0 思维导图

https://file.moluuser.com/mind/co.html

1 概述

发展历程

软件

系统软件

- 用来管理整个计算机系统

应用软件

- 按任务需要编制成的各种程序

硬件

- 第一代：电子管时代
- 第二代：晶体管时代
- 第三代：中小规模集成电路时代
- 第四代：大规模、超大规模集成电路时代

基本组成

五大部分

输入设备

- 将信息转换成机器能识别的形式

输出设备

- 将结果转换成人们熟悉的形式

主存储器

- 存放数据和程序

运算器

- 算术运算、逻辑运算

控制器

- 指挥各部件，使程序运行

冯诺依曼结构

- 首次提出“存储程序”概念
- 以运算器为中心

现代计算机结构

- 以存储器为中心
- CPU = 运算器 + 控制器

硬件概述

主存

存储体

- 存储单元
  - 每个存储单元存放一串二进制代码
- 存储字(word)
  - 存储单元中二进制代码的组合
- 存储字长
  - 存储单元中二进制代码的组合
- 存储元
  - 存储二进制的电子元件，每个存储元可存1bit
- 地址

MAR

- 地址寄存器，用于指明要读/写哪个存储单元。其位数反映存储单元数量

MDR

- 数据寄存器，用于暂存要读/写的数据。其位数=存储字长

运算器

ACC

- 累加计数器，存放操作数、运算的结果

MQ

- 乘商寄存器，进行乘、除法时用得到

X

- 通用寄存器，存放操作数

ALU

- 算数逻辑单元，用电路实现各种算数运算、逻辑运算

控制器

PC

- 程序计数器，存放下一条指令的地址

IR

- 指令寄存器，存放当前执行的指令

CU

- 控制单元，分析指令，给出控制信号

工作过程

- 初始：指令、数据存入主存，PC指向第一条指令
- 从主存中取指令放入IR、PC自动加1、CU分析指令、CU指挥其他部件执行指令

层次结构

五层

- M4：高级语言机器(执行高级语言)
- M3：汇编语言机器(执行汇编语言)
- M2：操作系统机器(向上提供广义指令)
- M1：传统机器(执行机器语言指令)
- M0：微程序机器(执行微指令)

三个级别的语言

- 高级语言、汇编语言、机器语言
- 编译程序：将高级语言一次全部翻译为汇编语言，或直接翻译为机器语言
- 汇编程序：将汇编语言翻译成机器语言
- 解释程序：高级语言翻译为机器语言(翻译一句执行一句)

性能指标

存储器容量

- MAR的位数反应存储单元数量
- MDR反应每个存储单元大小

CPU

时钟周期

- CPU中最小的时间单位，每个动作至少要1个时钟周期

主频

- $=\frac{1}{\text{时钟周期}}$，单位：Hz

CPI

- 执行一条指令所需的时钟周期数

CPU执行时间

- 运行一个程序所花费的时间
- $=\frac{\text{指令条数}\times CPI}{\text{主频}}$

IPS

- $=\frac{\text{主频}}{\text{平均}CPI}$，每秒执行多少条指令

FLOPS

- 每秒执行多少次浮点运算

其他

- 数据通路宽度
  - 数据总线一次所能并行传送信息的位数
- 吞吐量
  - 指系统在单位时间内处理请求的数量
- 响应时间
  - 指从用户向计算机发送一个请求，到系统对该请求做出响应并获得它所需要结果的等待时间
- 基准程序
  - 测量计算机处理速度的一种程序

常用数量单位

- 描述存储容量、文件大小时：$K=2^{10}, M=2^{20}, G=2^{30}, T=2^{40}$
- 描述频率、速率时：$K=10^3, M=10^6, G=10^9, T=10^{12}$

2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
4096	2048	1024	512	256	128	64	32	16	8	4	2	1	0.5	0.25	0.125

2 数据的表示和运算

进位计数制

r进制

- 基数=r，每个数码位可能出现r种字符。逢r进1

r进制->十进制

- r进制数的数值=各数码位与位权的乘积之和

二进制<-->八进制

- 每3个二进制位对应一个八进制位

二进制<-->十六进制

- 每4个二进制位对应一个十六进制位

十进制<-->r进制

- 整数部分：除基取余法，先取得的“余”是整数的低位
- 小数部分：乘基取整法，先取得的“整”是小数的高位

真值和机器数

- 真值：实际的带正负号的数值(人类习惯的样子)
- 机器数：把正负号数字化的数(存到机器里的样子)

BCD码

8421码

- 每4个二进制位对应一个十进制位(有6个冗余状态)
- 8、4、2、1分别对应每一位的权值
- 0000~1001分别对应0~9，进行加法后若超出该范围，则需+0110进行修正(强制向高位进1)

余3码

- 8421码 + 0011

2421码

- 2、4、2、1分别对应每一位的权值
- 表示0~4时最高位为0，表示5~9时最高位为1

字符与字符串

ASCII码

- 通常用8bit表示一个字符，最高位都为0
- 共128个字符。0~31、127为控制/通信字符，32~126为可印刷字符
- 所有大写字母、所有小写字母、所有数字的编码都连续

汉字

- 区位码、国标码、汉字内码、输入编码、字形码
- 国标码 = 区位码 + 2020H
- 机内码 = 国标码 + 8080H

字符串

- 从低地址到高地址逐个字符存储，常采用'\0'作为结尾标志
- 对于多字节的数据(如汉字)，可采取大/小端存储模式
- 大端模式：将数据的最高有效字节存放在低地址单元中
- 小端模式：将数据的最高有效字节存放在高地址单元中

奇偶校验码

校验原理

- 码字间的距离：两个码字之间有几个位不同
- 码距：一个编码方案中，合法码字间的最小距离
- 若码距=2，有检错能力；若码距≥3，可能还有纠错能力

奇偶校验

- 在信息位的首部或尾部添加一个奇偶校验位
- 奇校验：整个校验码(信息位和校验位)中“1”的个数为奇数
- 偶校验：整个校验码(信息位和校验位)中“1”的个数为偶数
- 奇偶校验码的码距d=2，仅能检测出奇数位错误，无纠错能力

异或运算(模二加)

- 同0异1

海明校验码

基本思想

- 分组偶校验，多个校验位可反映出错位置

求解步骤

- 确认校验位个数
- 确认校验位分布
- 求校验位
- 纠错

其他

- 海明码有1位纠错，2位检错能力
- 为了区分1位错和2位错，还需添加“全校验位”对整体进行偶校验
- 注意：有的题目位置编号可能是从小到大的，但处理方法雷同

循环冗余校验码

构造

- 由生成多项式确定“除数”。若生成多项式中x的最高次为R，则“除数”有R+1位
- K个信息位+R个0，作为“被除数”
- 被除数、除数进行“模二除”，得R位余数
- K个信息位+R位余数=CRC码

校验

- 收到K+R位数据，与生成多项式模二除，计算R位余数
- 余数为0，说明无错误
- 余数非0，说明出错

检错纠错能力

- 可检测出所有奇数个错误
- 可检测出所有双比特的错误
- 可检测出所有小于等于校验位长度的连续错误
- 若选择合适的生成多项式，且$2^R \geq K + R + 1$，则可纠正单比特错

定点数的表示

无符号数

- 整个机器字长的全部二进制位均为数值位，没有符号位，相当于数的绝对值。
- n位的无符号数的表示范围为$0 \sim 2^n - 1$

有符号数

概念

- 可用原码、反码、补码三种方式来表示定点整数和定点小数。还可用移码表示定点整数。
- 若机器字长为n+1位，则尾数占n位。
- 若真值为x，则用$[x]_{原}, [x]_{反}, [x]_{补}, [x]_{移}$分别表示真值所对应的原码、反码、补码、移码

原码

- 符号位“0/1”对应“正/负”，用尾数表示真值的绝对值
- 真值0有 +0 和 -0 两种形式
- 若机器字长n+1位，原码整数的表示范围：$-(2^n - 1) \leq x \leq 2^n - 1$
- 若机器字长n+1位，原码小数的表示范围：$-(1 - 2^{-n}) \leq x \leq 1 - 2^{-n}$

反码

- 若符号位为0(正数)，则反码与原码相同
- 若符号位为1(负数)，则数值位全部取反
- 表示范围和原码相同

补码

- 正数的补码 = 原码
- 负数的补码 = 反码末位 +1(要考虑进位)
- 补码的真值0只有一种表示形式$[+0]_{补}=[-0]_{补}=00000000$
- 定点整数补码$[x]_{补}=1,0000000$ 表示 $x=-2^7$
- 若机器字长n+1位，补码整数的表示范围：$-2^n \leq x \leq 2^n-1$ (比原码多表示一个-2^n)
- 定点小数补码$[x]_{补}=1,0000000$ 表示 $x=-1$
- 若机器字长n+1位，补码小数的表示范围：$-1 \leq x \leq 1-2^{-n}$ (比原码多表示一个-1)

移码

- 补码的基础上将符号位取反
- 注意：移码只能用于表示整数
- 移码整数的表示范围与补码相同
- 移码全0真值最小，移码全1真值最大(移码表示的整数很方便对比大小)

其他

- $[x]_{补}$与$[-x]_{补}$，符号位取反，数值位取反加1
- 原码和反码的真值0有两种表示；补码和移码的真值0只有一种表示
- 负数补—>原：①数值位取反+1；②负数补码中，最右边的1及其右边同原码。最右边的1的左边同反码(数值位最靠右的1左侧取反码)

定点数的运算

		加	减	乘	除
Accumulator	ACC	被加数、和	被减数、差	乘积高位	被除数、余数
Multiple-Quotient Register	MQ			乘数、乘积低位	商
Arithmetic and Logic Unit	X	加数	减数	被乘数	除数

移位运算

- 通过改变各个数码位和小数点的相对位置，从而改变各数码位的位权。可用移位运算实现乘法、除法
- 左移相当于×2；右移相当于÷2
- 由于原、反、补码位数有限，因此某些时候算数移位不能精确等效乘法、除法

算数移位

- 原码的算数移位
  - 符号位保持不变，仅对数值位进行移位
  - 右移：高位补0，低位舍弃。若舍弃的位=0，则相当于÷2；若舍弃的位≠0，则会丢失精度
  - 左移：低位补0，高位舍弃。若舍弃的位=0，则相当于×2；若舍弃的位≠0，则会出现严重误差

- 反码的算数移位
  - 正数的反码与原码相同
  - 负数的反码数值位与原码相反：右移：高位补1，低位舍弃。左移：低位补1，高位舍弃。
- 补码的算数移位
  - 正数的补码与原码相同
  - 负数补码的算数移位
    - 负数补码 = 反码末位+1导致反码最右边几个连续的1都因进位而变为0，直到进位碰到第一个0为止
    - 负数补码中，最右边的1及其右边同原码。最右边的1的左边同反码负数补码的算数移位规则如下：
    - 右移(同反码)：高位补1，低位舍弃。左移(同原码)：低位补0，高位舍弃。

逻辑移位

- 左移、右移都补0，移出的位舍弃

循环移位

- 不带进位位
  - 用移出的位补上空缺
- 带进位位
  - 移出的位放到进位位，原进位位补上空缺

加减运算

原码

- 加法
  - 同号相加
    - 数值部分 = 被加数、加数的绝对值进行相加
    - 符号位：不变
  - 异号相加
    - 数值部分 = 被加数、加数中，绝对值更大的减绝对值更小的
    - 符号位：与绝对值更大的数相同
- 减法
  - 将减数取负，转变为加法

补码的加减法

- 符号位参与运算，直接相加

溢出判断(补码)

- 只有“正数+正数”才会上溢——正+正=负
- 只有“负数+负数”才会下溢——负+负=正
- 三种方法
  - 两个数的符号位和运算结果的符号位判断表达式
  - 符号位的进位和最高数值位的进位异或
  - ☆双符号位，正数符号为00，负数符号为11。01上溢，10下溢

- (双符号位补码又称模4补码，单符号位补码又称模2补码)
- (存储时只有一个符号位，运算时复制一个符号位)

符号扩展

- 解决短数据到长数据多余位的填补问题
- 定点整数的符号扩展：在原符号位和数值位中间添加新位，正数都添0;负数原码添0，负数反、补码添1
- 定点小数的符号扩展：在原符号位和数值位后面添加新位，正数都添0;负数原、补码添0，负数反码添1

乘法运算

- 原码一位乘法：先加法再逻辑移位，重复n次；符号位异或
- 补码一位乘法(Booth算法)：符号位、数值位都是由被乘数和乘数进行n轮加法、算术右移，最后再多来一次加法，乘数的符号位参与运算

除法运算

- 原码除法：恢复余数法
- 原码除法：加减交替法
- 补码除法：加减交替法

除法类型	符号位参与运算	加减次数	移位		上商、加减原则	说明
			方向	次数		
原码加减交替法	否	N+1或N+2	左	N	余数的正负	若最终余数为负，需恢复余数
补码加减交替法	是	N+1	左	N	余数和除数是否同号	商末位恒置1

强制类型转换

- 无符号数与有符号数：不改变数据内容，改变解释方式。
- 长整数变短整数：高位截断，保留低位。
- 短整数变长整数：符号扩展。

数据的存储和排列

- 最高有效字节(MSB)
- 最低有效字节(LSB)
- 多字节数据在内存里一定是占连续的几个字节
- 大端方式：便于人类阅读
- 小端方式：便于机器处理

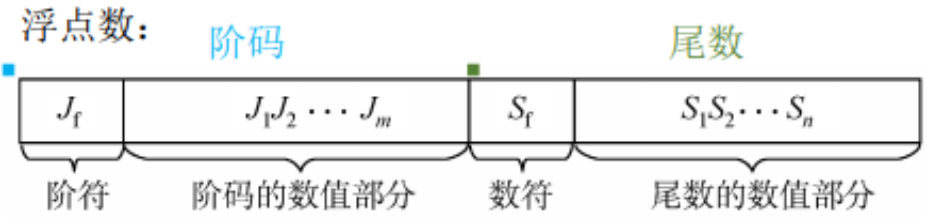
边界对齐

- 现代计算机通常是按字节编址，即每个字节对应1个地址
- 通常也支持按字、按半字、按字节寻址(逻辑左移转化为字节编址)
- 假设存储字长为32位，则1个字=32bit，半字=16bit。每次访存只能读/写1个字

浮点数的表示和运算

浮点数的表示

表示

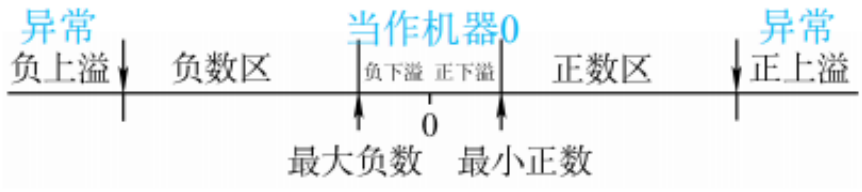


- 尾数给出具体数值，阶码指明小数点前移、后移多少位
- 阶码：常用补码或移码表示的定点整数
- 尾数：常用原码或补码表示的定点小数
- 阶码E反映浮点数的表示范围及小数点的实际位置
- 尾数M的数值部分的位数n反映浮点数的精度
- 真值
  - $N = r^E \times M$

