \text{ K} \times 8\text{位}$
 $16 \quad 32 \quad 64 \text{ K} \times 32\text{位}$
字节数
如 $2^{13}\text{b} = 1 \text{ KB}$ $1\text{K} = 2^{10}$
 $2^{21}\text{b} = 256 \text{ KB}$ $1\text{B} = 2^3\text{b}$

字节数 80 GB
 $1\text{GB} = 2^{30}\text{B}$

2015/11/4 哈尔滨工业大学 刘宏伟 57

1.1. 电脑发展

2. 第二章

2.1. 字符

2.1.1. 进制

十进制到任意进制

整数部分 除基取余法

对整数部分的处理使用短除法，并从下向上，得1111011

2	123	余数
2	61	1
2	30	1
2	15	0
2	7	1
2	3	1
2	1	1
	0	1

乘基取整法（小数）

小数部分采用乘基取整的思路，如下所示：

$$\begin{array}{r} 0.6875 \\ * \quad \quad 2 \\ \hline 1.3750 \\ 0.3750 \\ * \quad \quad 2^{qquad} \\ \hline 0.7500^{qquad1} \quad \text{最高位} \\ * \quad \quad 2^{qquad0} \\ \hline 1.5000^{qquad1} \quad \text{最低位} \\ 0.5000^{qquad1} \\ * \quad \quad 2^{qquad\dots} \\ \hline 1.0000 \\ \dots \end{array}$$

一般十进制无法精确化为二进制小数，取一定精度即可。

2.2. 字符与字符串

ASCII码

- 32~126位称为可印刷字符
- A-Z, a-z是连续的，但是Z和a之间不连续
- 以字节的形式存储



例：已知'A'的ASCII码码值是65，H存放在某存储单元M中，求M中存放的值

$$65 + (8 - 1) = 72$$

$$72 = 0100\ 1000$$

字符串

字符串：IF_A>B THEN READ(C)，每个存储单元存放4B

大端模式：存储单元内先存储高位字节、后存储低位字节的顺序

小端模式：存储单元内先存储低位字节、后存储高位字节的顺序

2.2.1. 汉字编码

包括输入的输入编码，计算机内部处理的汉字内码和输出的汉字字形码三种。

2.2.2. 校验码

码距：任意两个合法码字之间不同的二进制的位数。

- 码距大于等于2时，开始具有检错能力
- 码距越大，纠错、检错能力越强

2.2.3. 奇偶校验码

添加一个校验位，校验位的取值满足以下规定：

奇校验码：整个校验码（有效信息位和校验位）中“1”的个数为奇数。

偶校验码：整个校验码（有效信息位和校验位）中“1”的个数为偶数。

即在有效信息的前面添加1或者0即可。

2.2.4. 海明校验码

- 若信息位的位数为n，海明码的位数为k，则有：

(1) $2k \geq n+k+1$

- 将校验位依次放在位于 $2i-1$ 位上，并将信息位按照顺序填入剩下的位置
- 海明码的取值相当于将各数据位的第i组的所有位求异或
- 每个校验组分别用校验位和参与计算校验位的信息位进行奇偶校验，并得到响应结果。

例：求1010的海明码

确定位数

因为 $n=4$ ，则由 (1) 计算得 $k=3$

则，设信息位

$$D_4 D_3 D_2 D_1 (1010)$$

，共4位，校验位

$$P_3 P_2 P_1$$

，共3位，对应的海明码为

$$H_7 H_6 H_5 H_4 H_3 H_2 H_1$$

确定校验位的分布

- 将校验码放在 $2i-1$ 位置上

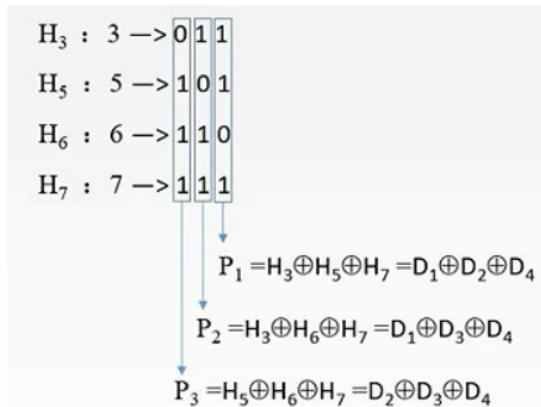
H7	H6	H5	H4	H3	H2	H1
			P3		P2	P1

- 将信息位填入剩下的位置

H7	H6	H5	H4	H3	H2	H1
D4	D3	D2	P3	D1	P2	P1
1	0	1		0		

求校验位的值

先将信息位的海明位号表示为二进制，再对相应组含有1的求异或



$$P_1 = D_1 \oplus D_2 \oplus D_4 = 0 \oplus 1 \oplus 1 = 0 \\ P_2 = D_1 \oplus D_3 \oplus D_4 = 0 \oplus 0 \oplus 1 = 1 \\ P_3 = D_2 \oplus D_3 \oplus D_4 = 1 \oplus 0 \oplus 1 = 0$$

因此有

H_7	H_6	H_5	H_4	H_3	H_2	H_1
D_4	D_3	D_2	P_3	D_1	P_2	P_1
1	0	1	0	0	1	0

纠错

将校验码和与之相应的数据位进行异或，应当全部为0。

$$P_1 \oplus D_1 \oplus D_2 \oplus D_4 = 0 \\ P_2 \oplus D_1 \oplus D_3 \oplus D_4 = 0 \\ P_3 \oplus D_2 \oplus D_3 \oplus D_4 = 0$$

得1则说明有错。

循环冗余校验

2.3. 定点数和有符号数

无符号

有符号数

原码

最高位表示符号位

补码

- 正数的补码：补码与原码相同
- 负数的补码：原码符号位不变，数值位取反，末位加1
- 当一个正数和一个负数互为补数时，他们的绝对值之和称为模数，类似于时钟， $+2 = -10$ ，12为模数

反码

正数的反码：反码与原码相同

负数的反码：原码符号位不变，数值位取反

移码

在真值X上加上一个常数（偏置值）

- 同一个数，补码和移码只有符号位相反
- 移码大真值就大，移码小真值就小
 - 移码全为0： -2^n
 - 移码全为1： $2^n - 1$
- 移码的0唯一

2.4. 定点数的运算

移位

算术移位

机器码采用有符号数，符号位不参与

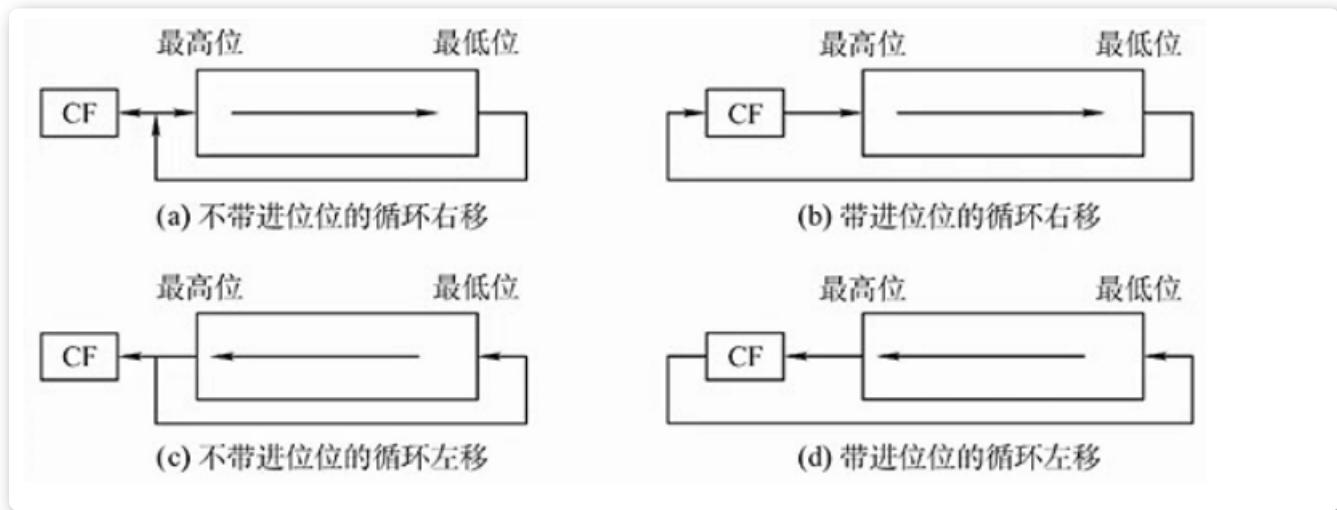
- 正数
 - 空位补0
- 负数
 - 原码：0
 - 补码：
 - 左移：0
 - 右移：1

- 反码：1

逻辑移位

机器码采用无符号数，直接补0。

循环移位



加减法

补码直接相加

例：设机器字长为8位（含一位符号位）， $A=15$, $B=-24$, 求 $\$[A+B]\{补\} \$\$$ 和 $\$[A-B]\{补\} \$\$$

先求原码：

$$A = +1111 = 0,1111 = 0,0001111 \quad B = -11000 = 1,11000 = 1,0011000$$

则可得补码：

$$A = 0,0001111 \quad B = 1,1101000$$

另外 $\$[-B]\{补\} = [B]\{补\} \$\$$ 连同符号位一起取反再+1

可得结果：

$$[A+B]\{补\} = [A]\{补\} + [B]\{补\} = 0,0001111 + 1,1101000 = 1,1110111 \quad [A-B]\{补\} = [A]\{补\} + [-B]\{补\} = 0,0001111 + 0,0011000 = 0,0100111$$

溢出的概念和判断方式

大于机器所能表示的最大正数称为上溢；

小于机器所能表示的最小负数称为下溢。

当两个符号相同的数相加或者两个符号相异的数相减时才会发生溢出。

源码一位乘法

补码Booth乘法

定点数除法

2.5. 浮点数表示和运算

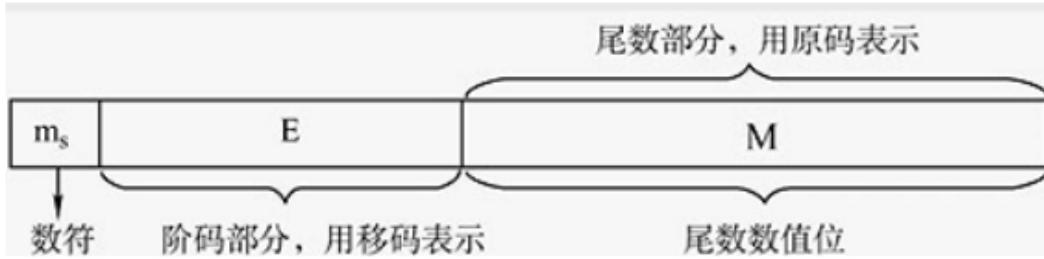
$$N = \text{power}(r, e) * M$$

r为阶码的底，通常为2

e为阶码 反应浮点数表示范围和小数点的实际位置，用补码或者移码表示

m 尾数其位数反映小数的精度 用原码或者补码表示

IEEE 754标准



类型	数阶	阶码	尾数数值	总位数	偏置值
短浮点数	1	8	23	32	7FH (127)
长浮点数	1	11	52	64	3FFH (1023)
临时浮点数	1	15	64	80	3FFFH

2.6. 浮点数的运算

加减法

对阶，尾数加减，规格化，舍入，判溢出

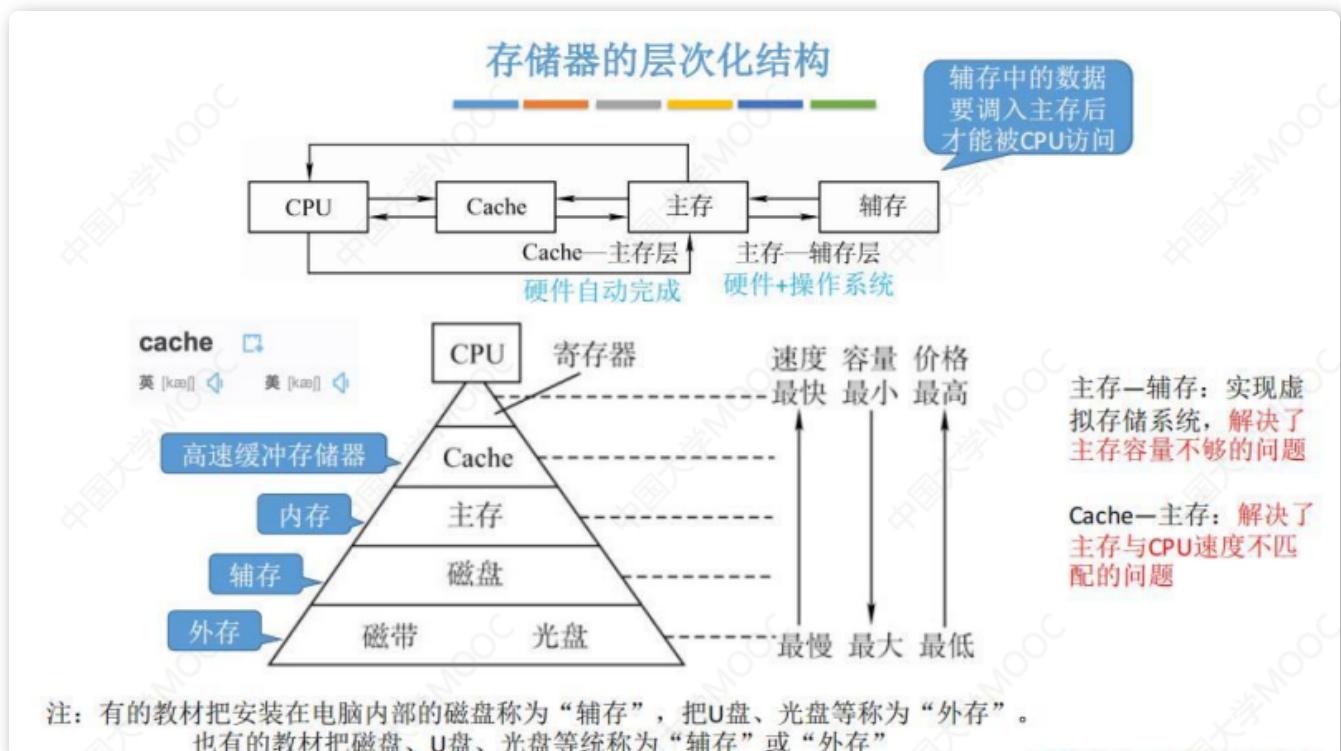
- 有符号数

- OF: 溢出标志位
 - 最高位产生的进位与次高位产生的进位异或
 - 判断有符号数的溢出
- SF: 符号标志位
 - 取运算结果的最高位

- 无符号数

- ZF: 零标志位
 - 运算结果全部为零
- CF: 进位标志
 - 最高位的进位信号与sub进行异或
 - 为1代表发生了借位，产生溢出
 - 判断无符号数的溢出