规格化

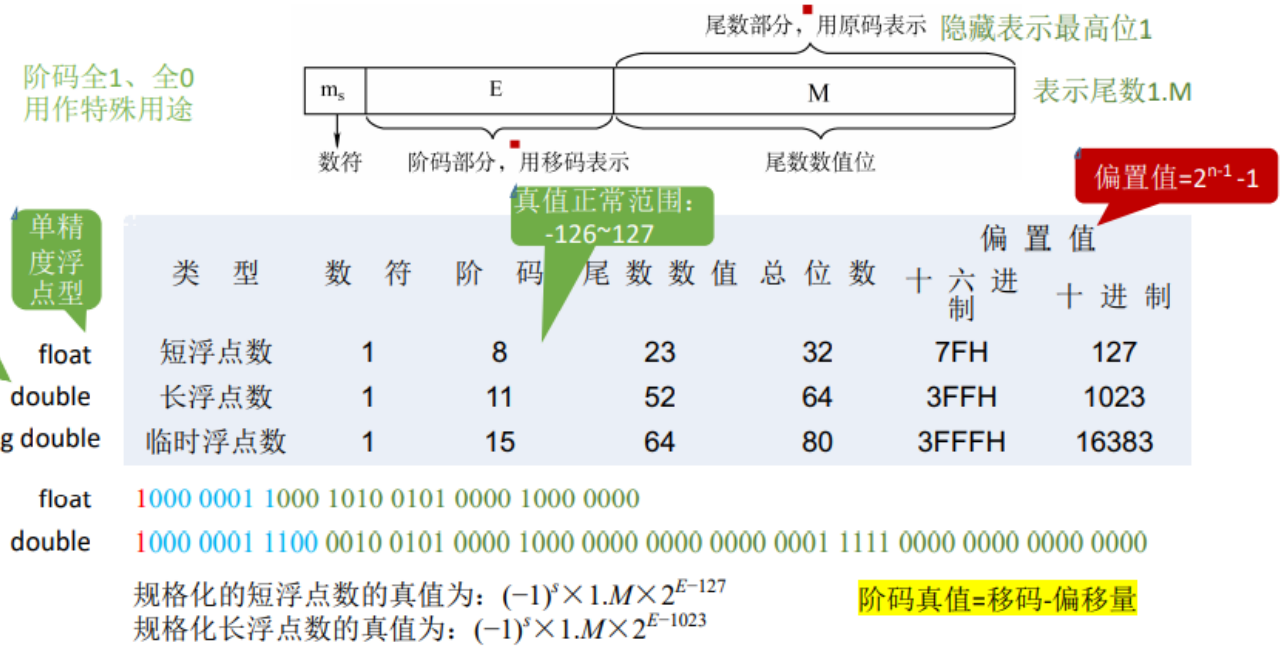
- 规格化浮点数：规定尾数的最高数值位必须是一个有效值
- 左规：数值位最高位无效时，通过尾数算数左移、阶码减1的方法处理，直到尾数最高数值位有效时停止
- 右规：若采用双符号位表示尾数，则当运算后尾数“假溢出”时，可以通过尾数右移、阶码加1的方法处理
- 原码表示的尾数规格化：尾数的最高数值位必须是1
- 补码表示的尾数规格化：尾数的最高数值位必须和尾数符号位相反

※表示范围



IEEE 754

- 移码：补码的基础上将符号位取反。注意：移码只能用于表示整数
- 移码的定义：移码 = 真值 + 偏置值(此处8位移码的偏置值 = 128D = 1000 0000B，即2^{n-1})
- IEEE 754中的偏置值 = 127D = 0111 1111B，即$2^{n-1}-1$



浮点数的运算

加减运算

1. 对阶

- 小阶向大阶看齐(可能导致丢失末位精度)

2. 尾数加减

- 通常采用双符号位表示尾数，挽救尾数溢出

3. 规格化

- 左规：尾数最高数值位为无效位时，尾数左移，阶码减1
- 右规：尾数双符号位不同时，尾数右移，阶码加1

4. 舍入

- 0舍1入法：在尾数右移时，被移去的最高数值位为0，则舍去；被移去的最高数值位为1，则在尾数的末位加1
- 恒置1法：尾数右移时，不论丢掉的最高数值位是1还是0，都使右移后的尾数末位恒置1

5. 判断溢出

- 阶码上溢
  - 抛出异常(中断)
- 阶码下溢
  - 按机器0处理

强制类型转换

类型	16位机器	32位机器	64位机器
char	8	8	8
short	16	16	16
int	16	32	32
long	32	32	64
long long	64	64	64
float	16	32	32
double	64	64	64

在32位机器的条件下

- int：1 + 32
- float：1 + 8 + 23

无损

- char -> int -> long -> double
- float -> double

有损

- int -> float
    - 可能会损失精度
- float -> int
    - 可能会溢出或损失精度

算数逻辑单元

ALU

- 实现算术运算、逻辑运算、辅助功能(移位、求补等)
- 基本结构：输入、输出、控制

电路知识

- 优化逻辑表达式就是在优化电路

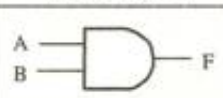
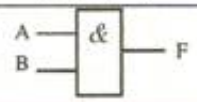
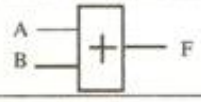
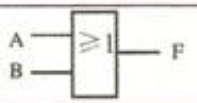
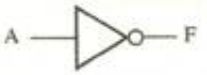
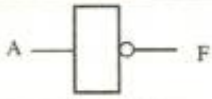
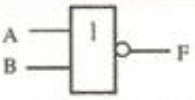
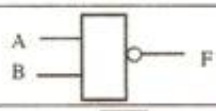
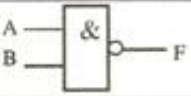
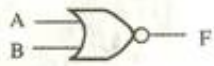
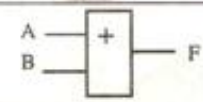
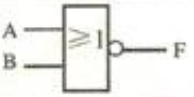
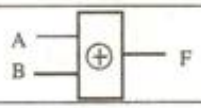
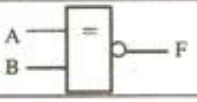
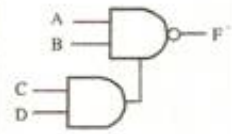
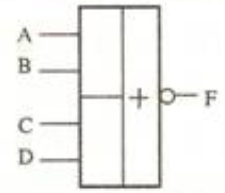
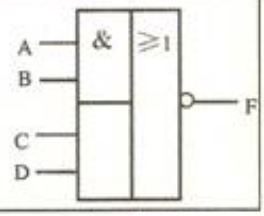
逻辑运算

- 与、或、非
- 复合逻辑：与非、或非、异或、同或

门电路

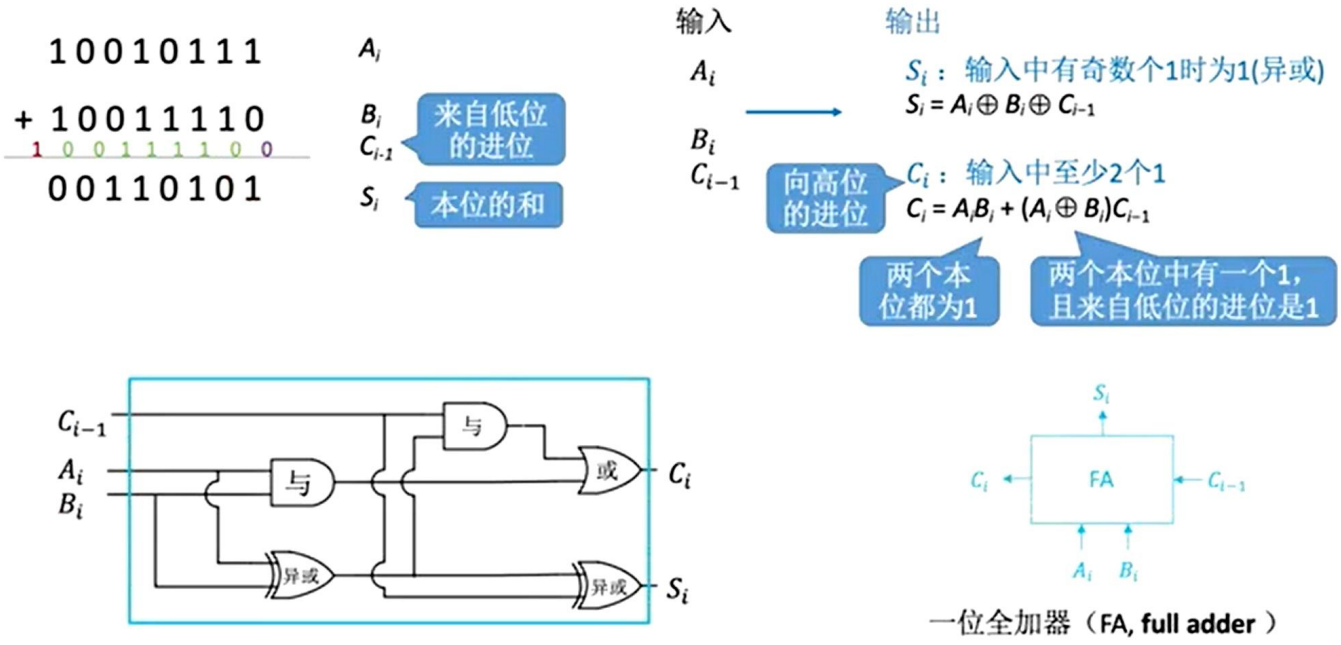
- 最基本的逻辑元件、用于实现逻辑运算

门电路图形

逻 辑 门	IEEE 推荐符号	我国部颁符号	输出表达式	标 准 符 号
"与"门			F=A·B	
"或"门			F=A+B	
"非"门			$F=\overline{A}$	
"与非"门			$F=\overline{A \cdot B}$	
"或非"门			$F=\overline{A+B}$	
"异或"门			$F=A \oplus B$ $=A\overline{B}+\overline{A}B$	
"与或非"门			$F=\overline{AB+CD}$	

加法器的实现

一位全加器



一位全加器（FA, full adder）

- 本位和
- 本位向高位的进位

串行加法器

- 只有一个全加器，数据逐位串行送入加法器中进行运算。
- 进位触发器用来寄存进位信号，以便参与下一次运算。

串行进位的并行加法器

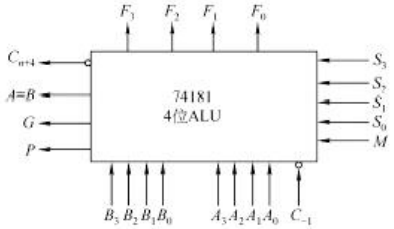
- 多个全加器简单串联，可多位同时加
- 计算速度取决于进位产生和传递的速度

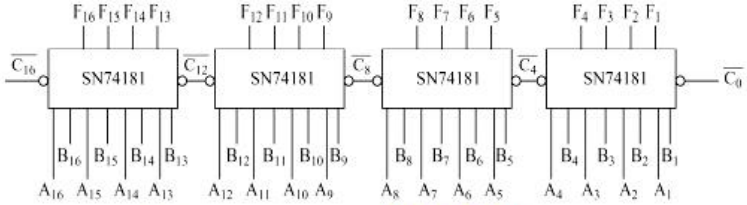
加法器与ALU的改进

并行加法器的优化

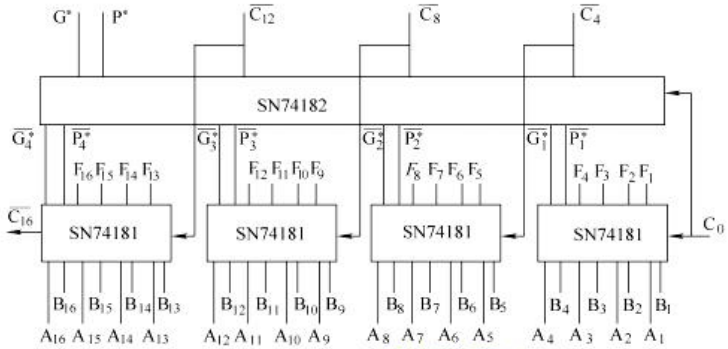
- 并行进位的并行加法器：各级进位信号同时形成，又称为先行进位、同时进位
- 单级先行进位方式，又称为组内并行、组间串行进位方式
- 多级先行进位方式，又称为组内并行、组间并行进位方式

ALU芯片的优化





16位的组内并行、组间串行进位ALU



16位的组内并行、组间并行进位ALU

3 存储系统

概念

层次结构

- 高速缓存(Cache)
- 主存储器(主存、内存)
- 辅助存储器(赋存、外存)(不能直接被CPU读写)

- Cache—主存：解决了主存与CPU速度不匹配的问题
- 主存—辅存：实现虚拟存储系统，解决了主存容量不够的问题

分类

按层次结构

按存储介质

- 半导体存储器
- 磁表面存储器
- 光存储器

按存取方式

- 随机存取存储器(RAM)
  - 读写任何一个存储单元所需时间都相同，与存储单元所在的物理位置无关
- 顺序存取存储器(SAM)
  - 读写一个存储单元所需时间取决于存储单元所在的物理位置
  - 例如：磁带
- 直接存取存储器(DAM)

    - 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，然后按顺序方式存取
    - 例如：HDD
- 相联存储器(CAM)
    - 可以按照内容检索到存储位置进行读写
    - “快表”就是一种相联存储器
- 串行访问存储器：包括SAM和DAM。读写某个存储单元所需时间与存储单元的物理位置有关

按信息可更改性

- 读写存储器(Read/Write Memory)
- 只读存储器(Read Only Memory)

断电后信息是否消失

- 易失性存储器(主存、Cache)：断电后，存储信息消失的存储器
- 非易失性存储器(磁盘、光盘)：断电后，存储信息依然保持的存储器

- 破坏性读出(如DRAM芯片，读出数据后要进行重写)：信息读出后，原存储信息被破坏
- 非破坏性读出(如SRAM芯片、磁盘、光盘)：信息读出后，原存储信息不被破坏

存储器性能指标

- 存储容量：存储字数 × 字长(如1M×8位)
- 单位成本：每位价格 = 总成本 / 总容量
- 存储速度
    - 数据传输率(主存带宽) = 数据的宽度 / 存储周期
    - 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)
- 存储周期 = 存取时间 + 恢复时间