3. 存储器



存储器的分类--层次

高速缓存cache，主存储器（主存，内存），辅助存储器（辅存，外存）

CAM 相联存储器” 快表 “就是相联存储器

RAM随机存储器

串行访问存储器：读写某个存储单元所需时间与存储单元物理位置无关

SAM顺序存储器

DAM直接存取存储器 既有随机存取特性也有顺序存取特性，先直接选取信息所在区域如何顺序方式存取

读写存储器和只读存储器ROM

信息可保存性

断电后消失的存储器 易失存储器（主存，cache）

断电后存储信息依然保持的存储器 非易失性存储器（磁盘，光盘）

信息读出后，原存储信息被破坏——破坏性读出（如DRAM芯片，读出数据后要进行重写）

信息读出后，原存储信息不被破坏——非破坏性读出（如SRAM芯片、磁盘、光盘）

存储器性能指标

存储容量：存储字数*字长（如1M*8位） MDR位数反应存储字长

单位成本 每位价格=总成本/总容量

存储速度：数据传输率=数据的宽度/存储周期 数据的宽度即存储字长

- ①存取时间(T_a) :存取时间是指从启动一次存储器操作到完成该操作所经历的时间，分为读出时间和写入时间。
- ②存取周期(T_m):存取周期又称为读写周期或访问周期。它是指存储器进行一次完整的读写操作所需的全部时间，即连续两次独立地访问存储器操作(读或写操作) 之间所需的最短时间间隔。
- 主存带宽 (B_m) :主存带宽又称数据传输率，表示每秒从主存进出信息的最大数量，单位为字/秒、字节/秒(B/s) 或位/秒 (b/s) 。

总容量等于 存储单元的个数*存储字长

寻址 字节寻址

按字寻址

按半字寻址

按双字寻址

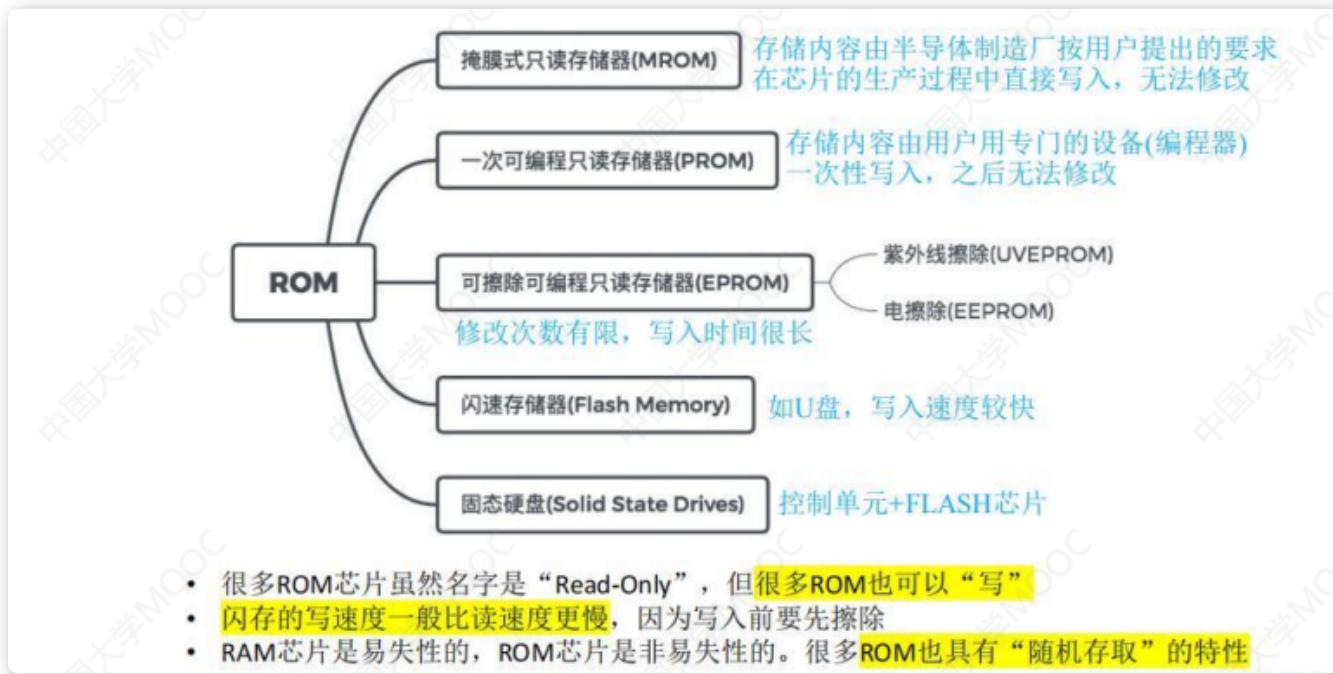
3.1. 内存

DRAM (动态) 和SRAM (静态)

区别存储元不一样，前者是栅极电容，后者是双稳态触发器

DRAM v.s SRAM		
	Static Random Access Memory	Dynamic Random Access Memory
类型特点	SRAM (静态RAM)	DRAM (动态RAM)
存储信息	触发器	电容
破坏性读出	非	是
读出后需要重写? (再生)	不用	需要
运行速度	快	慢
集成度	低	高
发热量	大	小
存储成本	高	低
易失/非易失性存储器?	易失 (断电后信息消失)	易失 (断电后信息消失)
需要“刷新”?	不需要	需要
送行列地址	同时送	分两次送
	常用作Cache	常用作主存

DRAM的刷新



增加主存的存储字长-位扩展

字扩展

线选法	译码片选法
n 条线 $\rightarrow n$ 个选片信号	n 条线 $\rightarrow 2^n$ 个选片信号
电路简单	电路复杂
地址空间不连续	地址空间可连续

MROM 掩模式只读存储器，生产过程直接写入，之后任何人不可重写

PROM 可编程只读存储器 写入一次后不可写入

EPROM 可进行多长重写

Flash Memory 闪速存储器 可进行多长快速擦除重写

SSD 固态硬盘

逻辑上 主存由RAM+ROM组成且二者常统一编址

线选法 n 条线 $\rightarrow n$ 个选片信号 电路简单 地址空间不连续

译码片选法 n 条线 $\rightarrow 2^n$ 个选片信号 地址空间连续

字位同时扩展

双端口RAM

优化多核CPU访问一根内存条的速度

需要有两组完全独立的数据线，地址线，控制线,CPU, RAM中也要有更复杂的控制电路

发生冲突 发出BUSY信号，其中一个CPU的访问端口暂时关闭

多体并行存储器

高位交叉编址的多体存储器 体号|体内地址

理论上多个存储体并行访问，但是由于连续访问相当于扩容

低位交叉编址的多体存储器 体内地址|体号

当存储模块数 $m \geq T/r$ 时，可使流水线不间断，每个存储周期内可读写地址连续的 m 个字
微观上， m 个模块被串行访问；宏观上，每个存储周期内所有模块被并行访问

存储周期 T 存取时间 r

存储周期 T 总线传输周期为 r

3.2. 外存

外存储器

外存储器

计算机的外存储器又称为辅助存储器，目前主要使用磁表面存储器。

所谓“磁表面存储”，是指把某些磁性材料薄薄地涂在金属铝或塑料表面上作为载磁体来存储信息。磁盘存储器、磁带存储器和磁鼓存储器均属于磁表面存储器。

磁表面存储器的优点：

- ① 存储容量大，位价格低；
- ② 记录介质可以重复使用；
- ③ 记录信息可以长期保存而不丢失，甚至可以脱机存档；
- ④ 非破坏性读出，读出时不需要再生。

磁表面存储器的缺点：

- ① 存取速度慢；
- ② 机械结构复杂；
- ③ 对工作环境要求较高。

原理：当磁头和磁性记录介质有相对运动时，通过电磁转换完成读/写操作。

编码方法：按某种方案（规律），把一连串的二进制信息转换成存储介质磁层中一个磁化翻转状态的序列，并使读/写控制电路容易、可靠地实现转换。

磁记录方式：通常采用调频制（FM）和改进型调频制（MFM）的记录方式。

外存储器既可以作为输入设备，也可以作为输出设备。
(既可以存数据，也可以读数据)

磁盘存储器

1. 磁盘设备的组成**① 存储区域**

一块硬盘含有若干个记录面，每个记录面划分为若干条磁道，而每条磁道又划分为若干个扇区，扇区（也称块）是磁盘读写的最小单位，也就是说磁盘按块存取。

磁头数 (Heads)

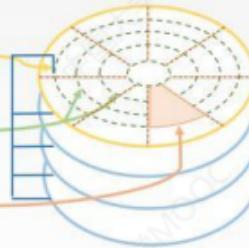
即记录面数，表示硬盘总共有多少个磁头，磁头用于读取/写入盘片上记录面的信息，一个记录面对应一个磁头。

柱面数 (Cylinders)

表示硬盘每一面盘片上有多少条磁道。在一个盘组中，不同记录面的相同编号（位置）的诸磁道构成一个圆柱面。

扇区数 (Sectors)

表示每一条磁道上有多少个扇区。

**② 硬盘存储器**

硬盘存储器由磁盘驱动器、磁盘控制器和盘片组成。

磁盘驱动器：核心部件是磁头组件和盘片组件，温彻斯特盘是一种可移动头固定盘片的硬盘存储器。

磁盘控制器：是硬盘存储器和主机的接口，主流的标准有IDE、SCSI、SATA等。

性能指标

1 磁盘容量：一个磁盘所能存储的字节总数

非格式化容量是指磁记录表面可以利用的磁化单元总数

格式化容量指按照某种特定的记录格式所能存储信息的总量

2 记录密度：盘片单位面积记录的二进制的信息量

道密度是沿磁盘半径方向单位长度上的磁道数

位密度是磁道单位长度上能记录的二进制代码位数

面密度是位密度和道密度的乘积

3 平均存取时间

平均存取时间 = 寻道时间（磁头移动到目的磁道）+ 旋转延迟时间（磁头定位到所在扇区的）
+ 传输时间（传输数据所花费的时间）

4 数据传输率

磁盘存储器在单位时间内向主机传送数据的字节数

磁盘地址

主机向磁盘控制器发送寻址信息

驱动器号 | 柱面 (磁道) 号 | 盘面号 | 扇区号 |

多个磁盘 移动磁头臂 (寻道) 激活某个磁头 通过旋转将特定扇区划过磁头下方

硬盘的主要操作是寻址、读盘、写盘。每个操作都对应一个控制字，硬盘工作时，第一步是取控制字，第二步是执行控制字。

硬盘属于机械式部件，其读写操作是串行，不可能在同一时刻既读又写，也不可能在同一时刻读两组数据或写两组数据。

RAID (Redundant Array of Inexpensive Disks, 廉价冗余磁盘阵列) 是将多个独立的物理磁盘组成一个独立的逻辑盘，数据在多个物理盘上分割交叉存储、并行访问，具有更好的存储性能、可靠性和安全性。

RAID的分级如下所示。在RAID1~R中，无论何时有磁盘损坏，都可以磁盘再插入好的磁盘，而数据不会

RAIDO：无冗余和无校验的磁

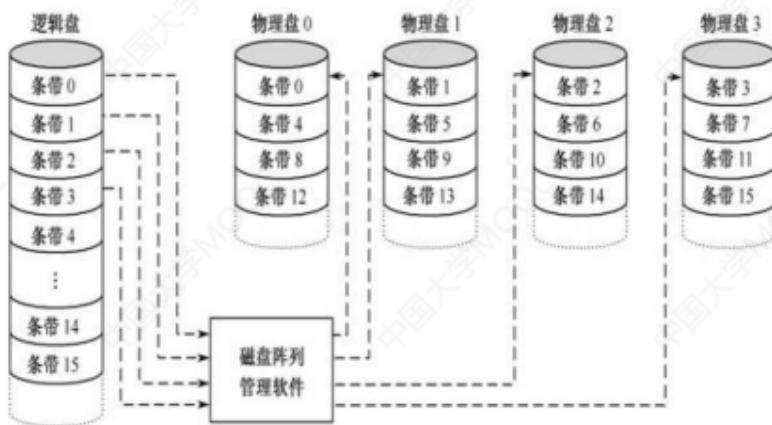
RAID1：镜像磁盘阵列。

RAID2：采用纠错的海明码的

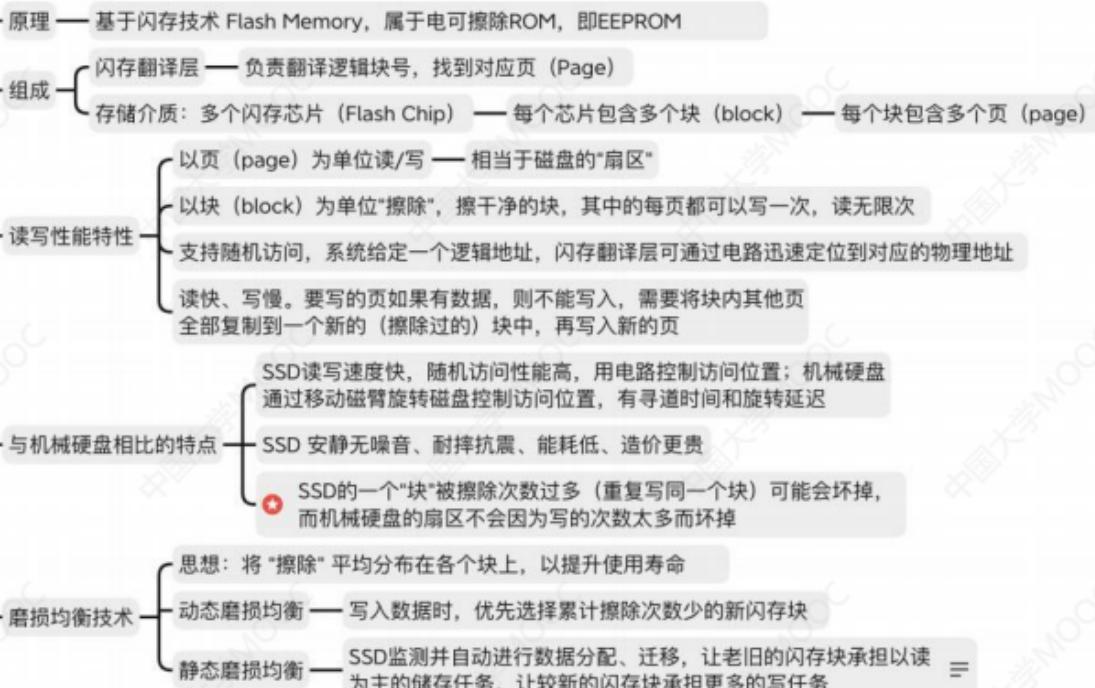
RAID3：位交叉奇偶校验的磁

RAID4：块交叉奇偶校验的磁

RAID5：无独立校验的奇偶校



固态硬盘SSD



3.3. Cache

工作原理 (实际上，Cache被集成在CPU内部用SRAM实现，速度快，成本高)