主存储器的基本组成

基本元件

- MOS管，作为通电开关
- 电容，存储电荷

存储芯片的结构

- 译码驱动电路
    - 译码器将地址信号转化为字选通线的高低电平
- 存储矩阵(存储体)
    - 由多个存储单元构成，每个存储单元又由多个存储元构成
- 读写电路
    - 每次读/写一个存储字
- 地址线、数据线、片选线、读写控制线(可能两根或一根)

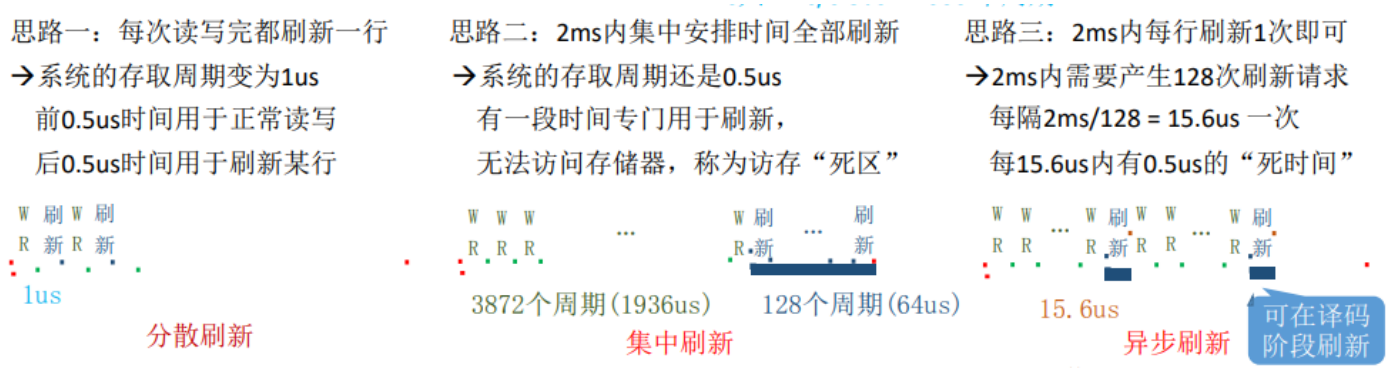
寻址

- 现代计算机通常按字节编址(每个字节)，即每个字节对应一个地址
- 按字节寻址、按字寻址、按半字寻址、按双字寻址

SRAM和SDAM

	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

- SRAM
  - Static Random Access Memory
  - 双稳态触发器
  - 常用作Cache
- DRAM
  - Dynamic Random Access Memory
  - 栅极电容
  - 常用作主存
  - 读出后需要重写
  - DRAM的地址线复用技术：行、列地址分两次送，可使地址线更少，芯片引脚更少
  - 电容里的电荷只能维持2ms，需要刷新(给电容充电)
  - 刷新
    - 分散刷新：每次读写完都刷新一行
    - 集中刷新：2ms内集中安排时间全部刷新
    - 异步刷新：2ms内每行刷新1次即可
  - 刷新由存储器独立完成，不需要CPU控制



只读存储器ROM

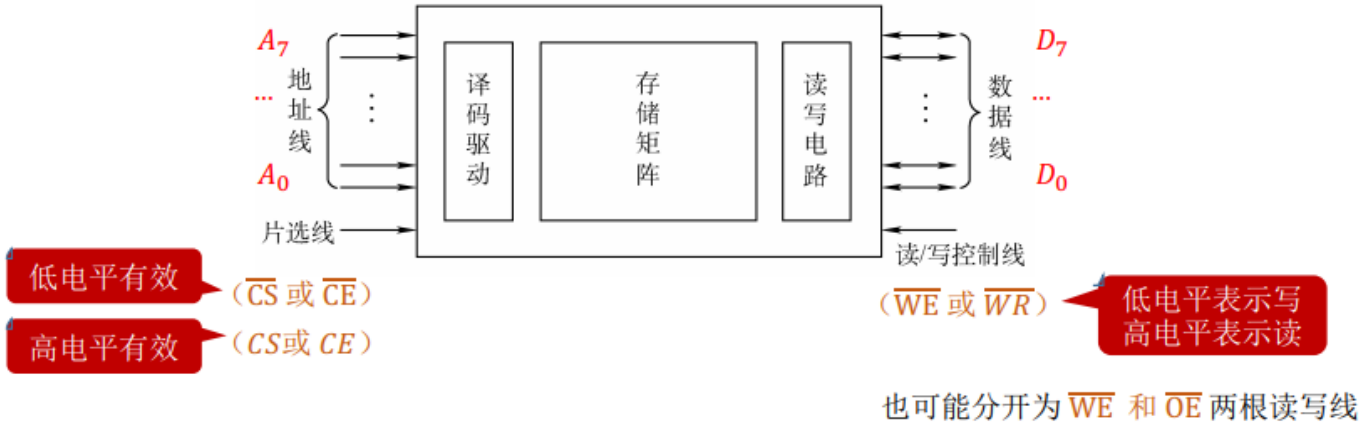
- MROM(Mask Read-Only Memory)
  - 掩模式只读存储器
  - 厂家按照客户需求，在芯片生产过程中直接写入信息，之后任何人不可重写(只能读出)
  - 可靠性高、灵活性差、生产周期长、只适合批量定制
- PROM(Programmable Read-Only Memory)
  - 可编程只读存储器
  - 用户可用专门的PROM写入器写入信息，写一次之后就不可更改
- EPROM(Erasable Programmable Read-Only Memory)
  - 可擦除可编程只读存储器，允许用户写入信息，之后用某种方法擦除数据，可进行多次重写
  - UVEPROM(ultraviolet rays)：用紫外线照射8~20分钟，擦除所有信息
  - EEPROM(也常记为E^2PROM，第一个E是Electrically)：可用“电擦除”的方式，擦除特定的字
- Flash Memory
  - 闪速存储器(注：U盘、SD卡就是闪存)
  - 在EEPROM基础上发展而来，断电后也能保存信息，且可进行多次快速擦除重写
  - 注意：由于闪存需要先擦除在写入，因此闪存的“写”速度要比“读”速度更慢。
  - 每个存储元只需单个MOS管，位密度比RAM高
- SSD(Solid State Drives)
  - 固态硬盘
  - 由控制单元+存储单元(Flash 芯片)构成
  - 与闪速存储器的核心区别在于控制单元不一样，但存储介质都类似，可进行多次快速擦除重写

主存储器与CPU的连接

存储器芯片的基本结构

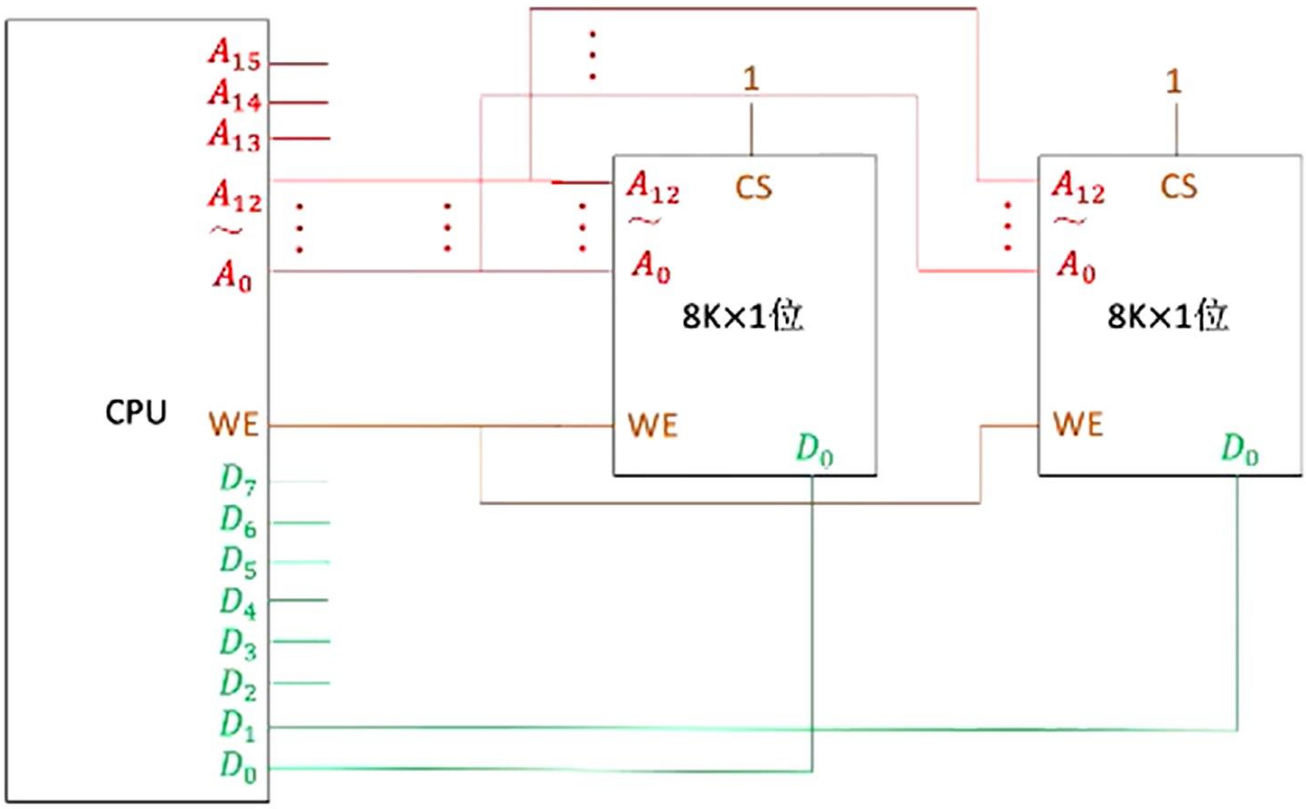
- 译码驱动电路
- 存储矩阵
- 读写电路
- 地址线、数据线、片选线、读写控制线

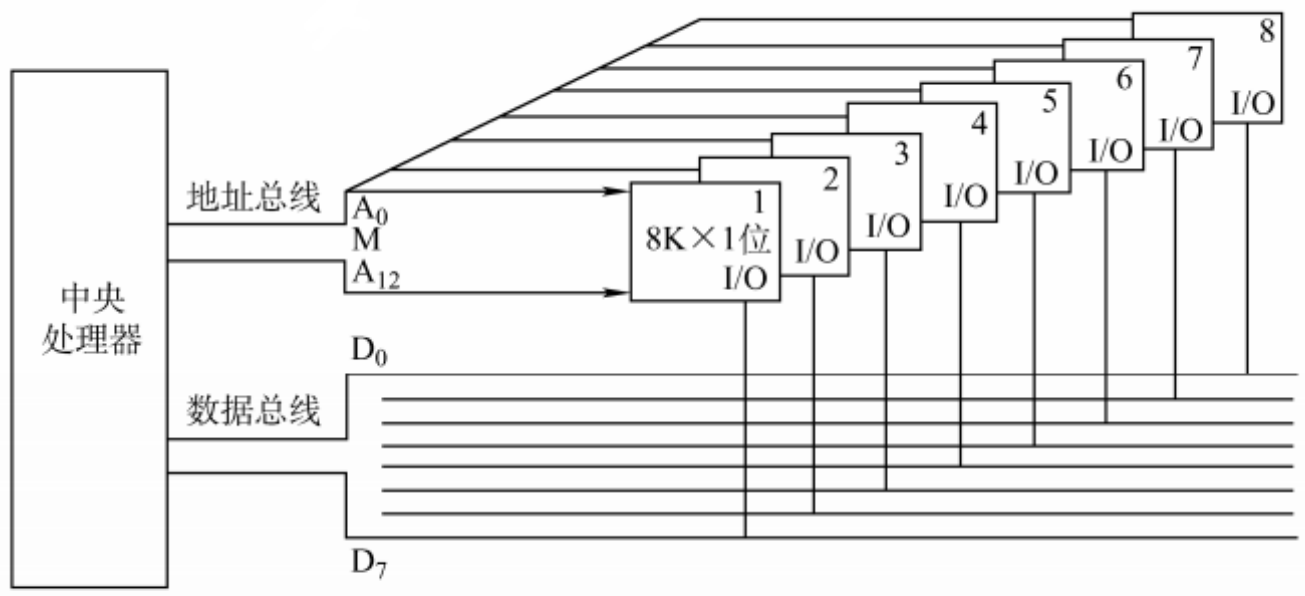
单块存储芯片与CPU的连接



多块存储芯片与CPU的连接

位扩展法





8片8K × 1位的存储芯片

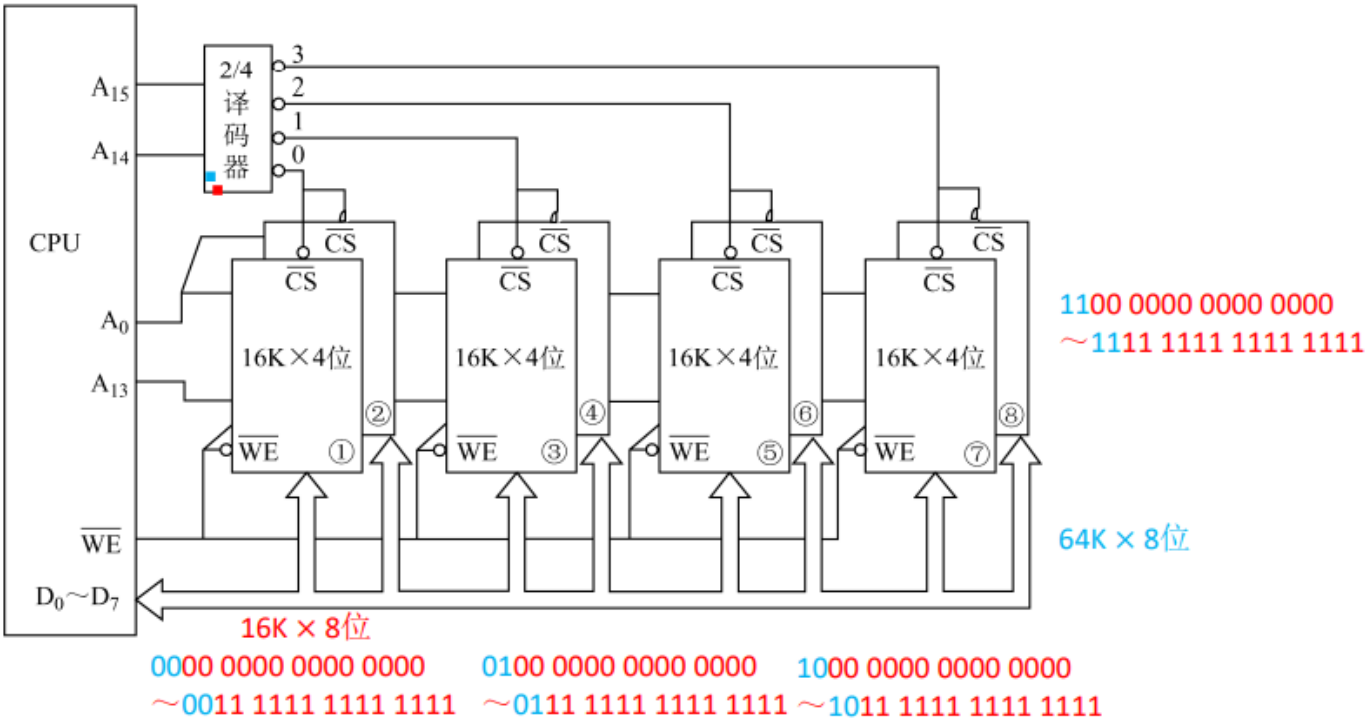
→ 1个8K × 8位的存储器，容量8KB

字扩展法

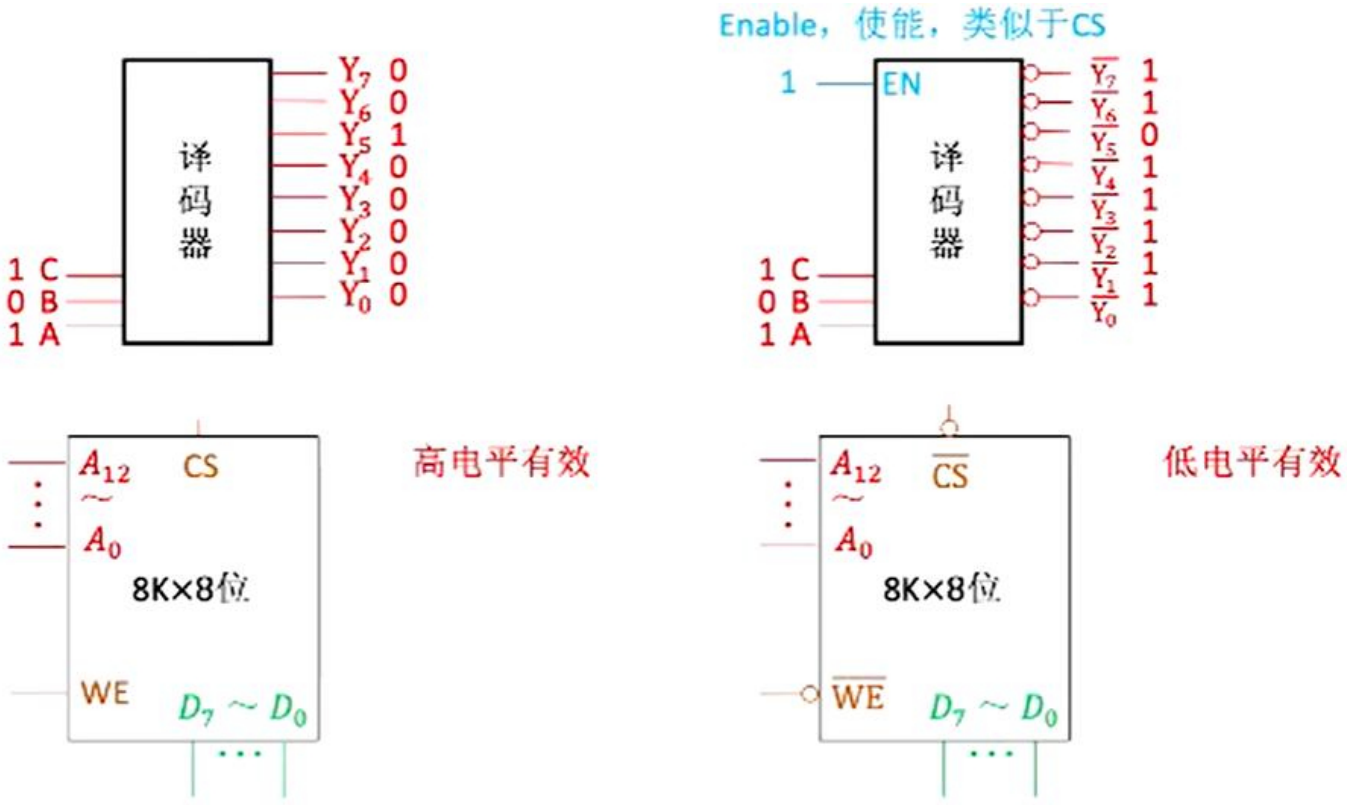
- 线选法
  - 地址线作为片选线
- 译码片选法
  - 例如1-2译码器

线选法	译码片选法
n条线→n个选片信号	n条线→ 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

字位扩展法

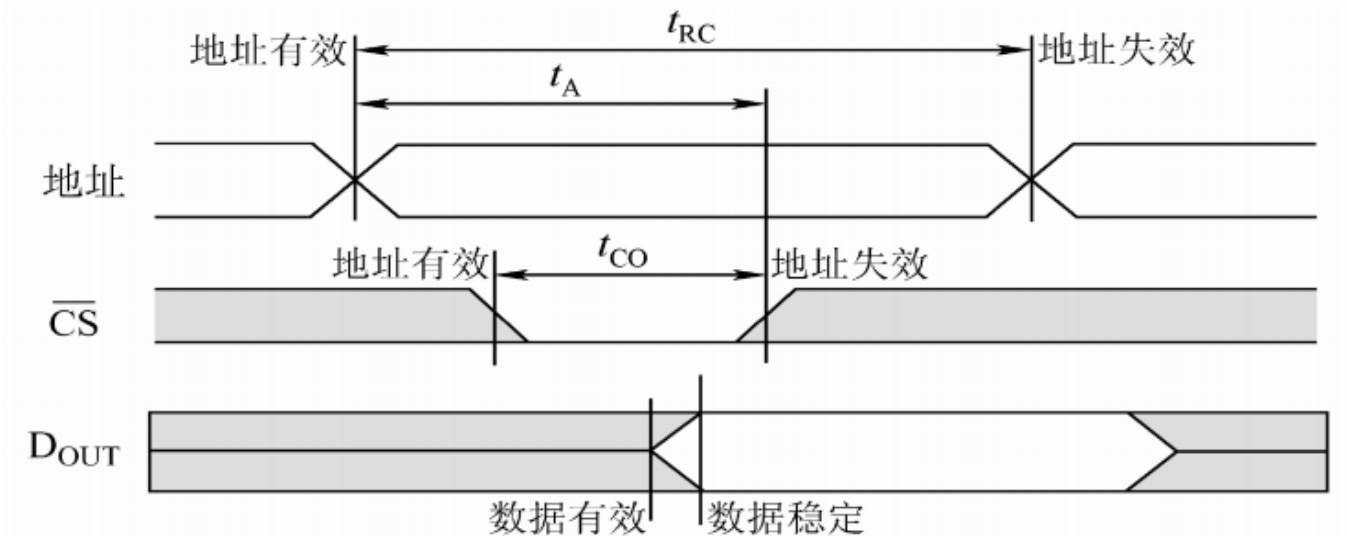


译码器



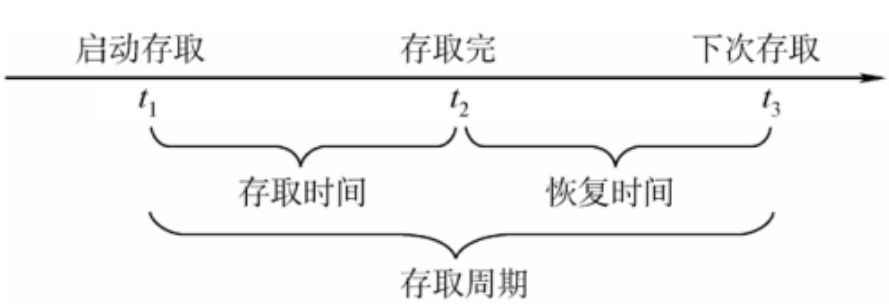
RAM的读周期

- CPU先送出地址信号
- 地址信号稳定后发出存储器请求信号
- CPU可使用译码器的使能端控制片选信号的生效时间



双口RAM与多模块存储器

存取周期



存取周期：可以连续读/写的最短时间间隔

注：DRAM芯片的恢复时间比较长，有可能是存取时间的几倍（SRAM的恢复时间较短）

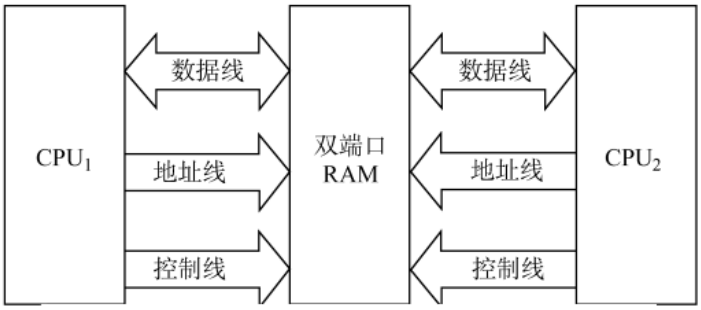
如：存取时间为 r，存取周期为 T，T=4r

- 可以连续读/写的最短时间间隔

双端口RAM

作用：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



解决方法：置“忙”信号为0，由判断逻辑决定暂时关闭一个端口（即被延时），未被关闭的端口正常访问，被关闭的端口延长一个很短的时间段后再访问。

两个端口对同一主存操作有以下4种情况：
1. 两个端口同时对不同的地址单元存取数据。☺
2. 两个端口同时对同一地址单元读出数据。☺
3. 两个端口同时对同一地址单元写入数据。☹写入错误
4. 两个端口同时对同一地址单元，一个写入数据，另一个读出数据。☹读出错误

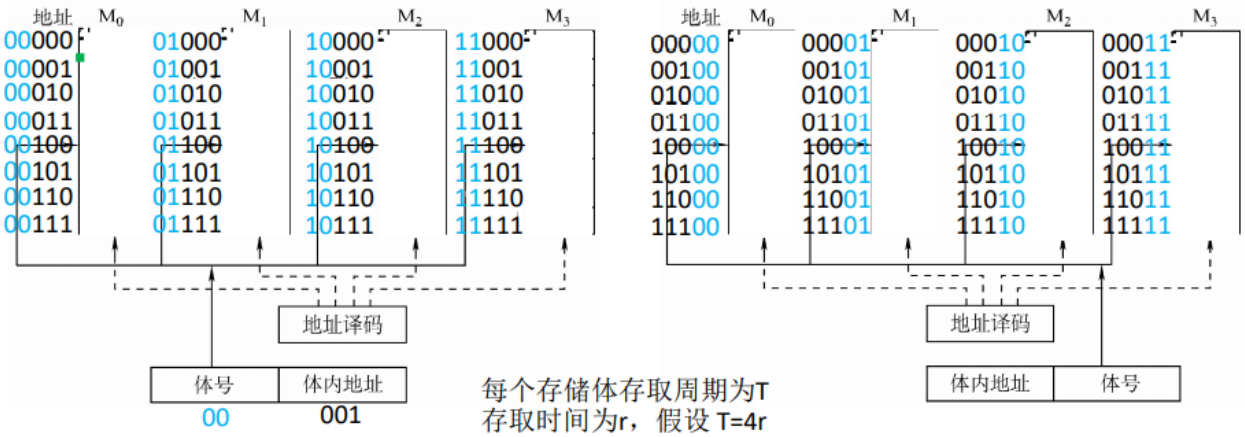
对比操作系统“读者-写者问题”

- 支持两个CPU同时访问RAM
- 可同时读/写不同的存储单元
  - 可同时读同一个存储单元
  - 不能同时写(或一读一写)同一个单元
- 若发生冲突，则发出BUSY信号，其中一个CPU的访问端口暂时关闭

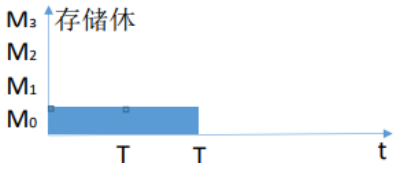
多模块存储器

可理解为“四根内存条”

多体并行存储器



高位交叉编址的多体存储器



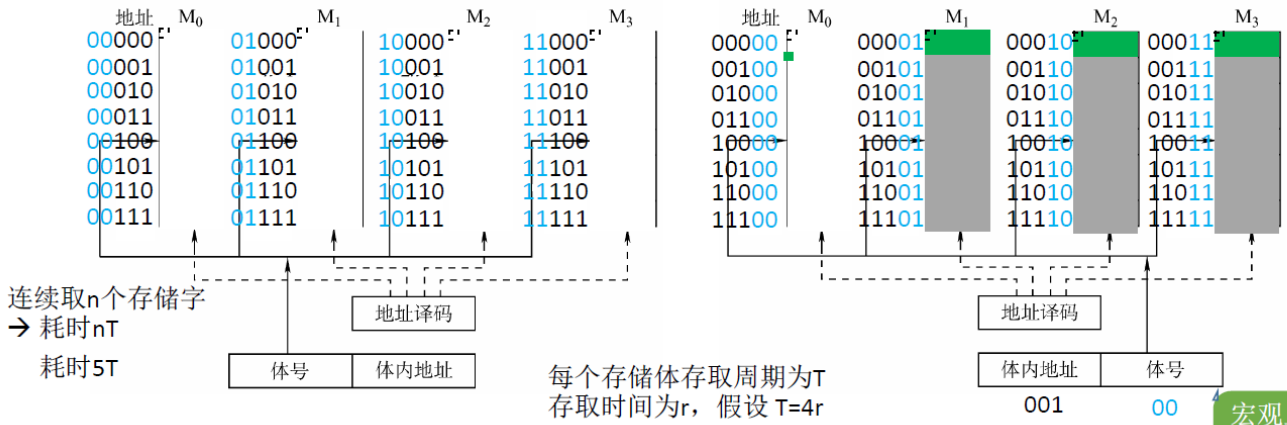
连续访问：
00000
00001
00010
00011
00100

低位交叉编址的多体存储器

多体并行存储器

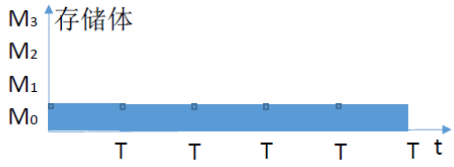
可理解为“四根内存条”