时间局部性 在最近的未来要用到的信息，很可能是现在正在使用到的信息

空间局部性 在最近未来要用到的信息（指令和数据），很可能鱼现在在使用的信息存储空间上是邻近的

tc为访问Cache的时间 tm为访问主存的时间

命中率为H

缺失率（未命中率） $M = 1-H$

平均访问时间 $t = Htc + (1-H)(tc+tm)$ (Cache未命中则访问主存)

同时访问 $t = Htc + (1-H)tm$

主存与Cache之间以块位单位进行数据交换

操作系统中，通常将主存中的一个块也称为一个页/页面/页框

Cache中的块也称行

3.3.1. Cache映射关系

全相联映射

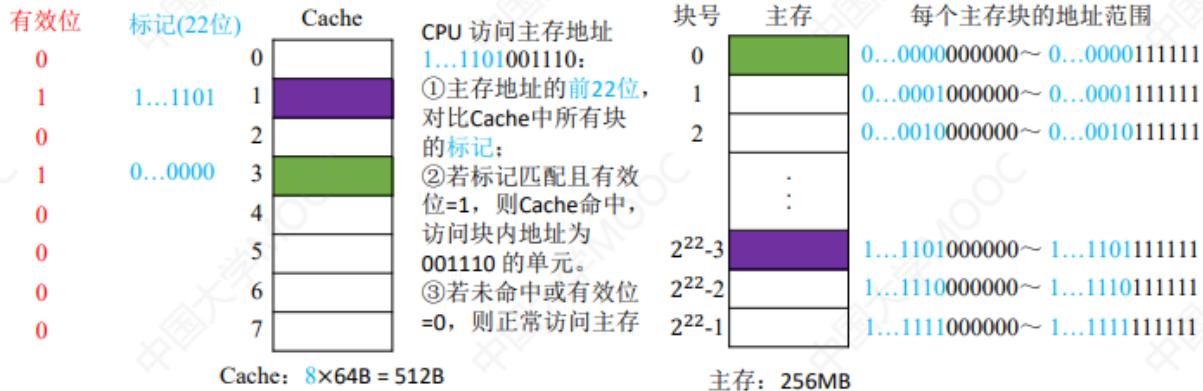
主存块可以放在Cache的任意位置

“全相联映射”如何访存?

假设某个计算机的主存地址空间大小为256MB，按字节编址，其数据Cache有8个Cache行，行长为64B。
即Cache块，与主存块的大小相等

$256M = 2^{28}$ 主存的地址共28位：

主存块号	块内地址
22位	6位



直接映射

每个主存块只能放到一个特定的位置， Cache块号 = 主存块号%Cache总块数

若Cache总块数=2的n次方 则主存块号末尾n位直接反映它在cache中的位置

将主存块号的其余位作为标记即可

组相联映射

所属分组=主存块号%分组数

n路组相联映射 --每n个cache行为一组

2路组相联映射--2块位一组，分为4组

“组相联映射”如何访存

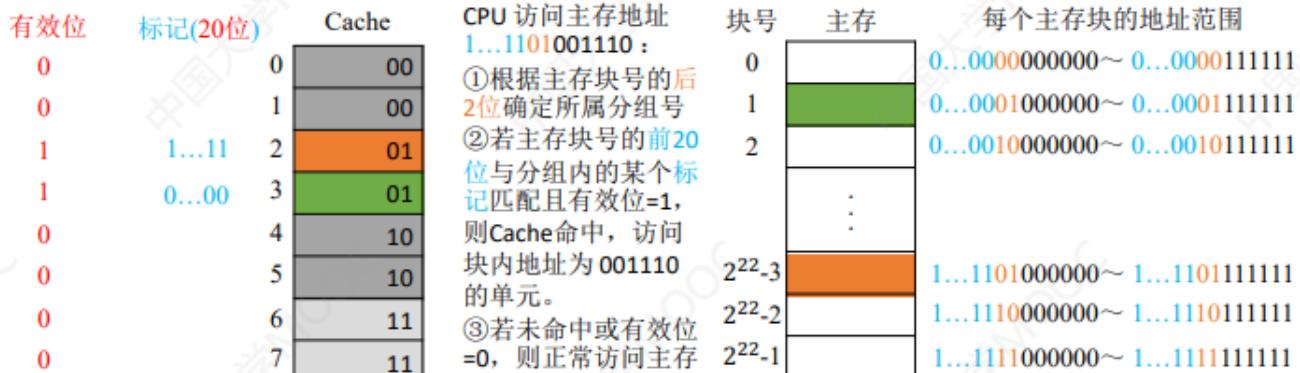
假设某个计算机的主存地址空间大小为256MB，按字节编址，其数据Cache有8个Cache行，行长为64B。

组相联映射，所属分组=主存块号%分组数

$256M = 2^{28}$ 主存的地址共28位：

主存块号	块内地址
22位	6位
20位 标记	2位 组号
	6位块内 地址

Cache 分为 2^2 组



Cache替换算法

随机算法RAND

先进先出算法FIFO

替换最先被调入cache块

近期最少使用LRU

为每个Cache设置一个计数器，用于记录每个Cache块已经有多久没有被访问

最近不经常使用LFU



写回法

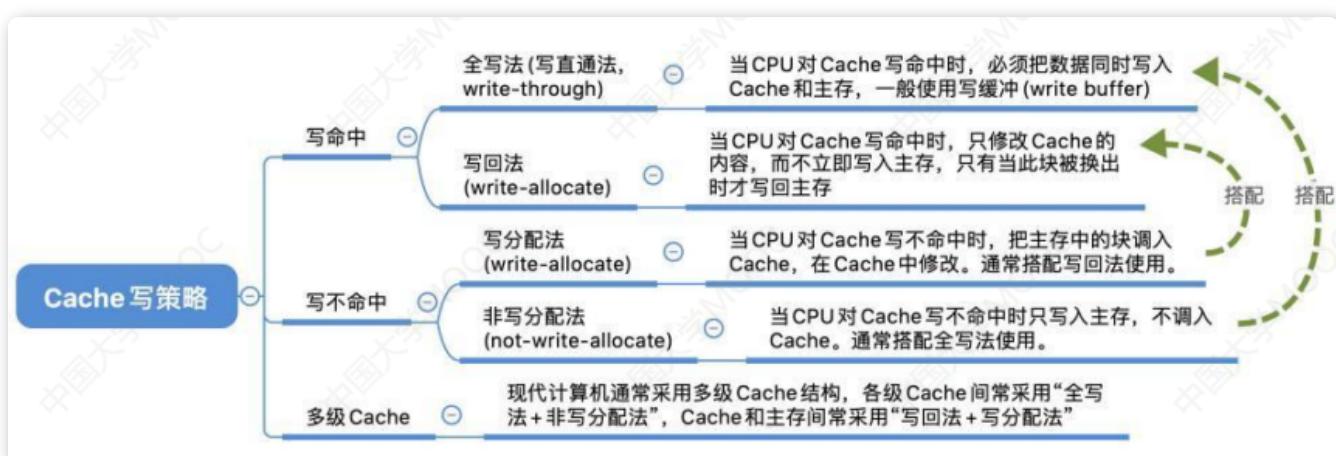
当CPU对Cache写命中时，只修改Cache的内容，而不立即写入主存，只有当此块被换出时才写回主存

减少了访存次数，但存在数据不一致的隐患

全写法，当CPU对Cache写命中时，必须把数据同时写入Cache和主存，一般使用写缓冲

访存次数增加，速度变慢，但能保证数据一致性

多级Cache



4. 指令系统

指令

x86架构 arm架构

一条指令就是机器语言的一个语句

操作嘛 (op) |地址码 (A)

一条指令可以包含0, 1, 2, 3, 4.....个地址码

零地址指令 OP

1, 不需要操作数, 如空操作, 停机, 关中断等指令

2, 堆栈计算机, 两个操作数隐含存放在栈顶和次栈顶, 结果压会栈顶

一地址指令 OP|A1

1.只需要单操作数, 如加1、减1、取反、求补等

指令含义: $OP(A1) \rightarrow A1$, 完成一条指令需要3次访存: 取指→读A1→写A1

2.需要两个操作数, 但其中一个操作数隐含在某个寄存器 (如隐含在ACC)

指令含义: $(ACC)OP(A1) \rightarrow ACC$

完成一条指令需要2次访存: 取指→读A1

A1表示主存地址 (A1)表示A1所指向的地址中的内容

二地址指令 OP|A1 (目的操作数) |A2 (源操作数)

常用于需要两个操作数的算算运算, 逻辑运算相关指令

完成一条指令需要访存4次 取指->读A1->读A2->写A1

三地址指令 OP|A1|A2|A3 (结果)

常用于需要两个操作数的算术运算，逻辑运算相关指令

完成一条需要访存4次

四地址指令 op|A1|A2|A3(结果)|A4 (下址)

完成需要访存4次 取指->读A1->读A2->写A3

正常情况下：取指令之后PC+1，指向下一条指令

四地址指令：执行指令后，将PC的值修改为A4所指的地址

指令字长：一条指令的总长度（可能会变）

机器字长：CPU进行一次整数运算所能处理的二进制数据的位数（通常和ALU直接相关）

存储字长：一个存储单元中的二进制代码位数（通常和MDR位数相同）

半长指令，单字长指令，双字长指令

定长指令字结构

变长指令字结构

定长操作码：指令系统中所有指令的操作码长度都相同

n位就是pow (2, n) 条指令

可变长操作码：指令系统中个指令的操作码长度可变

1. 数据传送	源	目的	
LOAD	作用：把 存储器 中的数据放到 寄存器 中		数据传送类：进行主存与CPU之间的数据传送
STORE	作用：把 寄存器 中的数据放到 存储器 中		
2. 算术逻辑操作			
	算术：加、减、乘、除、增1、减1、求补、浮点运算、十进制运算		
	逻辑：与、或、非、异或、位操作、位测试、位清除、位求反		运算类
3. 移位操作			
	算术移位、逻辑移位、循环移位(带进位和不带进位)		
4. 转移操作			程序控制类：改变程序执行的顺序
	无条件转移 JMP		
	条件转移 JZ: 结果为0; JO: 结果溢出; JC: 结果有进位		
	调用和返回 CALL和RETURN		
	陷阱(Trap)与陷阱指令		
5. 输入输出操作			输入输出类 (I/O)：进行CPU和I/O设备之间的数据传送
	CPU寄存器与IO端口之间的数据传送(端口即IO接口中的寄存器)		

4.1. 扩展操作码

在设计扩展操作码时

不允许短码是长码的前缀，即短操作码不能与长操作码的前面部分代码相同

各指令的操作码一定不能重复

地址长度位n，上一层留出m种状态，下一层可扩展出 m^2 的n次方种

4.2. 指令寻址

按字编址

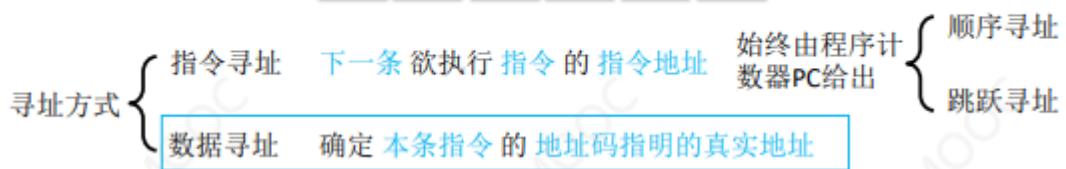
(pc) +1 ->pc

按字节编址

(pc) +2 ->pc

读入一个字，根据操作码判断这条指令的总字节数n，修改pc值

指令寻址 v.s. 数据寻址



直接寻址

指令中的形式地址A就是操作数的真实地址EA 即 $EA=A$

一条指令的执行 取指令 访存一次

执行指令 访存一次

间接寻址

间接寻址：指令的地址字段给出的形式地址不是操作数的真正地址，而是操作数有效地址所在的存储单元的地址，也就是操作数地址的地址，即 $EA=(A)$ 。

一次间接寻址需要两次访存，多次寻址需要根据存储字的最高位确定几次访存

寄存器寻址

寄存器寻址：在指令字中直接给出操作数所在的寄存器编号，即 $EA=R_i$ 其操作数在由 R_i 所指的寄存器内。

取指令 访存一次 执行指令访存0次

寄存器间接寻址

寄存器间接寻址：寄存器R中给出的不是一个操作数，而是操作数所在主存单元的地址，即 $EA=(R)$ 。

一条指令的执行：取指令访存1次执行指令访存1次暂不考虑存结果共访存2次

隐含寻址

隐含寻址：不是明显地给出操作数的地址，而是在指令中隐含着操作数的地址。

立即寻址

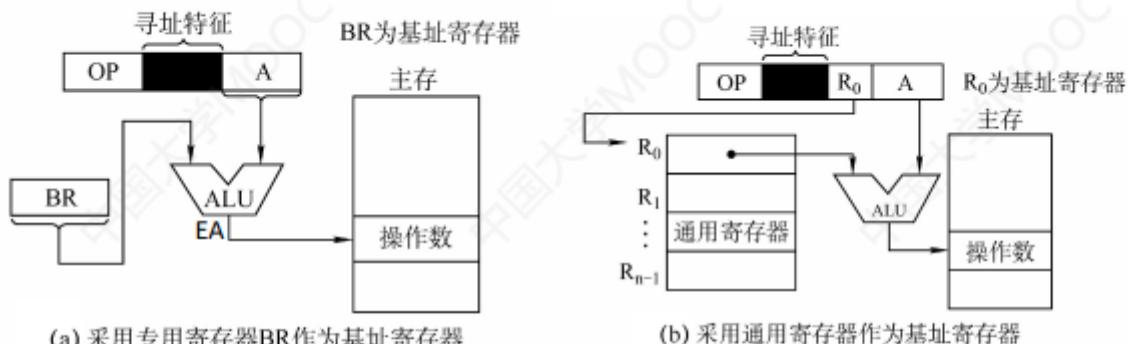
立即寻址：形式地址A就是操作数本身，又称为立即数，一般采用补码形式。#表示立即寻址特征。

寻址方式	有效地址	访存次数(指令执行期间)
隐含寻址	程序指定	0
立即寻址	A即是操作数	0
直接寻址	$EA = A$	1
一次间接寻址	$EA = (A)$	2
寄存器寻址	$EA = R_i$	0
寄存器间接一次寻址	$EA = (R_i)$	1