思考：为什么要探讨“连续访问”的情况？



宏观上读写一个字的时间接近 r

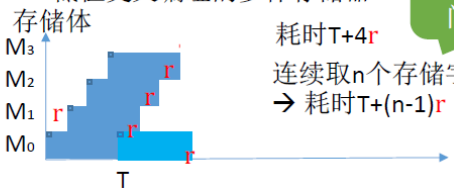
高位交叉编址的多体存储器



连续访问：

00000
00001
00010
00011
00100

低位交叉编址的多体存储器



耗时T+4r

连续取n个存储字
→ 耗时T+(n-1)r

应该取几个“体”？

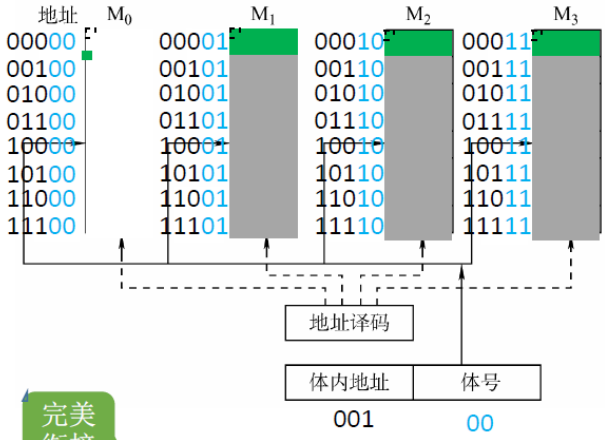
思考：给定一个地址 x，如何确定它属于第几个存储体？

采用“流水线”的方式并行存取（宏观上并行，微观上串行）
宏观上，一个存储周期内，m体交叉存储器可以提供的数据量为单个模块的m倍。

两种常见描述

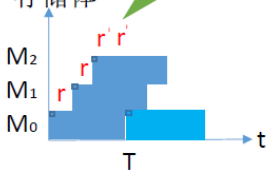
存取周期为T，存取时间为r，为了使流水线不间断，应保证模块数 m≥T/r

存取周期为T，总线传输周期为r，为了使流水线不间断，应保证模块数 m≥T/r



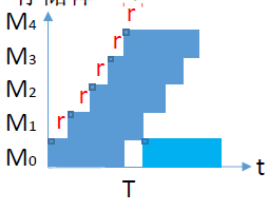
m<T/r

CPU需等待r



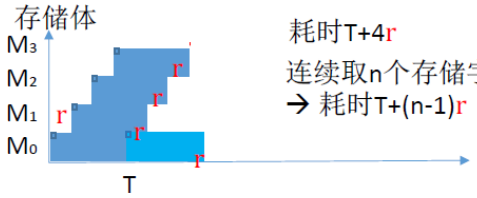
m>T/r

M0闲置r



完美衔接

低位交叉编址的多体存储器



耗时T+4r

连续取n个存储字
→ 耗时T+(n-1)r

单体多字存储器

- 每个存储单元存储m个字
- 总线宽度也为m个字
- 一次并行读出m个字
- 每次只能同时取m个字，不能单独取其中某个字
- 指令和数据在主存内必须是连续存放的

多体并行存储器

- 每个模块都有相同的容量和存取速度
- 各模块都有独立的读写控制电路、地址寄存器和数据寄存器。它们既能并行工作，又能交叉工作

高位交叉编址

- 理论上多个存储体可以被并行访问，但是由于通常会连续访问，因此实际效果相当于单纯的扩容

低位交叉编址

- 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，每个存取周期内所有模块被并行访问

Cache-高速缓冲存储器

工作原理

- 将某些主存块复制到Cache中，缓和CPU与主存之间的速度矛盾

局部性原理

- 时间局部性
  - 现在访问的地址，不久之后也很可能被再次访问
- 空间局部性
  - 现在访问的地址，其附近的地址也很可能即将被访问
- 性能分析
  - Cache命中率和缺失率
  - 两种访问方式
    - 先访问Cache，发现未命中再访问主存.
    - 同时访问Cache和主存，若Cache命中则停止访问主存
- 其他概念
  - 主存与Cache之间以“块”为单位进行数据交换
  - 主存的“块”又叫“页/页框/页面”，Cache的“块”又叫“行”
  - 主存地址可拆分为(主存块号，块内地址)的形式

Cache-主存映射方式

Cache中存储的信息

- 有效位(0/1)+标记+整块数据
- 其中“标记”用于指明对应的内存块，不同映射方式，“标记”的位数不同

全相联映射

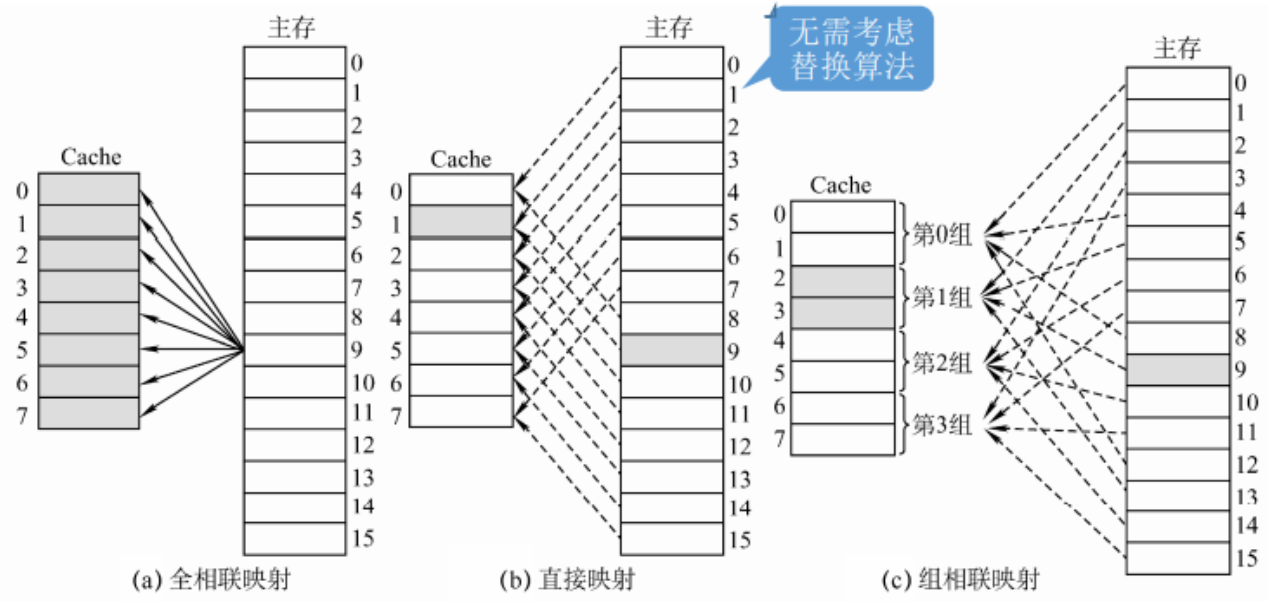
- 主存块可以放到Cache的任意位置
- 主存地址结构：标记(整个主存块号)+块内地址
- 优点：Cache存储空间利用充分，命中率高
- 缺点：查找“标记”最慢，有可能需要对比所有行的标记

直接映射

- 主存块只能放到特定的某个Cache行，行号=主存块号%总行数
- 主存地址结构：标记(主存块号前几位)+行号(主存块号末几位)块内地址
- 优点：对于任意一个地址，只需对比一个“标记”，速度最快
- 缺点：Cache存储空间利用不充分，命中率低

组相联映射

- 主存块可以放到特定分组中的任意位置，所属组号=主存块号%总组数
- 主存地址结构：标记(主存块号前几位)+组号(主存块号末几位)块内地址
- 优点：另外两种方式的折中，综合效果较好
- 术语：n路组相联映射——每n个Cache行为一组



Cache完全满了才需要替换
需要在全局选择替换哪一块

如果对应位置非空，则
毫无选择地直接替换

分组内满了才需要替换
需要在分组内选择替换哪一块

Cache-替换算法

随机算法(RAND)

- 随机选一个主存块替换
- 实现简单，但完全没考虑局部性原理，命中率低，实际效果很不稳定

先进先出算法(FIFO)

- 优先替换最先被调入Cache的主存块
- 不遵循局部性原理，效果差
- 抖动现象：频繁的换入换出现象(刚被替换的块很快又被调入)

最近最少使用算法(LRU)

- 将最久没有被访问过的主存块替换。每个Cache
- Cache块的总数$=2^n$，则计数器只需n位。且Cache装满后所有计数器的值一定不重复
  行设置一个“计数器”，用于记录多久没被访问
- 基于“局部性原理”。LRU算法的实际运行效果优秀，Cache命中率高
- 若被频繁访问的主存块数量 > Cache行的数量，则有可能发生抖动现象
- 方法
  - 命中时，所命中的行的计数器清零，比其低的计数器加1，其余不变
  - 未命中且还有空闲行时，新装入的行的计数器置0，其余非空闲行全加1
  - 未命中且无空闲行时，计数值最大的行的信息块被淘汰，新装行的块的计数器置0，其余全加1

最不经常使用(LFU)

- 将被访问次数最少的主存块替换。每个Cache行设置一个“计数器”，用于记录被访问过多少次
- 曾经被经常访问的主存块在未来不一定会用到，LFU实际运行效果不好

Cache-写策略

写命中

- 全写法(写直通法)
  - 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(write buffer)
- 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

写不命中

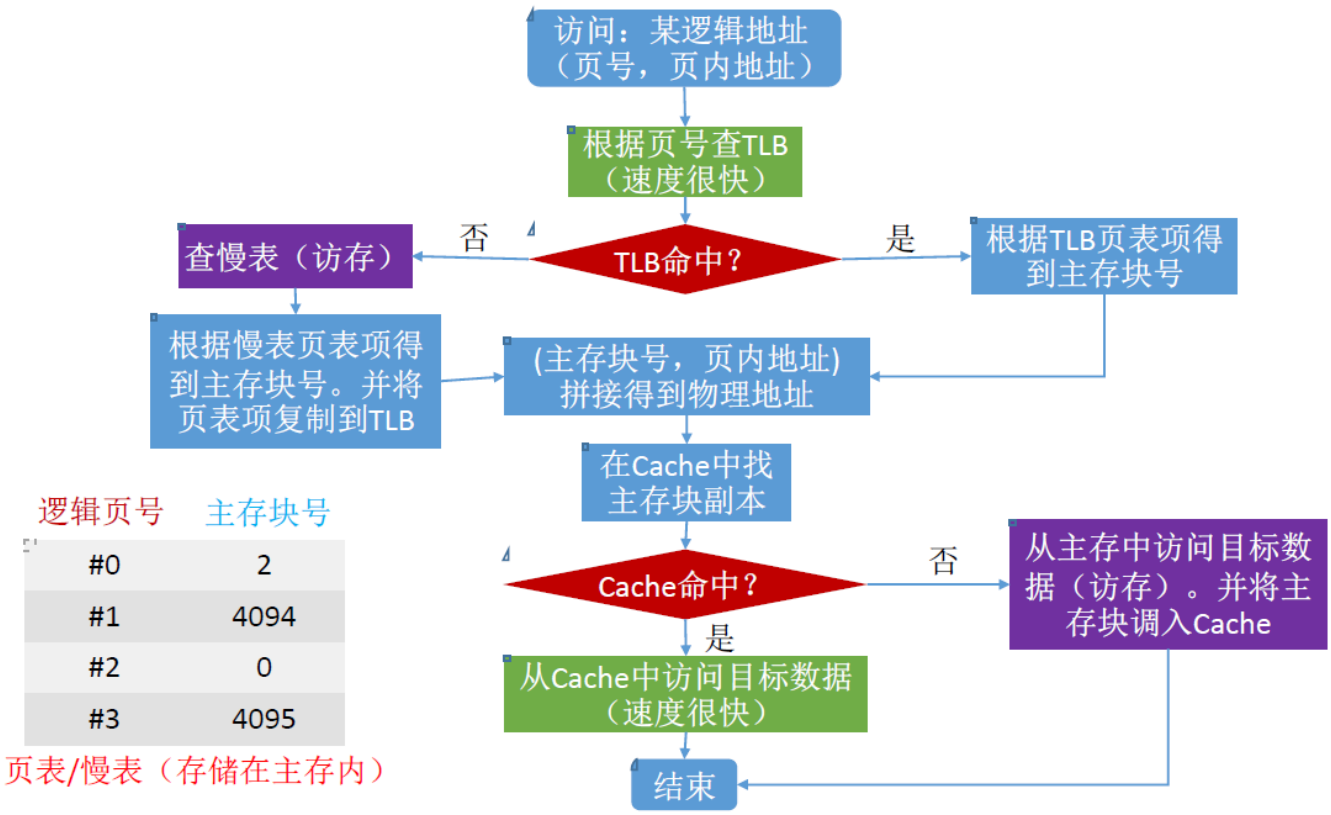
- 写分配法
  - 当CPU对Cache写不命中时，把主存中的块调入Cache，在Cache中修改。通常搭配写回法使用
- 非写分配法
  - 当CPU对Cache写不命中时只写入主存，不调入Cache。通常搭配全写法使用

多级Cache

现代计算机通常采用多级Cache结构，各级Cache间常采用“全写法+非写分配法”，Cache和主存间常采用“写回法+写分配法”

页式存储

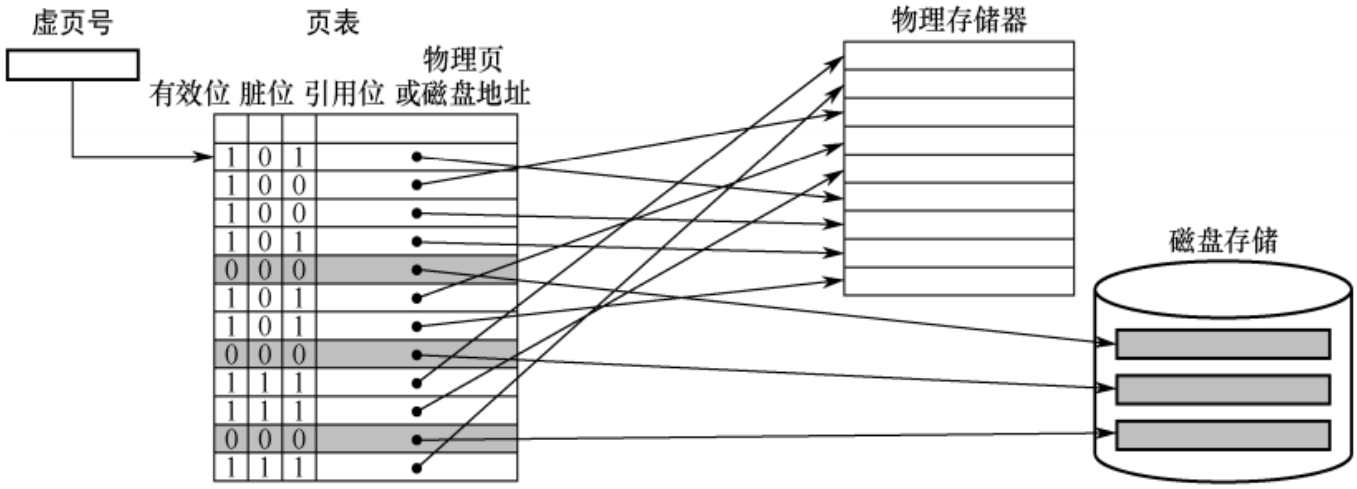
- 页式存储系统：一个程序在逻辑上被分为若干个大小相等的“页面”，“页面”大小与“块”的大小相同。每个页面可以离散地放入不同的主存块中。
- 逻辑地址= 逻辑页号+页内地址(虚地址=虚页号+页内地址)
- 物理地址= 主存块号+页内地址(实地址=实页号+页内地址)
- 快表中存储的是页表项的副本，Cache中存储的是主存块的副本



虚拟存储器

页式虚拟存储器

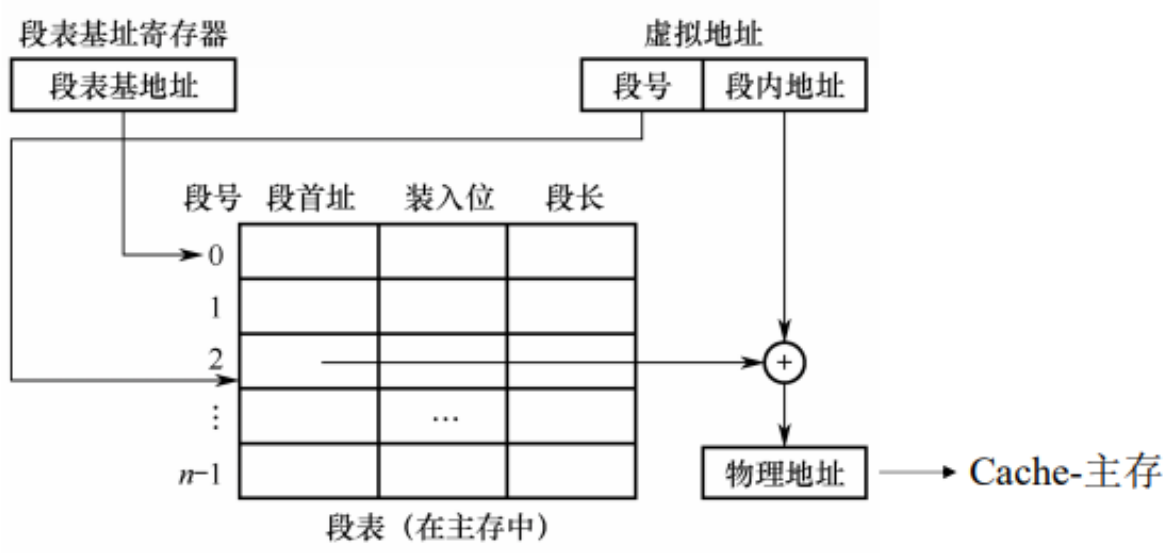
- 拆分成大小相等的页面



有效位：这个页面是否已调入主存
脏位：这个页面是否被修改过
引用位：用于“页面置换算法”，比如，可以用来统计这个页面被访问过多少次
物理页：即主存块号
磁盘地址：即这个页面的数据在磁盘中的存放位置

段式虚拟存储器

- 按照功能模块拆分。如：#0 段是自己的代码，#1 段是库函数代码，#2段是变量



段页式虚拟存储器

- 按照功能模块分段，再将各个段分页
- 把程序按逻辑结构分段，每段再划分为固定大小的页，主存空间也划分为大小相等的页。
- 程序对主存的调入、调出仍以页为基本传送单位。每个程序对应一个段表，每段对应一个页表。
- 虚拟地址：段号+段内页号+页内地址

4 指令系统

指令格式

四地址指令

OP	A_1	A_2	A_3（结果）	A_4（下址）

指令含义：$(A_1)OP(A_2)\rightarrow A_3$，$A_4$=下一条将要执行指令的地址

三地址指令

OP	A_1	A_2	A_3（结果）

指令含义：$(A_1)OP(A_2)\rightarrow A_3$

二地址指令

OP	A_1（目的操作数）	A_2（源操作数）

指令含义：$(A_1)OP(A_2)\rightarrow A_1$

一地址指令

OP	A_1

指令含义：1. $OP(A_1)\rightarrow A_1$，如加1、减1、取反、求补等
2. $(ACC)OP(A_1)\rightarrow ACC$，隐含约定的目的地址为ACC

零地址指令

OP

指令含义：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，计算结果压回栈顶

- 一条指令由操作码、地址码组成，其中地址码可能有0~4个
- 按地址码数目分类
  - 零/一/二/三/四地址指令
- 按指令长度分类
  - 概念
    - 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)
  - 指令长度是机器字长的多少倍：半字长指令、单字长指令、双字长指令
  - 定长指令字结构
    - 所有指令长度相同
  - 变长指令字结构
    - 各指令长度不同
- 按操作码长度分类
  - 定长操作码
    - 控制器的译码电路设计简单，但灵活性较低
  - 可变长操作码
    - 控制器的译码电路设计复杂，但灵活性较高
- 按操作类型分类
  - 数据传送类
    - CPU、主存之间的数据传送
  - 运算类

        - 算数逻辑操作、移位操作
    - 程序控制类
        - 改变程序执行流
    - 输入输出类
        - CPU、IO设备之间的数据传送

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)

指令操作码

- 操作码是识别指令、了解指令功能与区分操作数地址内容的组成和使用方法等的关键信息。

定长操作码

- 在指令字的最高位部分分配固定的若干位(定长)表示操作码。
- 一般n位操作码字段的指令系统最大能够表示2^n条指令
- 优点：定长操作码对于简化计算机硬件设计，提高指令译码和识别速度很有利
- 缺点：指令数量增加时会占用更多固定位，留给表示操作数地址的位数受限

扩展操作码(不定长操作码)

- 全部指令的操作码字段的位数不固定，且分散地放在指令字的不同位置上
- 最常见的变长操作码方法是扩展操作码，使操作码的长度随地址码的减少而增加
- 不同地址数的指令可以具有不同长度的操作码，在满足需要的前提下，有效地缩短指令字长
- 优点：在指令字长有限的前提下仍保持比较丰富的指令种类
- 缺点：增加了指令译码和分析的难度，使控制器的设计复杂化
- 需满足
    - 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的代码相同
    - 各指令的操作码一定不能重复

设指令字长固定为16位，试设计一套指令系统满足：

a) 有15条三地址指令
共2^4=16种状态
留出16-15=1种
b) 有12条二地址指令
共1 ×2^4=16种
留出16-12=4种
c) 有62条一地址指令
共4 ×2^4=64种
留出64-62=2种
d) 有32条零地址指令
共2 ×2^4=32种

	0000 - 1110	A_1	A_2	A_3
1111 XXXX XXXX XXXX	1111	0000 - 1011	A_1	A_2
1111 11XX XXXX XXXX	1111	1100 – 1110 1111	0000 – 1111 0000 – 1101	A_1
1111 1111 111X XXXX	1111	1111	1110 – 1111	0000 - 1111

设地址长度为n，上一层留出m种状态，下一层可扩展出m×2^n种状态

指令寻址

什么是指令寻址

- 确定下一条要执行的指令的存放地址
- 由程序计数器PC指明
- 每一条指令的执行都分为 “取指令”、“执行指令” 两个阶段

顺序寻址

- 每次取址结束后，PC一定+1
- “1”理解为1个指令字长
- 定长指令字结构
  - 主存按字编址
  - 主存按字节编址
- 变长指令字结构
  - 主存按字节编址

跳跃寻址

- 执行转移类指令导致PC值改变

数据寻址

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
偏移寻址 转移指令 相对寻址	EA=(PC)+A	1
多道程序 基址寻址	EA=(BR)+A	1
循环程序 变址寻址 数组问题	EA=(IX)+A	1
堆栈寻址	入栈/出栈时EA的确定方式不同	硬堆栈不访存，软堆栈访存1次

概念

- 确定本条指令的地址码指明的真实地址
- 寻址特征+形式地址：求出操作数的真实地址，称为有效地址(EA)
- 比较的硬件实现

高级语言视角：

```
if (a>b){
        …
} else {
        …
}
```

汇编语言中，条件跳转指令有很多种，如 je 2 表示当比较结果为 a=b 时跳转到2
jg 2 表示当比较结果为a>b时跳转到2

硬件视角：

- 通过"cmp指令"比较 a 和 b（如 cmp a, b），实质上是用 a-b
- 相减的结果信息会记录在程序状态字寄存器中（PSW）
- 根据PSW的某几个标志位进行条件判断，来决定是否转移

有的机器把PSW称为“标志寄存器”

PSW中有几个比特位记录上次运算的结果

- 进位/借位标志 CF：最高位有进位/借位时CF=1
- 零标志 ZF：运算结果为0则 ZF=1，否则ZF=0
- 符号标志 SF：运算结果为负，SF=1，否则为0
- 溢出标志 OF：运算结果有溢出OF=1否则为0

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*

直接寻址

- 指令字中的形式地址A就是操作数的真实地址EA，即EA=A
- 优点：简单，指令执行阶段仅访问一次主存，不需专门计算操作数的地址
- A的位数决定了该指令操作数的寻址范围。操作数的地址不易修改。

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)
- 优点：可扩大寻址范围(有效地址EA的位数大于形式地址A的位数)。便于编制程序(用间接寻址可以方便地完成子程序返回)
- 缺点：指令在执行阶段要多次访存(一次间址需两次访存，多次寻址需根据存储字的最高位确定几次访存)

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
- 优点：指令在执行阶段不访问主存，只访问寄存器，指令字短且执行速度快，支持向量/矩阵运算
- 缺点：寄存器价格昂贵，计算机中寄存器个数有限

寄存器间接寻址

- 寄存器R_i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$EA = (R_i)$
- 与一般间接寻址相比速度更快，但指令的执行阶段需要访问主存(因为操作数在主存中)

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(例如隐含在ACC中)
- 优点：有利于缩短指令字长
- 缺点：需增加存储操作数或隐含地址的硬件

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
- 优点：指令执行阶段不访问主存，指令执行时间最短
- 缺点：A的位数限制了立即数的范围

偏移寻址

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
- 优点：便于程序“浮动”，方便实现多道程序并发运行

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
- 变址寄存器是面向用户的，在程序执行过程中，变址寄存器的内容可由用户改变(作为偏移量)，形式地址A不变(作为基地址)
- 优点：在数组处理过程中，可设定A为数组的首地址，不断改变变址寄存器IX的内容，便可很容易形成数组中任一数据的地址，特别适合编制循环程序

基址&变址复合寻址

- 基址寻址：EA = (BR)+A
- 变址寻址：EA = (IX)+A
- 先基址后变址寻址：EA = (IX)+(BR)+A

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是相对于PC所指地址的位移量，可正可负，补码表示
- 优点：操作数的地址不是固定的，它随着PC值的变化而变化，并且与指令地址之间总是相差一个固定值，因此便于程序浮动(一段代码在程序内部的浮动)。相对寻址广泛应用于转移指令。
- 取出当前指令后，PC会指向下一条指令，相对寻址是相对于下一条指令的偏移

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
- 堆栈是存储器(或专用寄存器组)中一块特定的按“后进先出(LIFO)”原则管理的存储区，该存储区中被读/写单元的地址是用一个特定的寄存器给出的，该寄存器称为堆栈指针(SP)。

CISC和RISC

类别 / 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

- CISC：Complex Instruction Set Computer
  - 设计思路：一条指令完成一个复杂的基本功能。
- RISC：Reduced Instruction Set Computer
  - 设计思路：一条指令完成一个基本“动作”，多条指令组合完成一个复杂的基本功能。

5 中央处理器

CPU的功能

CPU的功能

- 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。
- 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。
- 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。
- 数据加工：对数据进行算术和逻辑运算。
- 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进行处理。

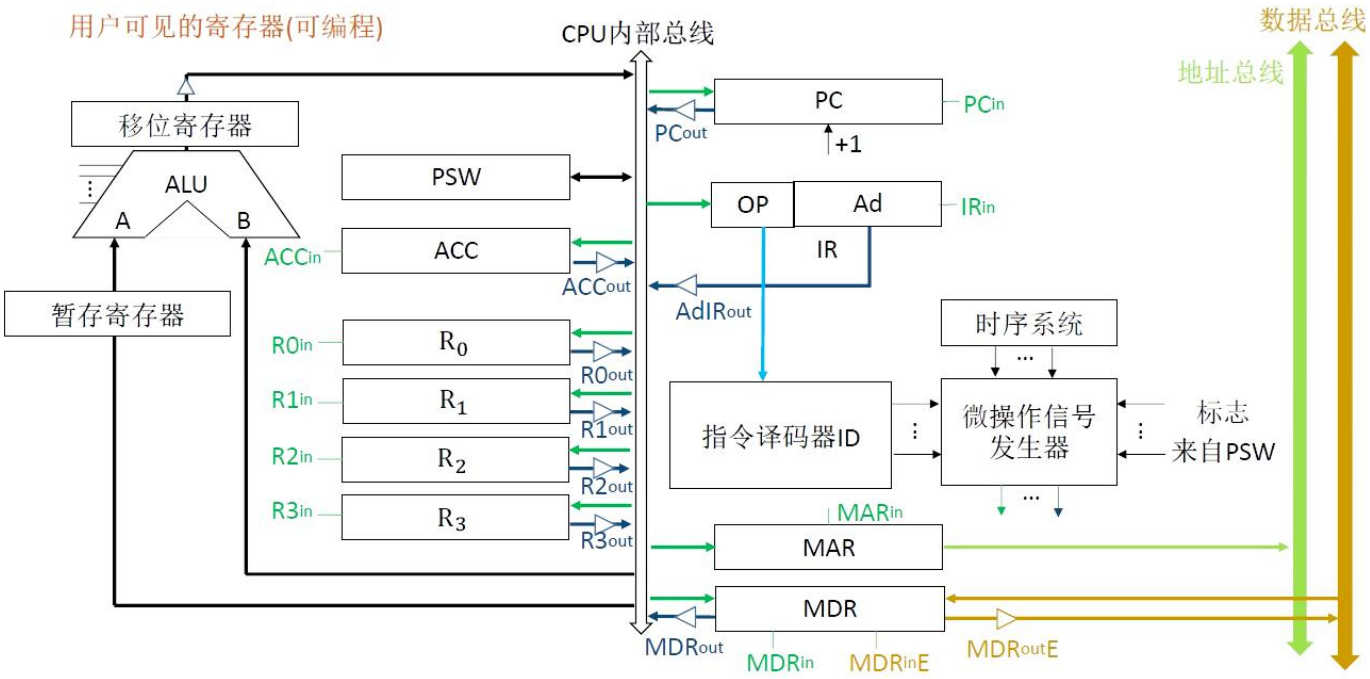
控制器的功能

- 协调并控制计算机各部件执行程序的指令序列
- 基本功能
  - 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)。