基址寻址

将CPU中基址寄存器（BR）的内容加上指令格式中的形式地址A，而形成操作数的有效地址，即 $EA = (BR) + A$ 。

基址寻址：将CPU中基址寄存器（BR）的内容加上指令格式中的形式地址A，而形成操作数的有效地址，即 $EA = (BR) + A$ 。



注：基址寄存器是面向操作系统的，其内容由操作系统或管理程序确定。在程序执行过程中，基址寄存器的内容不变（作为基址地址），形式地址可变（作为偏移量）。

当采用通用寄存器作为基址寄存器时，可由用户决定哪个寄存器作为基址寄存器，但其内容仍由操作系统确定。

优点：可扩大寻址范围（基址寄存器的位数大于形式地址A的位数）；用户不必考虑自己的程序存于主存的哪一空间区域，故有利于多道程序设计，以及可用于编制浮动程序（整个程序在内存里边的浮动）。

王道考研/C.SKAOYA

变址寻址

变址寻址：有效地址EA等于指令字中的形式地址A与变址寄存器IX的内容相加之和，即 $EA = (IX) + A$ ，其中IX可为变址寄存器（专用），也可用通用寄存器作为变址寄存器。

相对寻址

相对寻址：把程序计数器PC的内容加上指令格式中的形式地址A而形成操作数的有效地址，即 $EA = (PC) + A$ 其中A相对于pc所指地址的位移量可正可负，补码表示

硬件如何实现数的“比较”			注：无条件转移指令 jmp 2，就不会管PSW的各种标志位
主存地址	指令		注释
	操作码	地址码	
0	取数到ACC	#0	立即数 0 → ACC
1	取数到IX	#0	立即数 0 → IX
2	ACC加法	7 (数组始址)	(ACC)+(7+(IX)) → ACC
3	IX加法	#1	(IX)+1 → IX
4	IX比较	#10	比较10-(IX)
5	条件跳转	2	若结果>0 则PC跳转到2
6	从ACC存数	17	(ACC) → sum变量
7	随便什么值		a[0]
8	随便什么值		a[1]
9	随便什么值		a[2]
...
16	随便什么值		a[9]
17	初始为0		sum变量

堆栈寻址

操作数存放在堆栈中，隐含使用堆栈指针（SP）作为操作数地址

入栈/出栈是EA的确定方式不同，硬堆栈不访存，软堆栈访存一次

x86架构CPU

EAX

EBX

ECX

EDX

ESI 变址寄存器 I=Index S= Source D=Destineation

EDI

EBP 堆栈基指针

ESP 堆栈顶指针



4.3. 汇编指令

加	add	add d,s	#计算d+s, 结果存入d
减	subtract	sub d,s	#计算d-s, 结果存入d
乘	multiply	mul d,s imul d,s	#无符号数d*s, 乘积存入d #有符号数d*s, 乘积存入d
除	divide	div s idiv s	#无符号数除法 edx:eax/s, 商存入eax, 余数存入edx #有符号数除法 edx:eax/s, 商存入eax, 余数存入edx
取负数	negative	neg d	#将d取负数, 结果存入d
自增++	increase	inc d	#将d++, 结果存入d
自减--	decrease	dec d	#将d--, 结果存入d

AT&T格式 v.s. Intel格式

	AT&T 格式	Intel 格式
目的操作数d、源操作数s	<code>op s, d</code> 注：源操作数在左，目的操作数在右	<code>op d, s</code> 注：源操作数在右，目的操作数在左
寄存器的表示	<code>mov %ebx, %eax</code> 注：寄存器名之前必须加“%”	<code>mov eax, ebx</code> 注：直接写寄存器名即可
立即数的表示	<code>mov \$985, %eax</code> 注：立即数之前必须加“\$”	<code>mov eax, 985</code> 注：直接写数字即可
主存地址的表示	<code>mov %eax, (af996h)</code> 注：用“小括号”	<code>mov [af996h], eax</code> 注：用“中括号”
读写长度的表示	<code>movb \$5, (af996h)</code> <code>movw \$5, (af996h)</code> <code>movl \$5, (af996h)</code> <code>addb \$4, (af996h)</code> 注：指令后加 b、w、l 分别表示读写长度为 byte、word、dword	<code>mov byte ptr [af996h], 5</code> <code>mov word ptr [af996h], 5</code> <code>mov dword ptr [af996h], 5</code> <code>add byte ptr [af996h], 4</code> 注：在主存地址前说明读写长度byte、word、dword
主存地址偏移量的表示	<code>movl -8(%ebx), %eax</code> 注：偏移量(基础) <code>movl 4(%ebx, %ecx, 32), %eax</code> 注：偏移量(基础, 变址, 比例因子)	<code>mov eax, [ebx - 8]</code> 注：[基础+偏移量] <code>mov eax, [ebx + ecx * 32 + 4]</code> 注：[基础+变址*比例因子+偏移量]

无条件转义指令jmp

<pre>cmp a, b #比较a和b两个数</pre> <pre>je <地址> #jump when equal, 若a==b则跳转 jne <地址> #jump when not equal, 若a!=b则跳转 jg <地址> #jump when greater than, 若a>b则跳转 jge <地址> #jump when greater than or equal to, 若a>=b则跳转 jl <地址> #jump when less than, 若a<b则跳转 jle <地址> #jump when less than or equal to, 若a<=b则跳转</pre> <pre>cmp eax, ebx #比较寄存器eax和ebx里的值 jg NEXT #若 eax > ebx, 则跳转到 NEXT:</pre>	<p>a、b两个数可能来自寄存器/主存/常量</p> <p>条件转移指令一般要和 cmp 指令一起使用</p> <p style="color: yellow; font-size: 2em; font-weight: bold;">正确套路</p>
---	---

OF 溢出标志 溢出时为1 否则置0

SF 符号标志 结果为负时置1，否则为0

ZF 0标志 运算结果为0时ZF位置为1否则为0

CF 进位/借位时置1 否则为0

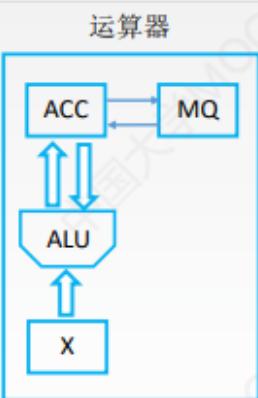
循环语句

CISC和RISC		
对比项目	CISC	RISC
指令系统	复杂, 庞大	简单, 精简
指令数目	一般大于200条	一般小于100条
指令字长	不固定	定长
可访存指令	不加限制	只有Load/Store指令
各种指令执行时间	相差较大	绝大多数在一个周期内完成
各种指令使用频度	相差很大	都比较常用
通用寄存器数量	较少	多
目标代码	难以用优化编译生成高效的目标代码程序	采用优化的编译程序, 生成代码较为高效
控制方式	绝大多数为微程序控制	绝大多数为组合逻辑控制
指令流水线	可以通过一定方式实现	必须实现

5. 中央处理器

运算器：用于实现算术运算（如：加减乘除），逻辑运算（如，与或非）

运算器基本组成



ACC：累加器，用于存放操作数，或运算结果。
MQ：乘商寄存器，在乘、除运算时，用于存放操作数或运算结果。

X：通用的操作数寄存器，用于存放操作数
ALU：算术逻辑单元，通过内部复杂的电路实现算数运算、逻辑运算

控制器的基本组成

控制器的基本组成



CU：控制单元，分析指令，给出控制信号

IR：指令寄存器，存放当前执行的指令

PC：程序计数器，存放下一条指令地址，有自动加1功能

5.1. CPU功能和基本结构

5.1.1. CPU的功能

1. 指令控制。完成取指令、分析指令和执行指令的操作，即程序的顺序控制。
2. 操作控制。一条指令的功能往往是由若干操作信号的组合来实现的。CPU管理并产生由内存取出的每条指令的操作信号，把各种操作信号送往相应的部件，从而控制这些部件按指令的要求进行动作。
3. 时间控制。对各种操作加以时间上的控制。时间控制要为每条指令按时间顺序提供应有的控制信号。
4. 数据加工。对数据进行算术和逻辑运算。
5. 中断处理。对计算机运行过程中出现的异常情况和特殊请求

5.1.2. 运算器和控制器的功能和结构

运算器对数据进行加工

控制器 协调并控制计算机各部件执行程序的指令序列

基本功能包括取指令、分析指令、执行指令

取指令：自动形成指令地址；自动发出取指令的命令。

分析指令：操作码译码(分析本条指令要完成什么操作)；产生操作数的有效地址。

执行指令：根据分析指令得到的“操作命令”和“操作数地址”，形成操作信号控制序列，控制运算器、存储器以及I/O设备完成相应的操作。

中断处理：管理总线及输入输出；处理异常情况(如掉电) 和特殊请求(如打印机请求打印一行字符)。

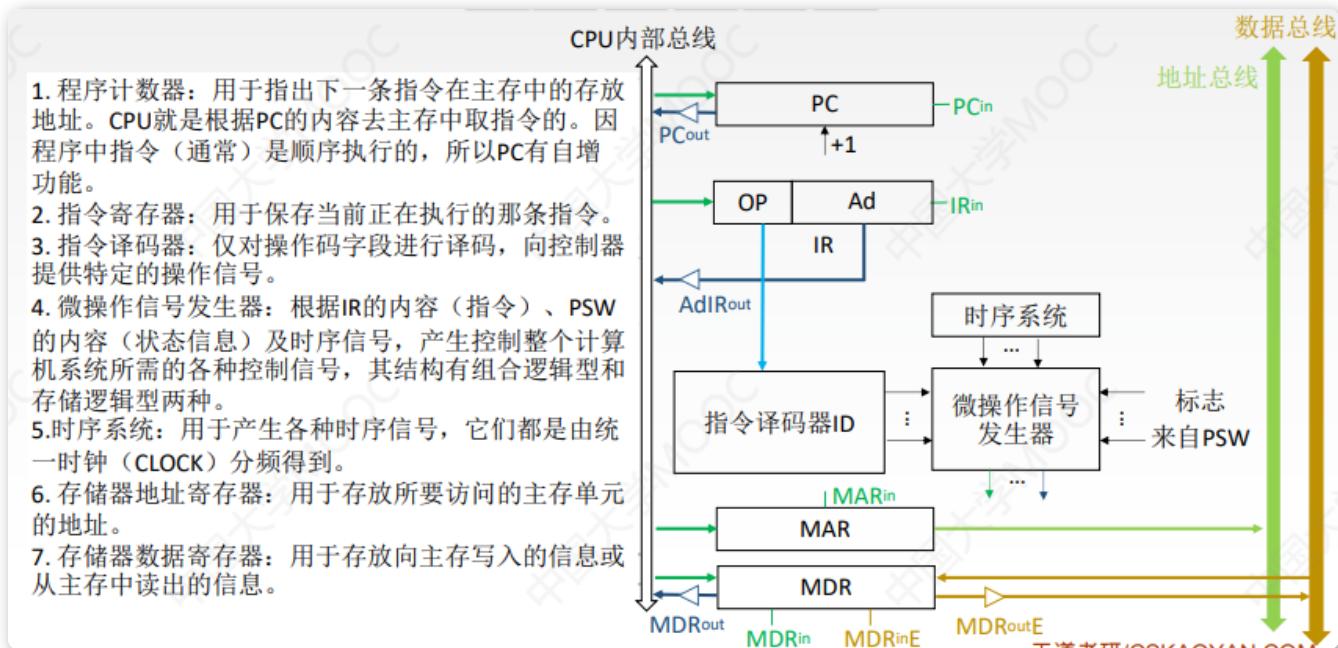
运算器的基本结构

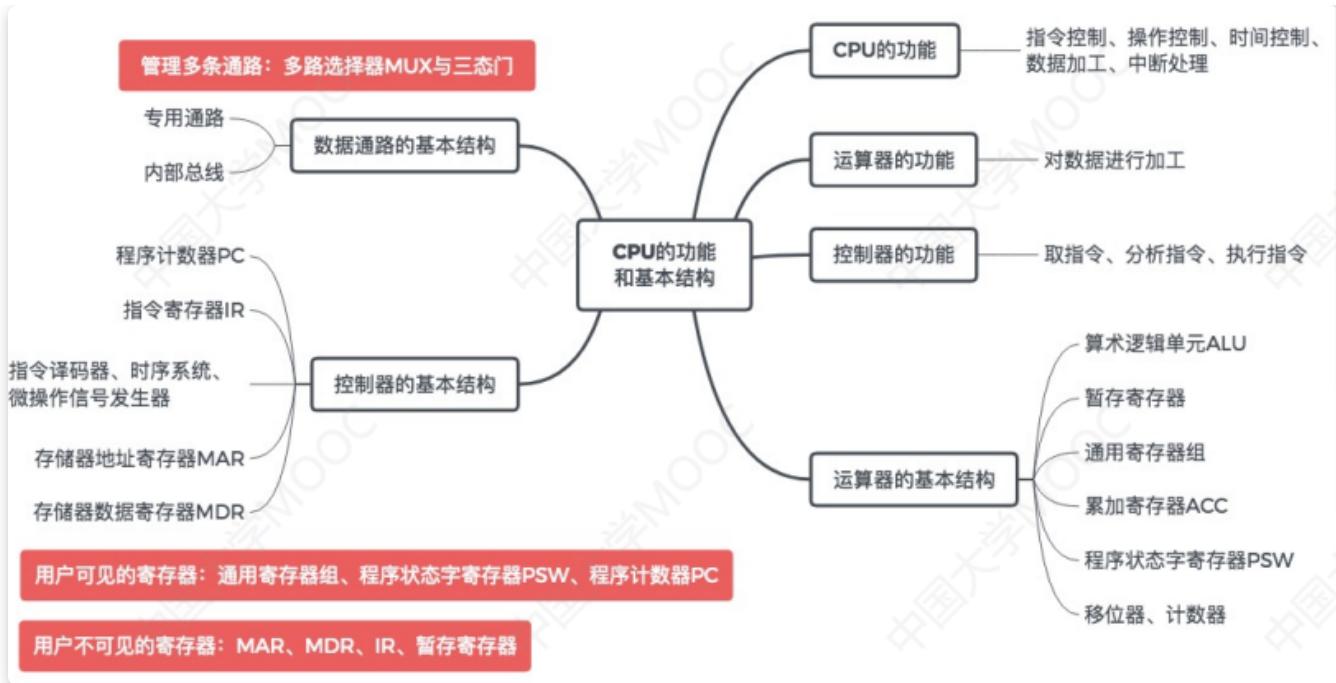
1. 算术逻辑单元：主要功能是进行算术/逻辑运算。
2. 通用寄存器组：如AX、BX、CX、DX、SP等，用于存放操作数（包括源操作数、目的操作数及中间结果）和各种地址信息等。SP是堆栈指针，用于指示栈顶的地址。
3. 暂存寄存器：用于暂存从主存读来的数据这个数据不能存放在通用寄存器中，否则会破坏其原有内容
4. 累加寄存器：它是一个通用寄存器，用于暂时存放ALU运算的结果信息，用于实现加法运算。
5. 程序状态字寄存器：保留由算术逻辑运算指令或测试指令的结果而建立的各种状态信息，如溢出标志（OP）、符号标志（SF）、零标志（ZF）、进位标志（CF）等。PSW中的这些位参与并决定微操作的形成。
6. 移位器：对运算结果进行移位运算。
7. 计数器：控制乘除运算的操作步数。

cpu内部单总线方式：将所有寄存器的输入端和输出端都连接到一条公共的通路上

多路选择器和三态门

控制器基本结构





5.2. 数据通路

指令周期的数据流

数据通路：数据在功能部件之间传送的路径

基本结构：CPU内部单总线结构，CPU内部多总线结构，专用数据通路方式

由控制部件产生的控制信号建立数据通路

内部总线是指同一部件，如CPU内部连接各寄存器及运算部件之间的总线

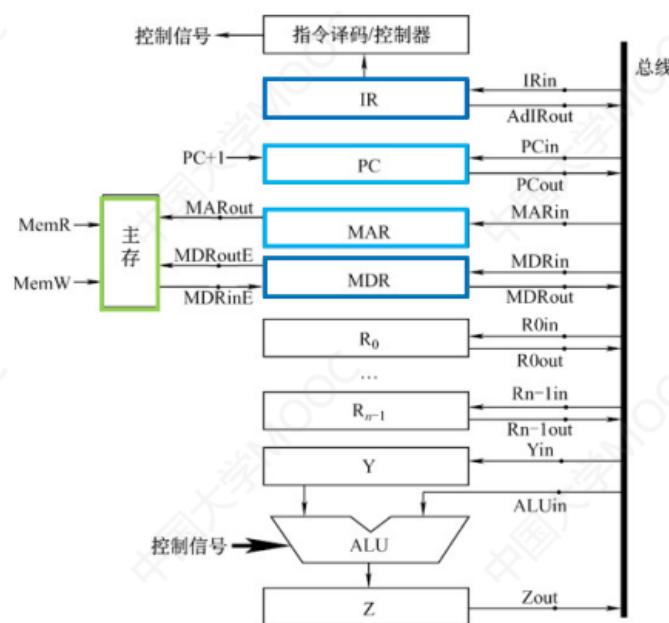
系统总线是指同一台计算机系统的各个部件，如CPU，内存，通道和各类I/O接口间互相连接的总线

1. 寄存器之间的数据传送

2. 主存与CPU之间的数据传送

3. 执行算术或逻辑运算

CPU内部单总线方式-例题



设有如图所示的单总线结构，分析指令 ADD (R0), R1 的指令流程和控制信号。

- 分析指令功能和指令周期
功能: $((R0)) + (R1) \rightarrow (R0)$
取指周期、间址周期、执行周期

- 写出各阶段的指令流程
取指周期: 公共操作

时序	微操作	有效控制信号
1	$(PC) \rightarrow MAR$	$PCout, MARin$
2	$M(MAR) \rightarrow MDR$ $(PC)+1 \rightarrow PC$	$MemR, MARout, MDRinE$
3	$(MDR) \rightarrow IR$	$MDRout, IRin$
4	指令译码	-

王道考研/GSKAOYAN.COM

与主存相连的寄存器是MAR和MDR

解题基本思路

寄存器之间的数据传送，主存和CPU之间的数据传送，使用ALU进行算术逻辑运算

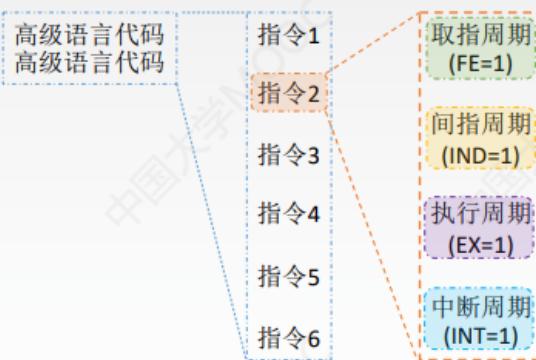
基本思路，利用题目提供的数据通路进行数据传送

有CU发出控制信号实现通路的建立

根据 指令操作码、目前的机器周期、节拍信号、机器状态条件，即可确定现在这个节拍下应该发出哪些“微命令”

内容回顾

CU发出一个微命令，可完成对应微操作。
如：微命令1使得 $PCout, MARin$ 有效。
完成对应的微操作1 $(PC) \rightarrow MAR$



T₀: 微操作1、微操作2
T₁: 微操作3
T₂: 微操作4

一个节拍内可以并行完成多个“相容的”微操作

T₀: 微操作5、微操作2
T₁: 微操作6
T₂: 微操作7

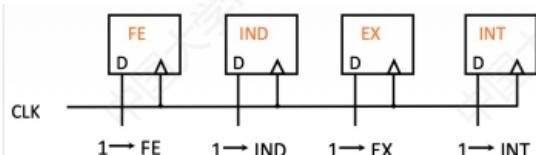
同一个微操作可能在不同指令的不同阶段被使用

T₀:
T₁: 微操作8
T₂: 微操作9、微操作6

不同指令的执行周期所需节拍数各不相同。为了简化设计，选择定长的机器周期，以可能出现的最大节拍数为准（通常以访存所需节拍数作为参考）

T₀:
T₁: 微操作10
T₂: 微操作11

若实际所需节拍数较少，可将微操作安排在机器周期末尾几个节拍上进行

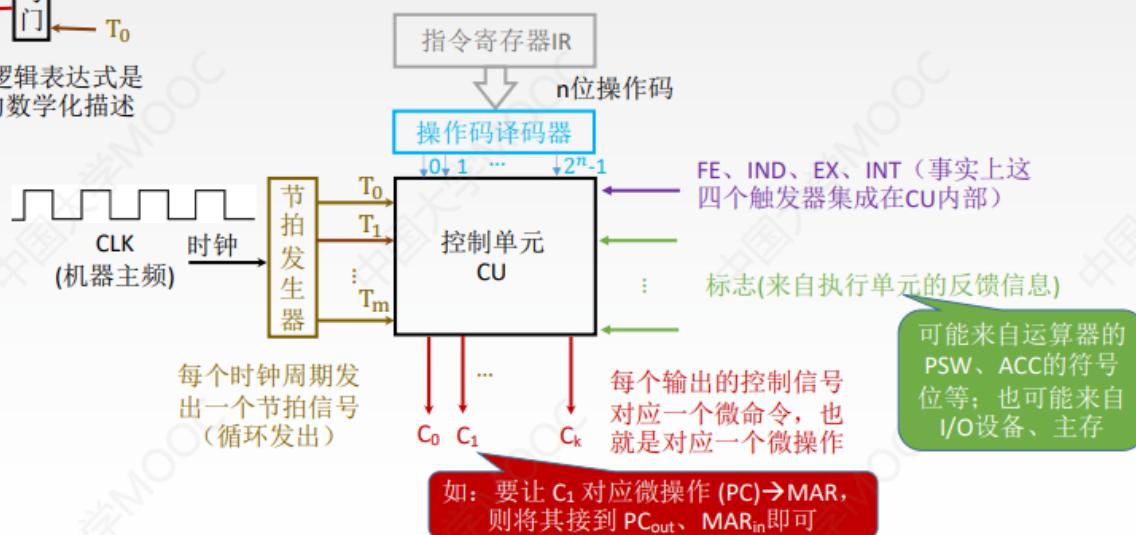


所有指令的取指周期、 T_0 节拍下一定要完成 $(PC) \rightarrow MAR$ 。则可知 $C_1 = FE \cdot T_0$



Tips: 逻辑表达式是电路的数学化描述

硬布线控制器



根据 **指令操作码**、目前的机器周期、节拍信号、机器状态条件，即可确定现在这个节拍下应该发出哪些“微命令”

硬布线控制器

1. 分析每个阶段的微操作序列（取值，间址，执行，中断四个阶段）

2. 选择CPU控制方式

3. 安排微操作时序

原则（1，先后顺序不得随意更改，2，被控对象不同的微操作尽量安排在一个节拍内完成，占用时间较短的微操作尽量安排在一个节拍内完成，并运行有先后顺序）

4. 电路设计 (确定每个微操作命令的逻辑表达式，并用电路实现)

5.3. 微程序

5.4. 指令流水

一条指令的执行过程

取指：根据PC内容访问主存储器，取出一条指令送到IR中。

分析：对指令操作码进行译码，按照给定的寻址方式和地址字段中的内容形成操作数的有效地址EA，并从有效地址EA中取出操作数。

执行：根据操作码字段，完成指令规定的功能，即把运算结果写到通用寄存器或主存中。

顺序执行方式

总耗时 $T = n * 3t = 3nt$

传统冯洛依曼机采用顺序执行方式，串行执行方式

控制简单，硬件代价小

一次重叠执行方式

$T = (1+2n)t$

二次重叠方式

$T = (2+n)t$

指令执行过程图

时空图

流水线的性能指标

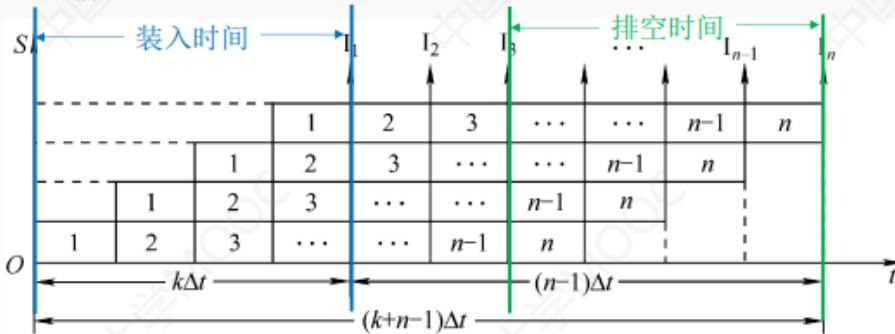
吞吐率 吞吐率是指在单位时间内流水线所完成的任务数量

设任务位n，处理完成n个任务所用的时间为Tk

则流水线吞吐率 $TP = n/T_k$

理想情况下，流水线的时空图如下：

当连续输入的任务 $n \rightarrow \infty$ 时，得最大吞吐率为 $TP_{max} = 1/\Delta t$ 。



$$T_k = (k+n-1)\Delta t$$

流水线的实际吞吐率为

$$TP = \frac{n}{(k+n-1)\Delta t}$$

一条指令的执行分为k个阶段，每个阶段耗时Δt，一般取Δt=一个时钟周期

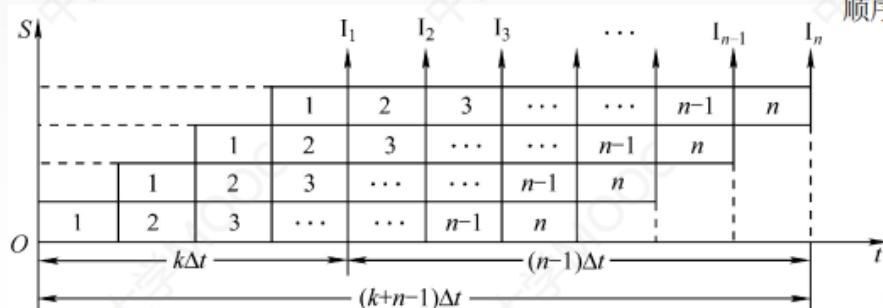
加速比

2. 加速比 完成同样一批任务，不使用流水线所用的时间与使用流水线所用的时间之比。

设 T_0 表示不使用流水线时的执行时间，即顺序执行所用的时间； T_k 表示使用流水线时的执行时间

则计算流水线加速比 (S) 的基本公式为 $S = \frac{T_0}{T_k}$ 当连续输入的任务 $n \rightarrow \infty$ 时，最大加速比为 $S_{\max} = k$ 。

理想情况下，流水线的时空图如下：



实际加速比为

$$S = \frac{kn \Delta t}{(k+n-1) \Delta t} = \frac{kn}{k+n-1}$$

一条指令的执行分为 k 个阶段，每个阶段耗时 Δt ，一般取 Δt = 一个时钟周期

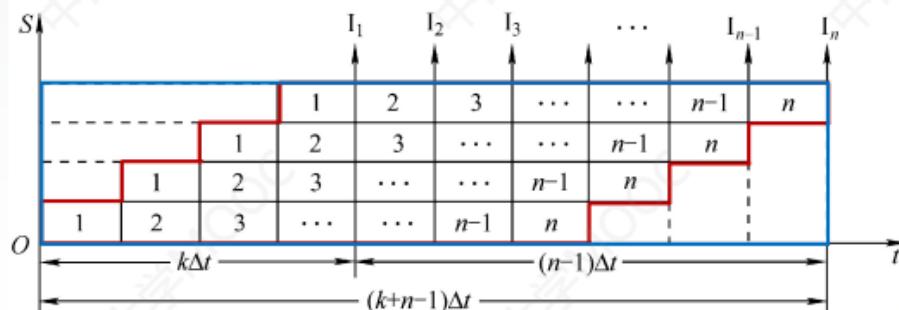
3. 效率

流水线的设备利用率为流水线的效率。

在时空图上，流水线的效率定义为 **完成 n 个任务占用的时空区有效面积** 与 **n 个任务所用的时间与 k 个流水段所围成的时空区总面积之比**。

则流水线效率 (E) 的一般公式为 $E = \frac{n \text{ 个任务占用 } k \text{ 时空区有效面积}}{n \text{ 个任务所用的时间与 } k \text{ 个流水段所围成的时空区总面积}} = \frac{T_0}{k T_k}$

理想情况下，流水线的时空图如下：



一条指令的执行分为 k 个阶段，每个阶段耗时 Δt ，一般取 Δt = 一个时钟周期

硬件多线程

细粒度多线程，粗粒度多线程，同时多线程 (SMT)