运算器的功能

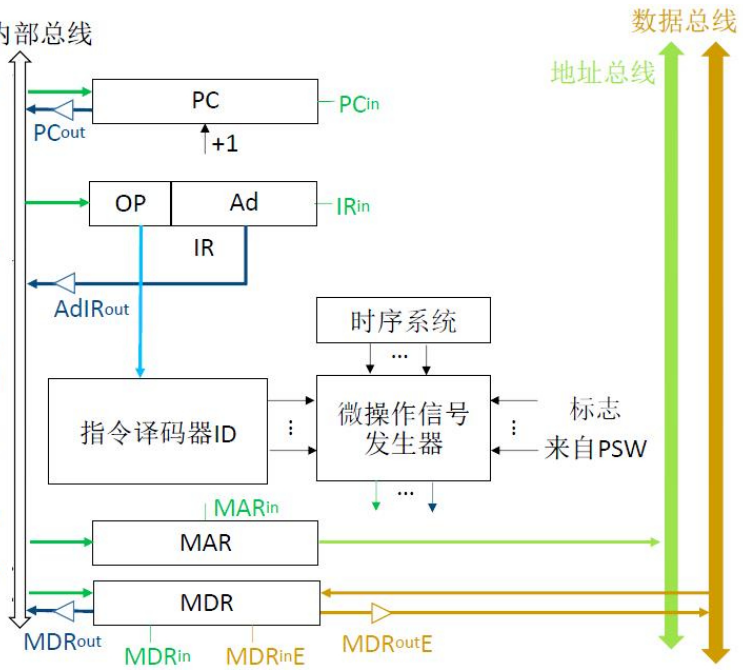
- 对数据进行加工

CPU的基本结构



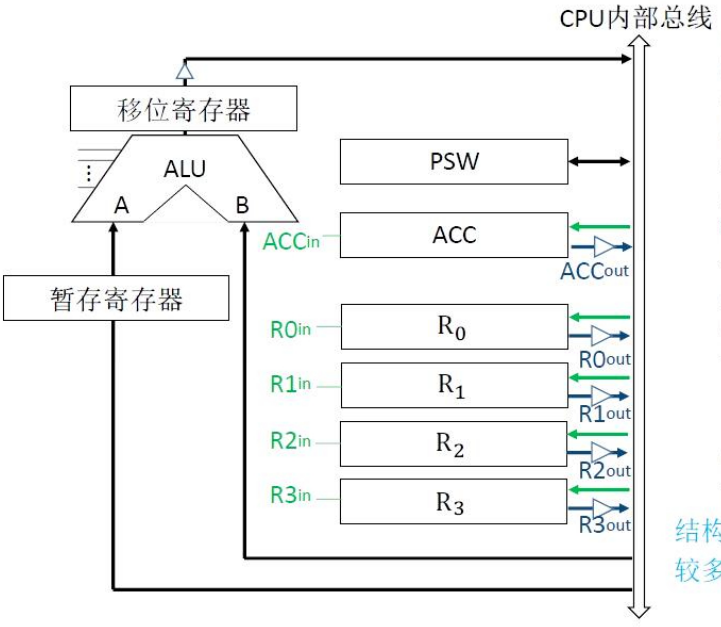
控制器的基本结构

1. 程序计数器：用于指出下一条指令在主存中的存放地址。CPU就是根据PC的内容去主存中取指令的。因程序中指令（通常）是顺序执行的，所以PC有自增功能。
2. 指令寄存器：用于保存当前正在执行的那条指令。
3. 指令译码器：仅对操作码字段进行译码，向控制器提供特定的操作信号。
4. 微操作信号发生器：根据IR的内容（指令）、PSW的内容（状态信息）及时序信号，产生控制整个计算机系统所需的各种控制信号，其结构有组合逻辑型和存储逻辑型两种。
5.时序系统：用于产生各种时序信号，它们都是由统一时钟（CLOCK）分频得到。
6. 存储器地址寄存器：用于存放所要访问的主存单元的地址。
7. 存储器数据寄存器：用于存放向主存写入的信息或从主存中读出的信息。



- 程序计数器PC
- 指令寄存器IR
- 指令译码器、时序系统、微操作信号发生器
- 存储器地址寄存器MAR
- 存储器数据寄存器MDR

运算器的基本结构



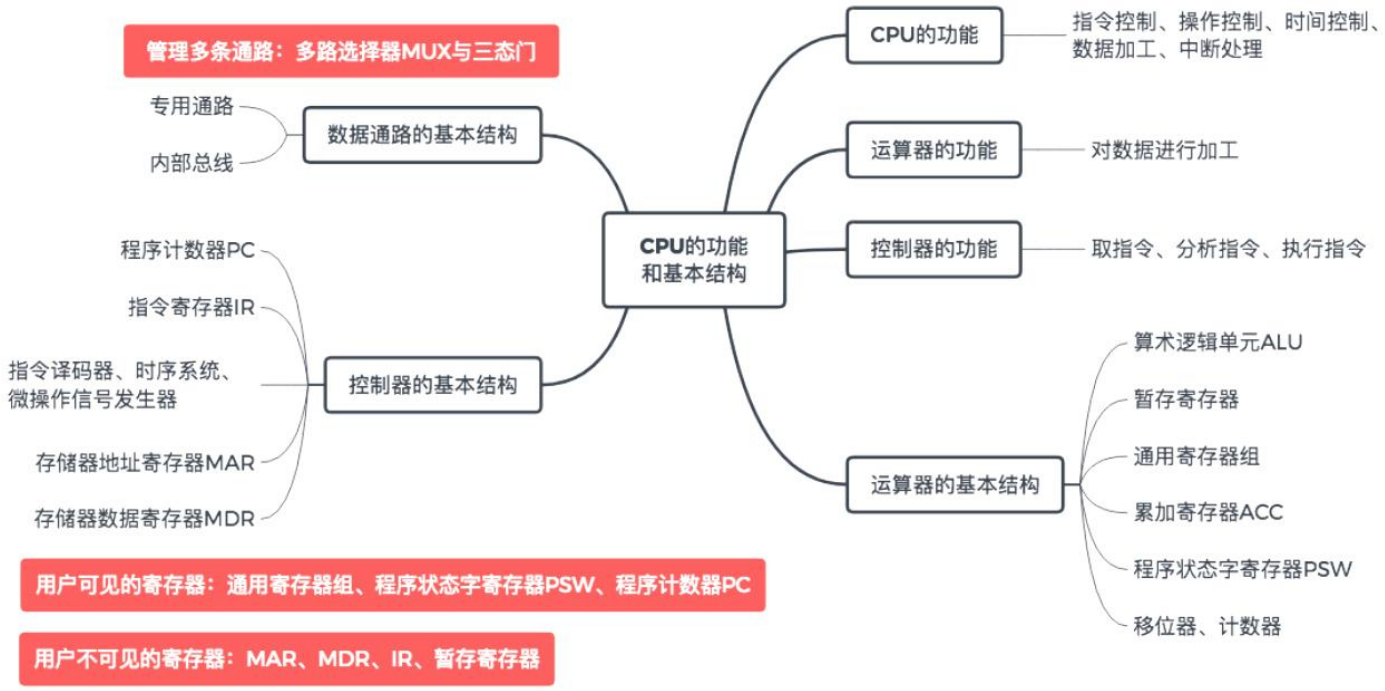
1. 算术逻辑单元：主要功能是进行算术/逻辑运算。
2. 通用寄存器组：如AX、BX、CX、DX、SP等，用于存放操作数（包括源操作数、目的操作数及中间结果）和各种地址信息等。SP是堆栈指针，用于指示栈顶的地址。
3. 暂存寄存器：用于暂存从主存读来的数据，这个数据不能存放在通用寄存器中，否则会破坏其原有内容。
4. 累加寄存器：它是一个通用寄存器，用于暂时存放ALU运算的结果信息，用于实现加法运算。
5. 程序状态字寄存器：保留由算术逻辑运算指令或测试指令的结果而建立的各种状态信息，如溢出标志（OP）、符号标志（SF）、零标志（ZF）、进位标志（CF）等。PSW中的这些位参与并决定微操作的形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

结构简单，容易实现，但数据传输存在较多冲突的现象，性能较低。

- 算术逻辑单元ALU
- 暂存寄存器
- 通用寄存器组
- 累加寄存器ACC
- 程序状态字寄存器PSW

- 移位器、计数器

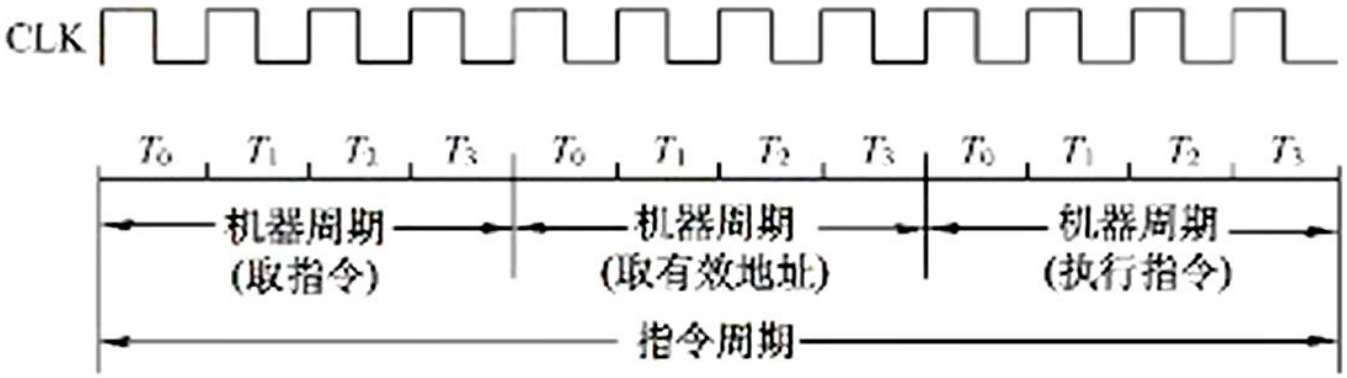
总结



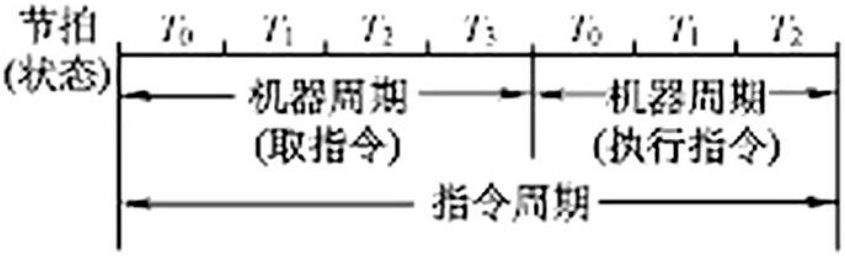
指令执行过程

指令周期

- 指令周期：CPU从主存中每取出并执行一条指令所需的全部时间。
- 指令周期常常用若干机器周期来表示，机器周期又叫CPU周期。
- 一个机器周期又包含若干时钟周期(也称为节拍、T周期或CPU时钟周期，它是CPU操作的最基本单位)
- 取指周期、间址周期、执行周期、中断周期
- 标志触发器FE、IND、EX、INT