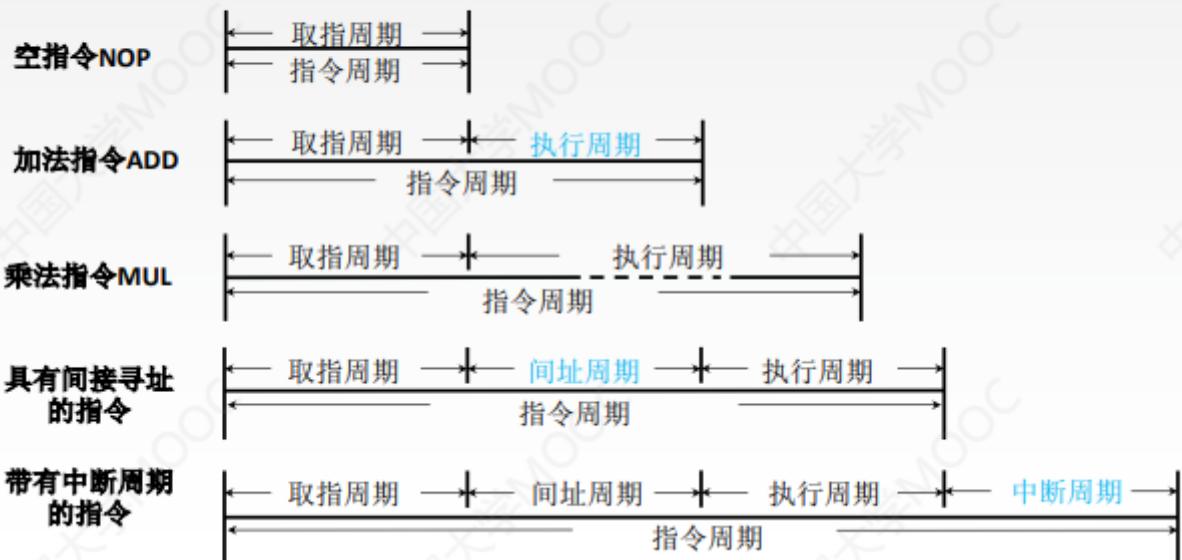
三种硬件多线程

硬件多线程			
	细粒度多线程	粗粒度多线程	同时多线程 (SMT)
指令发射	轮流发射各线程的指令 (每个时钟周期发射一个线程)	连续几个时钟周期，都发射同一线程的指令序列，流水线阻塞时，切换另一个线程	一个时钟周期内，同时发射多个线程的指令
线程切换频率	每个时钟周期切换一次线程	只有流水线阻塞时才切换一次线程	NULL
线程切换代价	低	高，需要重载流水线	NULL
并行性	指令级并行，线程间不并行	指令级并行，线程间不并行	指令级并行，线程级并行

5.5. 指令执行过程

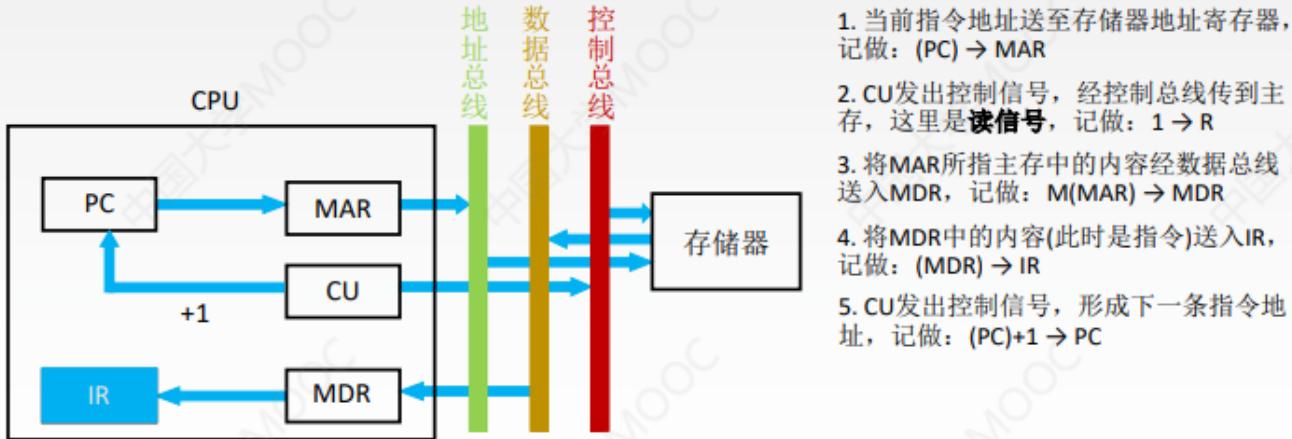
指令周期

每个指令周期内机器周期数可以不等，每个机器周期内的节拍数也可以不等。



四个工作周期都有CPU访存操作，只是访存的目的不同。取指周期是为了取指令，间址周期是为了取有效地址，执行周期是为了取操作数，中断周期是为了保存程序断点。

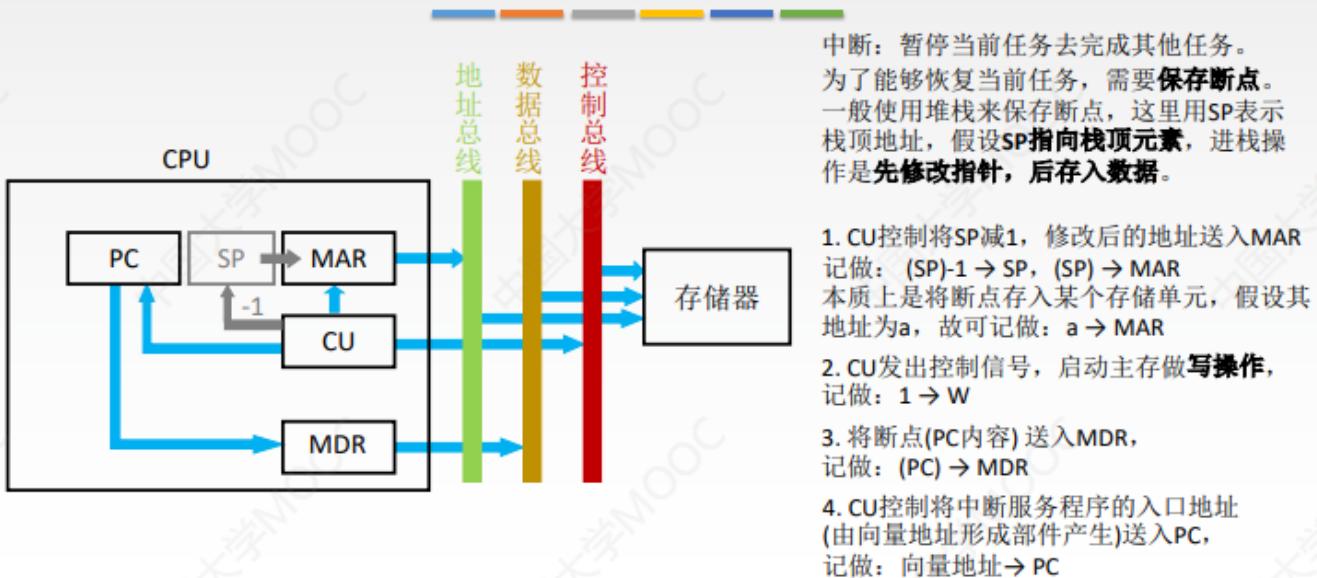
指令周期的数据流-取指周期



指令周期的数据流-间址周期

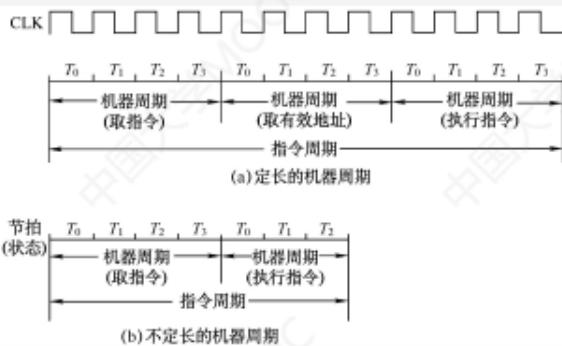


指令周期的数据流-中断周期



指令执行方案

一个指令周期通常要包括几个时间段（执行步骤），每个步骤完成指令的一部分功能，几个依次执行的步骤完成这条指令的全部功能。



方案1. 单指令周期

对所有指令都选用相同的执行时间来完成。

指令之间串行执行；指令周期取决于执行时间最长的指令的执行时间。

对于那些本来可以在更短时间内完成的指令，要使用这个较长的周期来完成，会降低整个系统的运行速度。

方案2. 多指令周期

对不同类型的指令选用不同的执行步骤来完成。

指令之间串行执行；可选用不同个数的时钟周期来完成不同指令的执行过程。

需要更复杂的硬件设计。

方案3. 流水线方案

在每一个时钟周期启动一条指令，尽量让多条指令同时运行，但各自处在不同的执行步骤中。
指令之间并行执行。

6. 系统总线

总线是连接各个部件的信息传输线是各个部件共享的传输介质

串行总线 并行总线

单总线

CPU双总线

存储双总线

6.1. 总线分类

片总线 芯片内部的总线

系统总线

计算机各部件之间的信息传输线（数据总线（双向，与机器字长，存储字长有关），地址总线（单向，与存储地址，I/O地址有关），控制总线（有出（中断请求，总线请求）有入（存储器读，存储器写，总线允许，中断缺人）））

通信总线

用于计算机系统之间或计算机系统与其它系统之间的通信

传输方式有串行和并行

6.2. 总线特性 性能指标

6.2.1. 特性

机械特性 尺寸，形状，管脚数，排列顺序

电气特性 传输方向和有效电平范围

功能特性 每根传输线的功能（地址，数据，控制）

时间特性 信号和时序关系

6.2.2. 总线的性能指标

总线的宽度 数据线的根数

标准传输率 每秒传输的最大字节数 (Mbps)

时钟同步/异步 同步，不同步

总线复用 地址线与数据线复用

信号线数 地址线、数据线和控制线的总和

总线控制方式 突发、自动、仲裁、逻辑、计数

其它 负载能力

总线的传输周期（总线周期）

一次总线操作所需的时间（包括申请阶段、寻址阶段、传输阶段和结束阶段），通常由若干个总线时钟周期构成。

总线的时钟周期

即机器的时钟周期。计算机有一个统一的时钟，以控制整个计算机的各个部件，总线也要受此时钟的控制。

总线的工作频率

总线上各种操作的频率，为总线周期的倒数。若总线周期=N个时钟周期，则总线的工作频率=时钟频率/N。实际上指一秒内传送几次数据。

总线的时钟频率

即机器的时钟频率，为时钟周期的倒数。数量若时钟周期为T，则时钟频率为 $1/T$ 。实际上指一秒内有多少个时钟周期。

总线宽度

又称为总线位宽，它是总线上同时能够传输的数据位数，通常是指数据总线的根数，如32根称为32位（bit）总线。

总线带宽

可理解为总线的数据传输率，即单位时间内总线上可传输数据的位数，通常用每秒钟传送信息的字节数来衡量，单位可用字节/秒（B/s）表示。

总线带宽=总线工作频率×总线宽度（bit/s）=总线工作频率×（总线宽度/8）（B/s）

总线宽度/总线周期(bit/s)=总线宽度/8/总线周期(B/s)

注：总线带宽是指总线本身所能达到的最高传输速率。

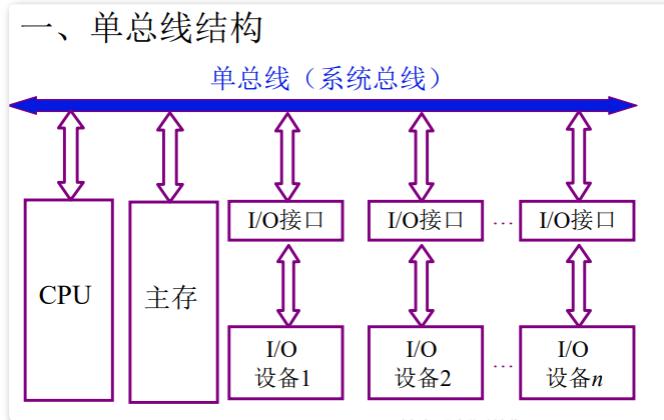
在计算实际的有效数据传输率时，要用实际传输的数据量除以耗时。

6.3. //((非)408)总线标准

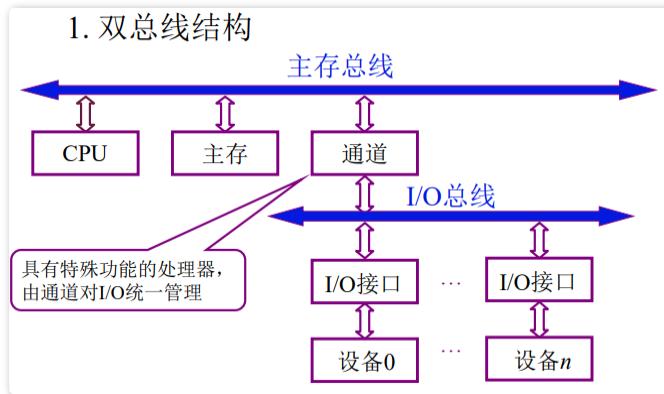
总线标准	数据线	总线时钟	带宽
ISA	16	8 MHz（独立）	16 MBps
EISA	32	8 MHz（独立）	33 MBps
VESA (VL-BUS)	32	32 MHz（CPU）	132 MBps
PCI	32 64	33 MHz（独立） 66 MHz（独立）	132 MBps 528 MBps
AGP	32	66.7 MHz（独立） 133 MHz（独立）	266 MBps 533 MBps
RS-232	串行通信 总线标准	数据终端设备（计算机）和数据通信设备 (调制解调器)之间的标准接口	
USB	串行接口 总线标准	普通无屏蔽双绞线 带屏蔽双绞线 最高	1.5 Mbps (USB1.0) 12 Mbps (USB1.0) 480 Mbps (USB2.0)

6.4. 总线结构

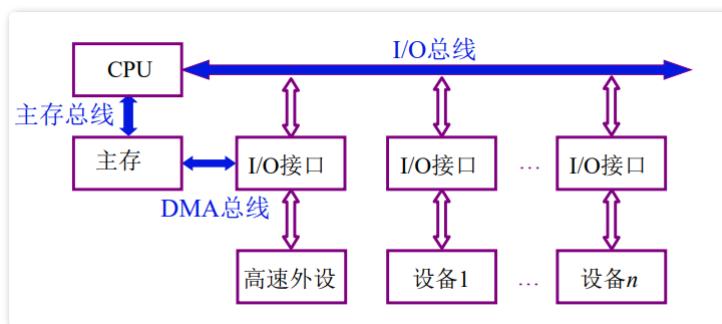
单总线结构

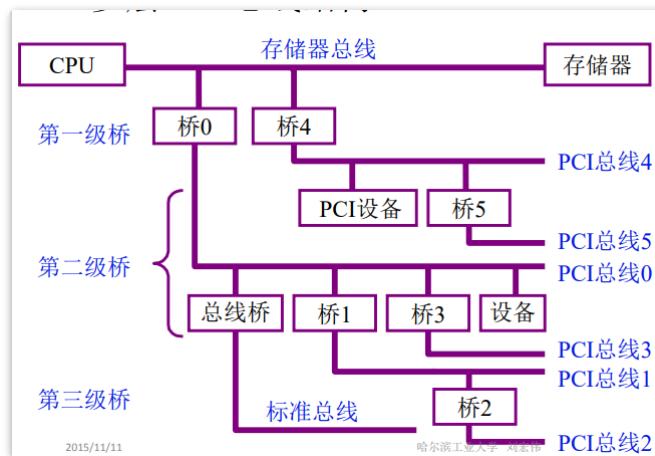
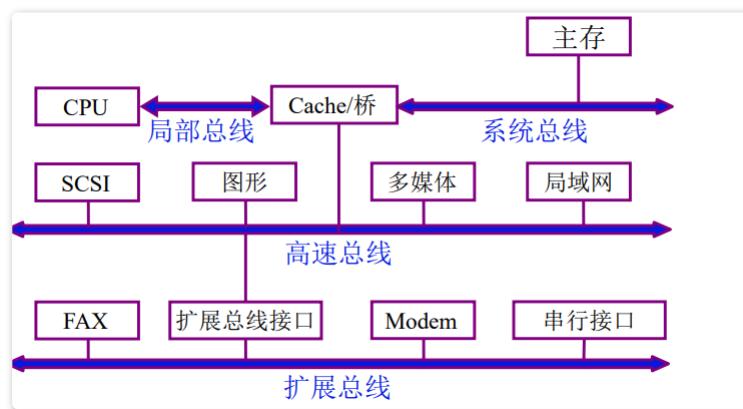
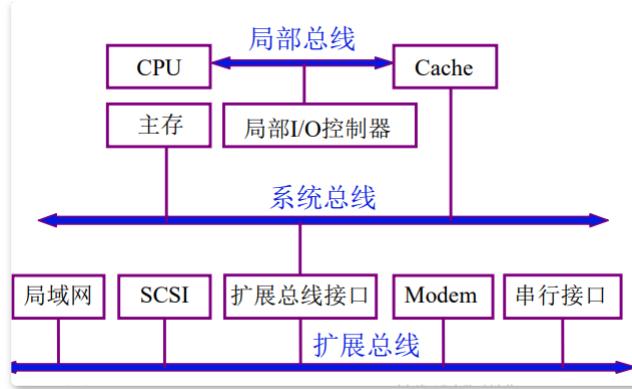


多总线结构



三总线结构





6.5. 总线操作和定时

总线传输的四个阶段

总线周期的四个阶段

1) **申请分配阶段**: 由需要使用总线的主模块(或主设备)提出申请, 经总线仲裁机构决定将下一传输周期的总线使用权授予某一申请者。也可将此阶段细分为**传输请求**和**总线仲裁**两个阶段。

2) **寻址阶段**: 获得使用权的主模块通过总线**发出**本次要访问的从模块的**地址**及有关**命令**, 启动参与本次传输的从模块。

3) **传输阶段**: 主模块和从模块进行**数据交换**, 可单向或双向进行数据传送。

4) **结束阶段**: 主模块的**有关信息**均从系统总线上**撤除**, 让出总线使用权。

总线定时是指总线在双方交换数据的过程中需要时间上配合关系的控制, 这种控制称为总线定时, 它的实质是一种协议或规则

同步通信(同步定时方式)	由 统一时钟 控制数据传送
异步通信(异步定时方式)	采用 应答方式 , 没有公共时钟标准
半同步通信	同步、异步结合
分离式通信	充分 挖掘 系统 总线每瞬间的潜力

同步定时方式(同步通信)

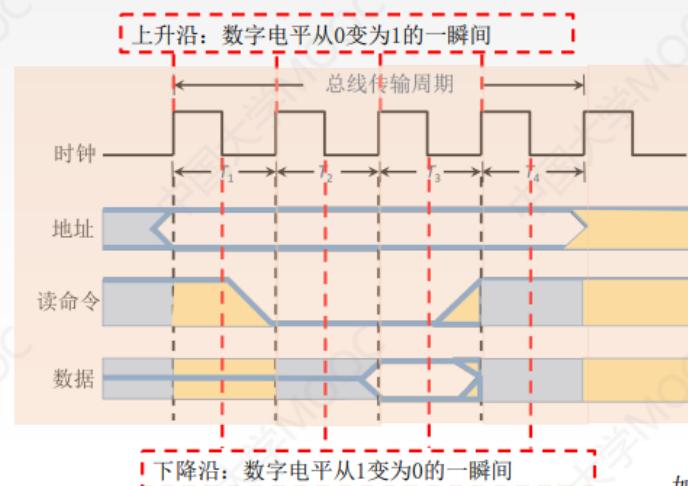
异步定时方式(异步通信)

半同步通信

分离式通信

同步定时方式-读命令

总线控制器采用一个**统一的时钟信号**来协调发送和接收双方的传送定时关系。



假设: CPU作为**主设备**, 某个输入设备作为**从设备**

1) CPU在T1时刻的上升沿给出地址信息

2) 在T2的上升沿给出读命令(低电平有效), 与地址信息相符合的输入设备按命令进行一系列的内部操作, 且必须在T3的上升沿来之前将CPU所需的数据送到数据总线上。

3) CPU在T3时钟周期内, 将数据线上的信息传送到其内部寄存器中。

4) CPU在T4的上升沿撤销读命令, 输入设备不再向数据总线上传送数据, 撤销它对数据总线的驱动。

如果从设备跟不上节奏, 在T3给不出数据, 就哦豁了~

同步定时方式(同步通信)

异步定时方式(异步通信)

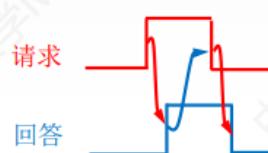
半同步通信

分离式通信

Eg: 地址信息、
读命令



Eg: 要读出的数据



1) 不互锁方式 速度最快 可靠性最差

主设备发出“请求”信号后，不必等到接到从设备的“回答”信号，而是经过一段时间，便撤销“请求”信号。

而从设备在接到“请求”信号后，发出“回答”信号，并经过一段时间，自动撤销“回答”信号。双方不存在互锁关系。

2) 半互锁方式

主设备发出“请求”信号后，必须待接到从设备的“回答”信号后，才撤销“请求”信号，有互锁的关系。

而从设备在接到“请求”信号后，发出“回答”信号，但不必等待获知主设备的“请求”信号已经撤销，而是隔一段时间后自动撤销“回答”信号，不存在互锁关系。

3) 全互锁方式 最可靠 速度最慢

主设备发出“请求”信号后，必须待从设备“回答”后，才撤销“请求”信号；

从设备发出“回答”信号，必须待获知主设备“请求”信号已撤销后，再撤销其“回答”信号。双方存在互锁关系。

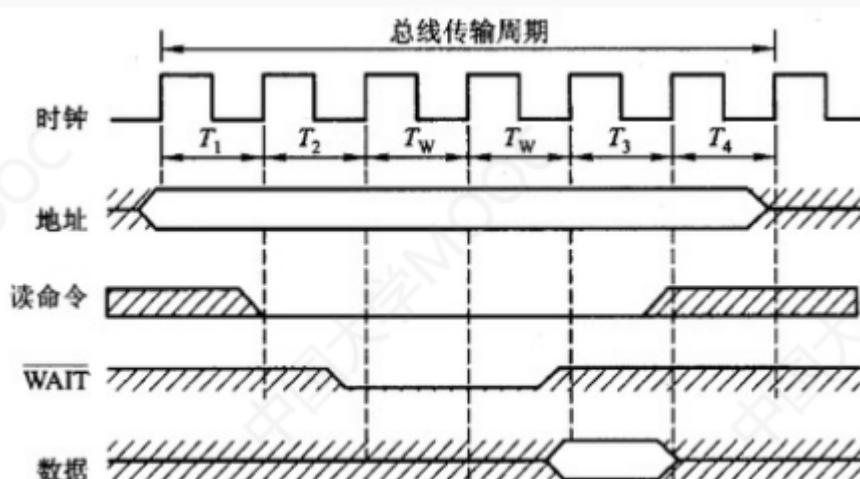
半同步通信

同步 发送方用系统时钟前沿发信号

接收方用系统时钟后沿判断、识别

异步 允许不同速度的模块和谐工作

半同步通信：统一时钟的基础上，增加一个“等待”响应信号 WAIT



分离式通信

上述三种通信的共同点

一个总线传输周期（以输入数据为例）

- | | |
|--------------|-----------------|
| • 主模块发地址、命令 | 使用总线 |
| • 从模块准备数据 | 不使用总线 总线空闲 |
| • 从模块向主模块发数据 | 使用总线 |

分离式通信的一个总线传输周期

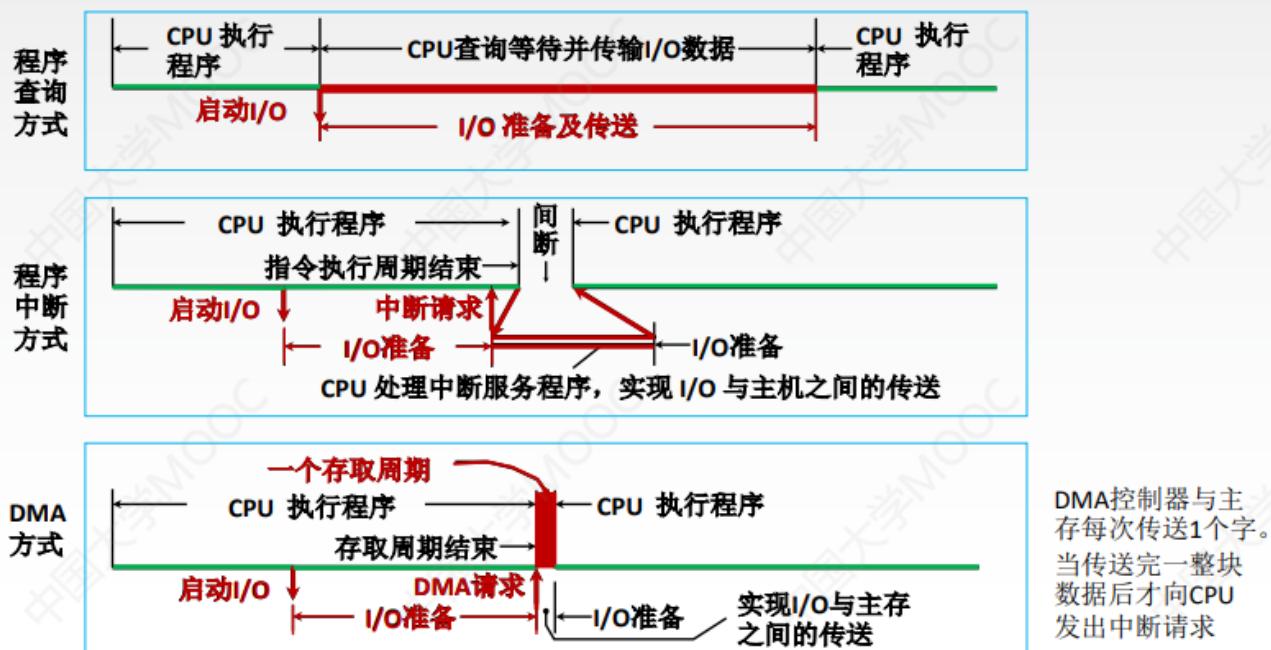
- {
- | | |
|------|----------------------------|
| 子周期1 | 主模块申请占用总线，使用完后
放弃总线的使用权 |
| 子周期2 | 从模块申请占用总线，将各种信
息送至总线上 |

特点：

1. 各模块均有权申请占用总线
2. 采用同步方式通信，不等对方回答
3. 各模块准备数据时，不占用总线
4. 总线利用率提高

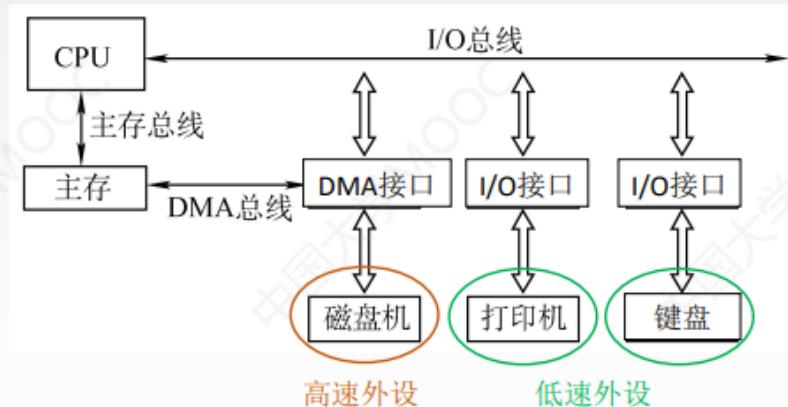
7. 输入输出系统

I/O接口，I/O控制器，设备控制器，负责协调主机和外部设备之间的数据传输



DMA控制方式

DMA控制方式



DMA: Direct Memory Access,
直接内存访问。

注: DMA接口, 即DMA控制器, 也是一种特殊的I/O控制器

DMA控制方式: 主存与高速I/O设备之间有一条直接数据通路(DMA总线)。CPU向DMA接口发出“读/写”命令, 并指明主存地址、磁盘地址、读写数据量等参数。

DMA控制器自动控制磁盘与主存的数据读写, 每完成一整块数据读写(如1KB为一整块), 才向CPU发出一次中断请求。

通道控制方式

通道是具有特殊功能的处理器, 能对I/O设备进行统一的管理

7.1. 基本组成

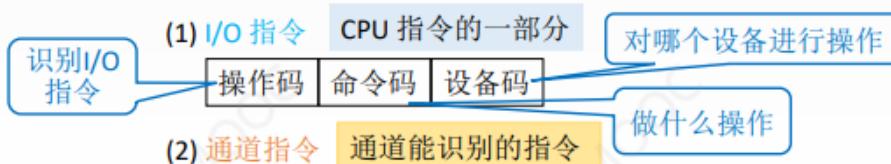
一般来说, I/O系统由I/O软件和I/O硬件两部分构成。

1. I/O 硬件 包括外部设备、I/O接口、I/O总线等。



2. I/O 软件 包括驱动程序、用户程序、管理程序、升级补丁等。

通常采用I/O指令和通道指令实现主机和I/O设备的信息交换。



注: I/O 指令与普通
指令格式略有不同,
操作码指明了CPU要
对I/O接口做什么, 命
令码指明了I/O接口要
对设备做什么

通道程序提前编制好放在主存中

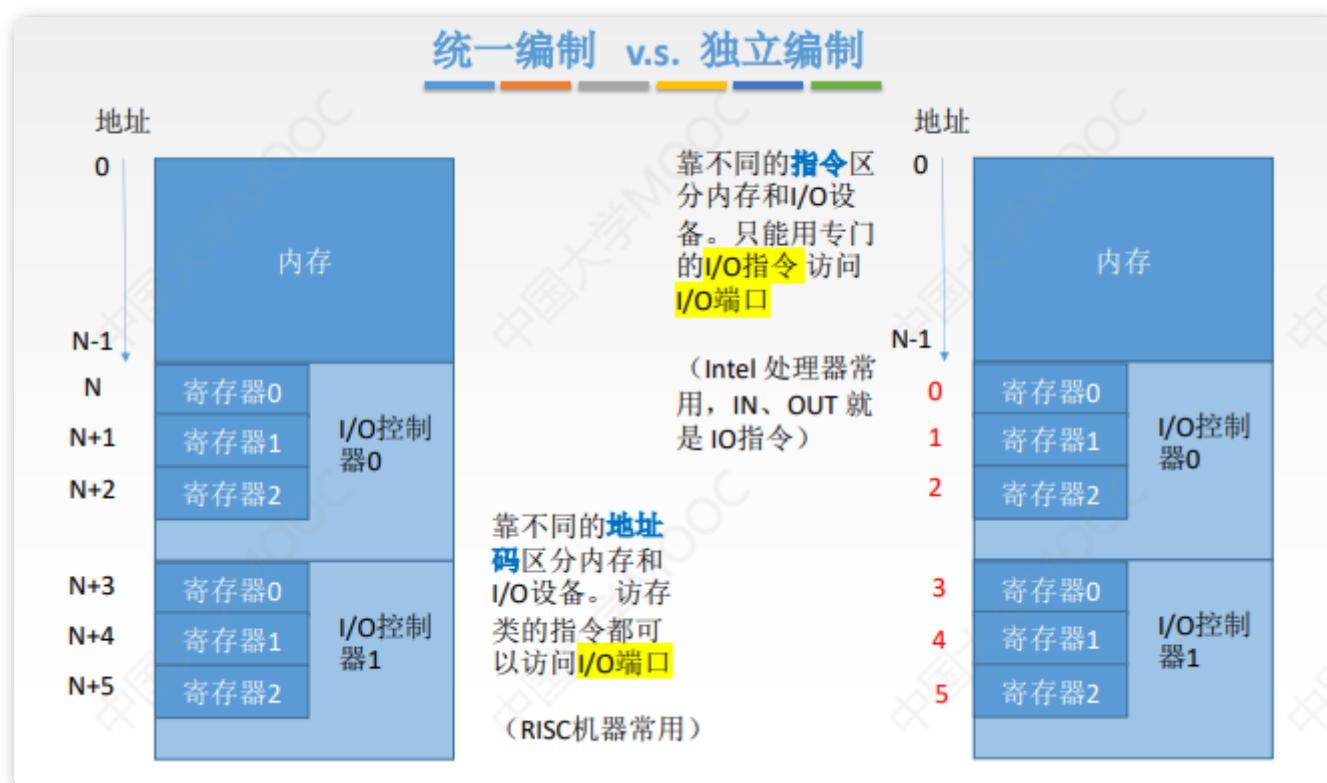
在含有通道的计算机中, CPU执行I/O指令对通道发出命令, 由通道执行一系列通道指令, 代替CPU对I/O设备进行管理

7.2. 设备

显示屏，灰度级 n位可以表示 2^n 种不同的亮度或者颜色

容量=分辨率x灰度级位数

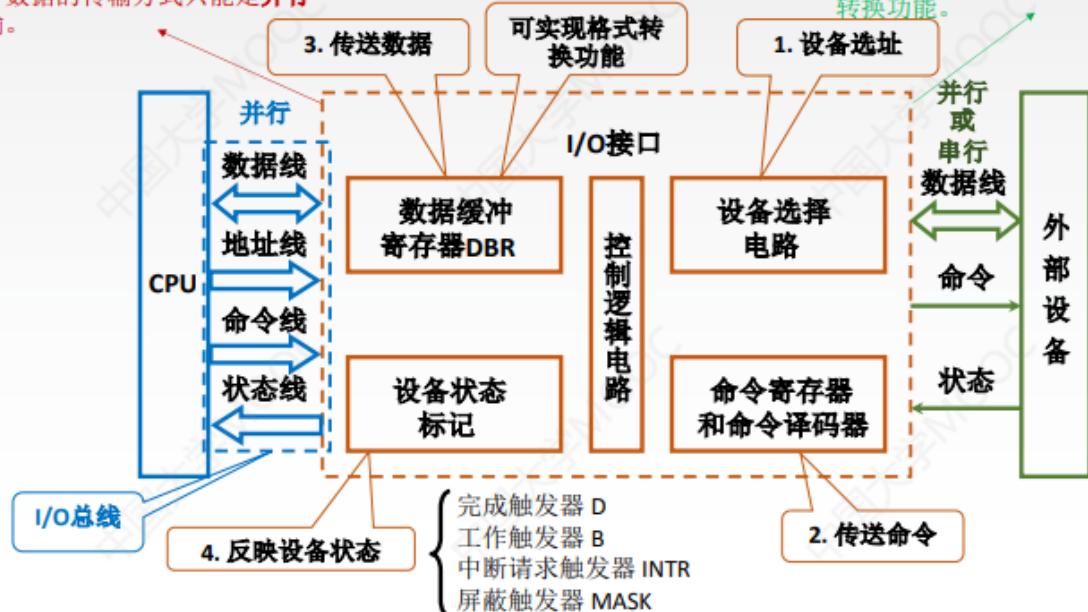
带宽=分辨率 x 灰度级位数*帧频



内部接口：内部接口与系统总线相连，实质上是与内存、CPU相连。数据的传输方式只能是并行传输。

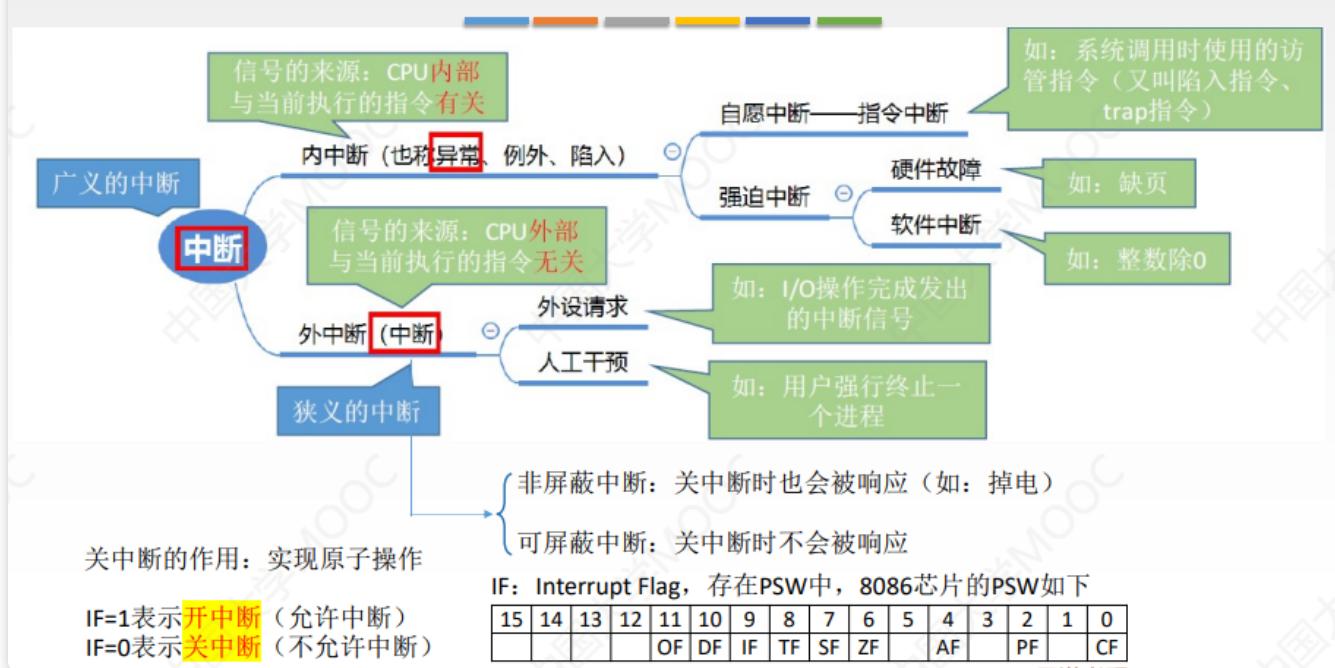
I/O接口的基本结构

外部接口：外部接口通过接口电缆与外设相连，外部接口的数据传输可能是串行方式，因此I/O接口需具有串/并转换功能。



独占查询和定时查询

中断请求的分类



中断响应，中断请求，中断处理

中断优先级

1. 硬件故障中断属于最高级，其次是软件中断；
2. 非屏蔽中断优于可屏蔽中断；
3. DMA请求优于I/O设备传送的中断请求
4. 高速设备优于低速设备；

5. 输入设备优于输出设备；
6. 实时设备优于普通设备。

中断隐指令的主要任务

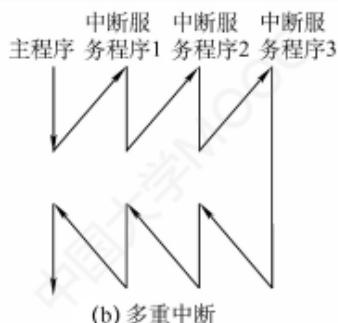
- ① 关中断。在中断服务程序中，为了保护中断现场（即CPU主要寄存器中的内容）期间不被新的中断所打断，必须关中断，从而保证被中断的程序在中断服务程序执行完毕之后能接着正确地执行下去。
- ② 保存断点。为了保证在中断服务程序执行完毕后能正确地返回到原来的程序，必须将原来程序的断点（即程序计数器（PC）的内容）保存起来。可以存入堆栈，也可以存入指定单元。
- ③ 引出中断服务程序。引出中断服务程序的实质就是取出中断服务程序的入口地址并传送给程序计数器（PC）

中断服务程序

中断服务程序的主要任务：

- ① 保护现场保存通用寄存器和状态寄存器的内容（eg：保存ACC寄存器的值），以便返回原程序后可以恢复CPU环境。可使用堆栈，也可以使用特定存储单元。
- ② 中断服务(设备服务)
主体部分，如通过程序控制需打印的字符代码送入打印机的缓冲存储器中（eg：中断服务的过程中有可能修改ACC寄存器的值）
- ③ 恢复现场
通过出栈指令或取指令把之前保存的信息送回寄存器中（eg：把原程序算到一般的ACC值恢复原样）
- ④ 中断返回
通过中断返回指令回到原程序断点处。

单重中断与多重中断



	单重中断	多重中断
中 断 隐 指 令	关中断	关中断
	保存断点 (PC)	保存断点 (PC)
	送中断向量	送中断向量
中 断 服 务 程 序	保护现场	保护现场和屏蔽字
	-	开中断
	执行中断服务程序	执行中断服务程序
	-	关中断
	恢复现场	恢复现场和屏蔽字
	开中断	开中断
	中断返回	中断返回

中断屏蔽技术

中断屏蔽字

屏蔽字设置的规律：

- 1.一般用'1表示屏蔽，"表示正常申请。
- 2.每个中断源对应一个屏蔽字（在处理该中断源的中断服务程序时，屏蔽寄存器中的内容为该中断源对应的屏蔽字）。
- 3.屏蔽字中'*1越多，优先级越高。每个屏蔽字中至少有一个1'（至少要能屏蔽自身的中断）。

程序执行轨迹

DMA方式与中断方式

	中断	DMA
数据传送	程序控制 程序的切换 → 保存和恢复现场	硬件控制 CPU只需进行预处理和后处理
中断请求	传送数据	后处理
响应	指令执行周期结束后响应中断	每个机器周期结束均可，总线空闲时即可响应DMA请求
场景	CPU控制，低速设备	DMA控制器控制，高速设备
优先级	优先级低于DMA	优先级高于中断
异常处理	能处理异常事件	仅传送数据

DMA传送方式

主存和DMA控制器之间有一条数据通路，因此主存和I/O设备之间交换信息时，不通过CPU。但当I/O设备和CPU同时访问主存时，可能发生冲突，为了有效地使用主存，DMA控制器与CPU通常采用以下3种方法使用主存。

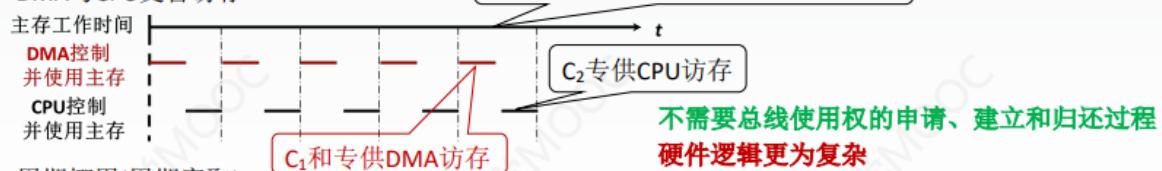
(1) 停止CPU访问主存



控制简单

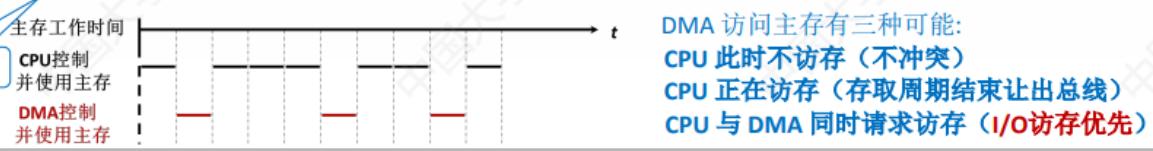
CPU 处于不工作状态或保持状态
未充分发挥 CPU 对主存的利用率

(2) DMA与CPU交替访存



不需要总线使用权的申请、建立和归还过程
硬件逻辑更为复杂

(3) 周期挪用(周期窃取)



DMA 访问主存有三种可能:

CPU 此时不访存 (不冲突)

CPU 正在访存 (存取周期结束让出总线)

CPU 与 DMA 同时请求访存 (I/O访存优先)

DMA方式具有下列特点：

- ①它使主存与CPU的固定联系脱钩，主存既可被CPU访问，又可被外设访问。
- ②在数据块传送时，主存地址的确定、传送数据的计数等都由硬件电路直接实现。
- ③主存中要开辟专用缓冲区，及时供给和接收外设的数据。
- ④ DMA传送速度快，CPU和外设并行工作，提高了系统效率。

DMA在传送开始前要通过程序进行预处理，结束后要通过中断方式进行后处理。