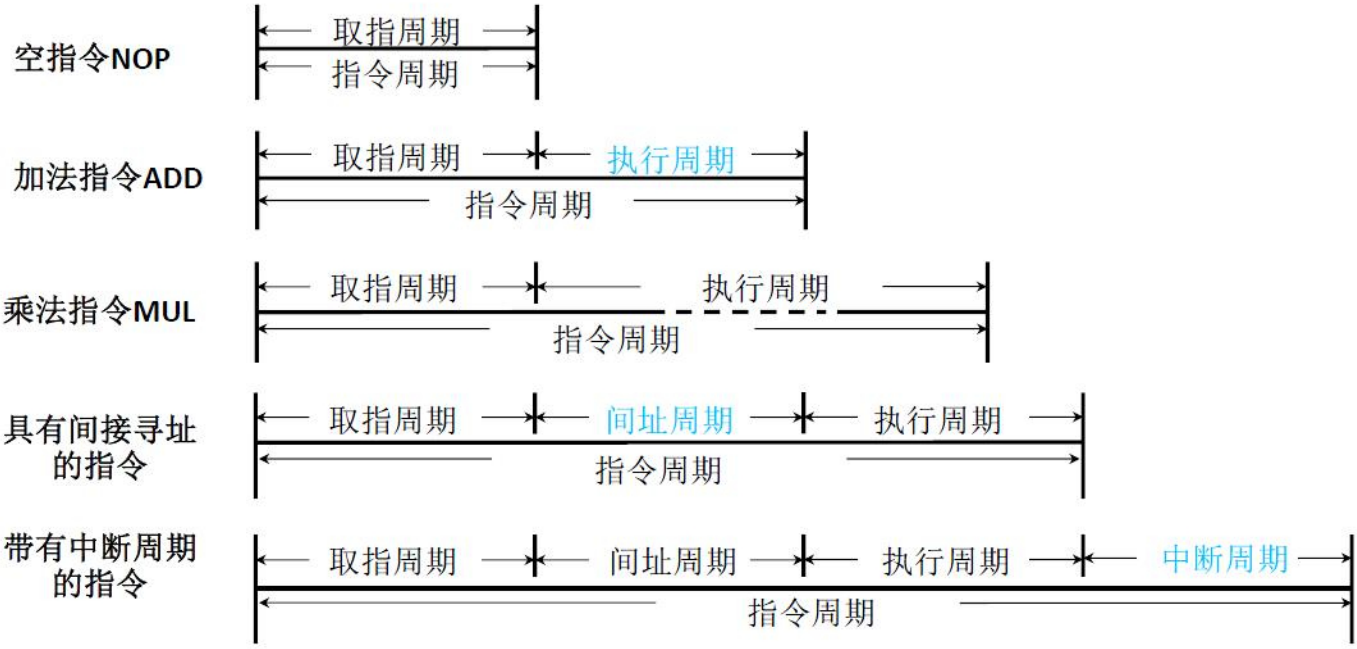


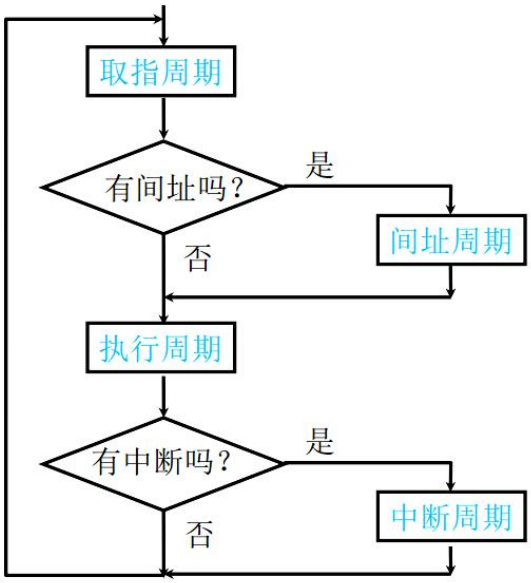
(a) 定长的机器周期

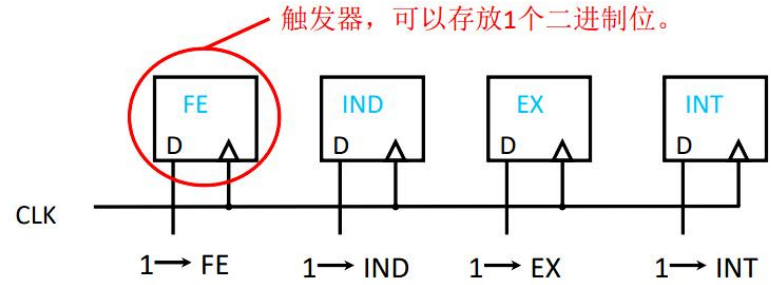


(b) 不定长的机器周期

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。





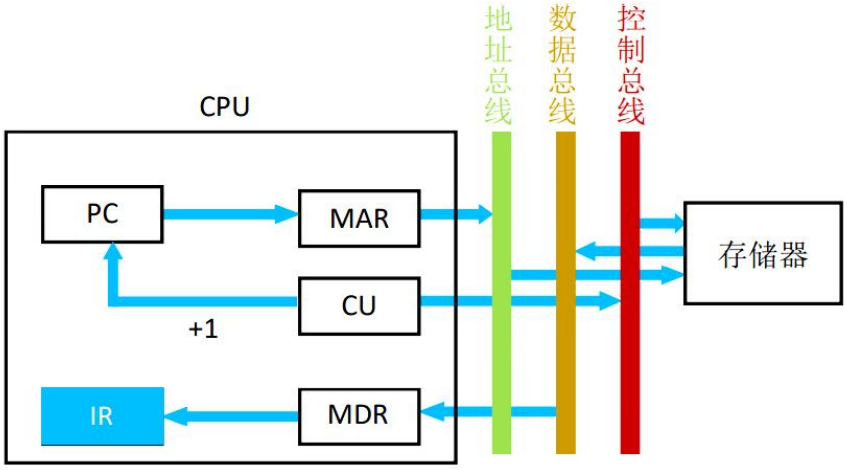


取指周期：	1	0	0	0
间址周期：	0	1	0	0
执行周期：	0	0	1	0
中断周期：	0	0	0	1

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**。

数据流

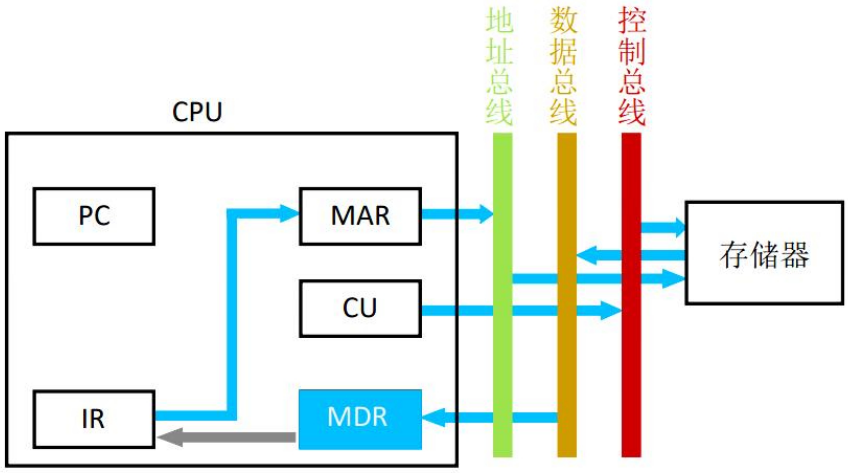
取指周期



1. 当前指令地址送至存储器地址寄存器，记做：(PC) → MAR
2. CU发出控制信号，经控制总线传到主存，这里是**读信号**，记做：1 → R
3. 将MAR所指主存中的内容经数据总线送入MDR，记做：M(MAR) → MDR
4. 将MDR中的内容(此时是指令)送入IR，记做：(MDR) → IR
5. CU发出控制信号，形成下一条指令地址，记做：(PC)+1 → PC

- 根据PC中的内容取出指令代码并存放在IR中

间址周期



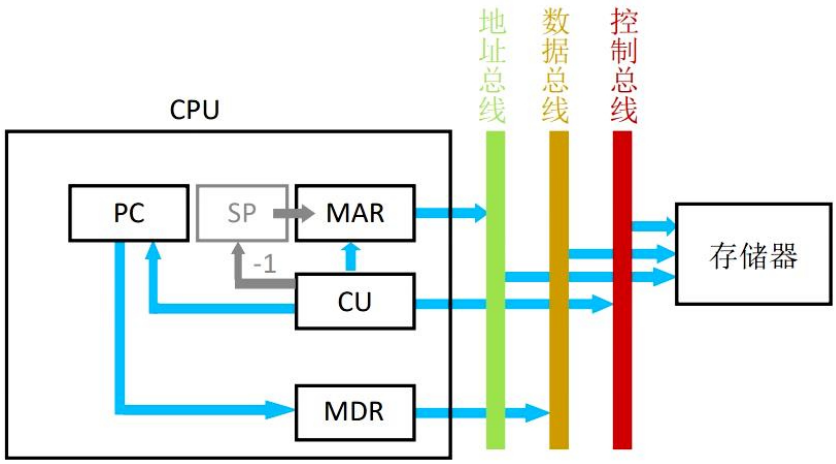
1. 将指令的地址码送入MAR，
记做：Ad(IR) → MAR
或Ad(MDR) → MAR
2. CU发出控制信号，启动主存做**读操作**，记做：1 → R
3. 将MAR所指主存中的内容经数据总线送入MDR，记做：M(MAR) → MDR
4. 将有效地址送至指令的地址码字段，记做：(MDR)→ Ad(IR)

- 根据IR中指令地址码取操作数有效地址

执行周期

- 根据指令字的操作码和操作数进行相应的操作

中断周期



中断：暂停当前任务去完成其他任务。
为了能够恢复当前任务，需要**保存断点**。
一般使用堆栈来保存断点，这里用SP表示栈顶地址，假设**SP指向栈顶元素**，进栈操作是**先修改指针，后存入数据**。

1. CU控制将SP减1，修改后的地址送入MAR
记做：(SP)-1 → SP，(SP) → MAR
本质上是将断点存入某个存储单元，假设其地址为a，故可记做：a → MAR
2. CU发出控制信号，启动主存做**写操作**，
记做：1 → W
3. 将断点(PC内容) 送入MDR，
记做：(PC) → MDR
4. CU控制将中断服务程序的入口地址
(由向量地址形成部件产生)送入PC，
记做：向量地址→ PC

- 保存断电，送中断向量，处理中断请求

执行方案

单指令周期

- 所有指令选用相同的执行时间，指令间串行

多指令周期

- 不同类型指令选用不同的执行步骤，指令间串行

流水线方案

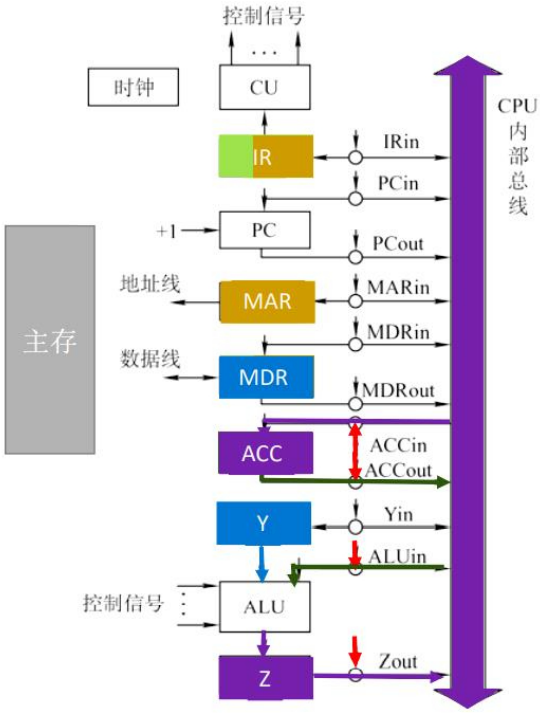
- 隔一段时间启动一条指令，多条指令处于不同阶段，同时运行

数据通路

- 数据通路：数据在功能部件之间传送的路径。

CPU内部总线

单总线



1．寄存器之间数据传送

比如把PC内容送至MAR，实现传送操作的流程及控制信号为：

(PC)→Bus　　PCout有效，PC内容送总线

Bus→MAR　　MARin有效，总线内容送MAR

2．主存与CPU之间的数据传送

比如CPU从主存读取指令，实现传送操作的流程及控制信号为：

(PC)→Bus→MAR　　PCout和MARin有效，现行指令地址→MAR

1→R　　CU发读命令(通过控制总线发出，图中未画出)

MEM(MAR)→MDR　　MDRin有效

MDR→Bus→IR　　MDRout和IRin有效，现行指令→IR

3．执行算术或逻辑运算

比如一条加法指令，微操作序列及控制信号为：

Ad(IR)→Bus→MAR　　MDRout和MARin有效

1→R　　CU发读命令

MEM(MAR)→数据线→MDR　　MDRin有效

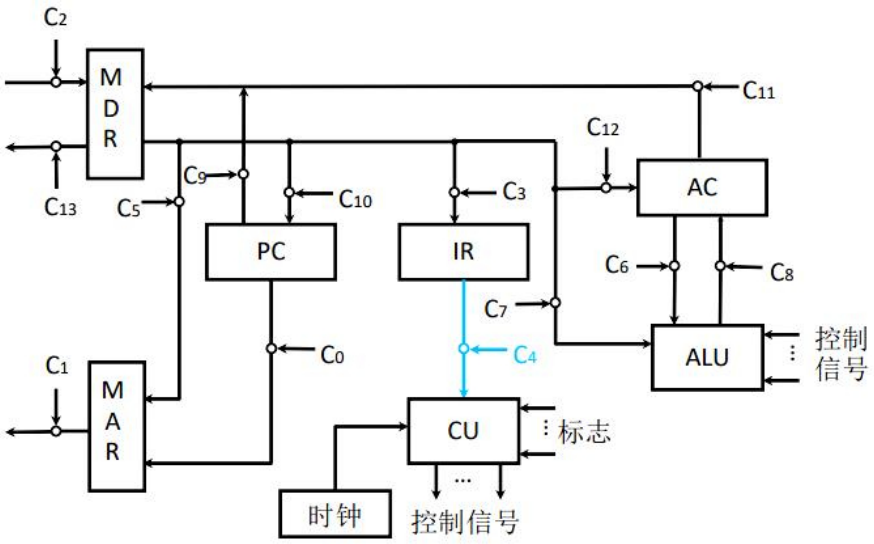
MDR→Bus→Y　　MDRout和Yin有效，操作数→Y

(ACC)+(Y)→Z　　ACCout和ALUin有效，CU向ALU发送加命令

Z→ACC　　Zout和ACCin有效，结果→ACC

多总线

专用数据通路



(PC)→MAR　　C0有效

(MAR)→主存　　C1有效

1→R　　控制单元向主存发送读命令

M(MAR)→MDR　　C2有效

(MDR)→IR　　C3有效

(PC)+1→PC

Op(IR)→CU　　C4有效

控制器设计

硬布线控制器

- 硬布线控制器：微操作控制信号由组合逻辑电路根据当前的指令码、状态和时序，即时产生

设计步骤

- 1.分析每个阶段的微操作序列
- 2.选择CPU的控制方式
- 3.安排微操作时序
- 4.电路设计

- 列出操作时间表
- 写出微操作命令的最简表达式
- 画出逻辑图

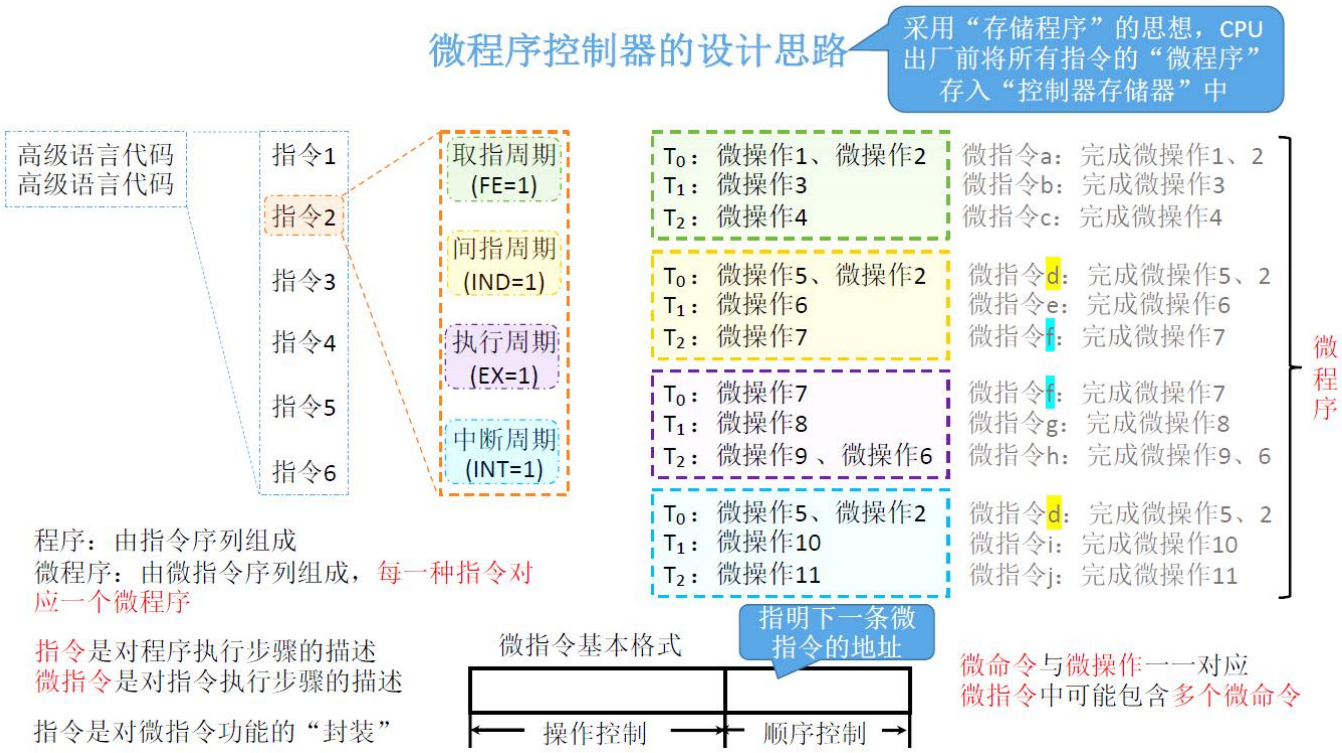
特点

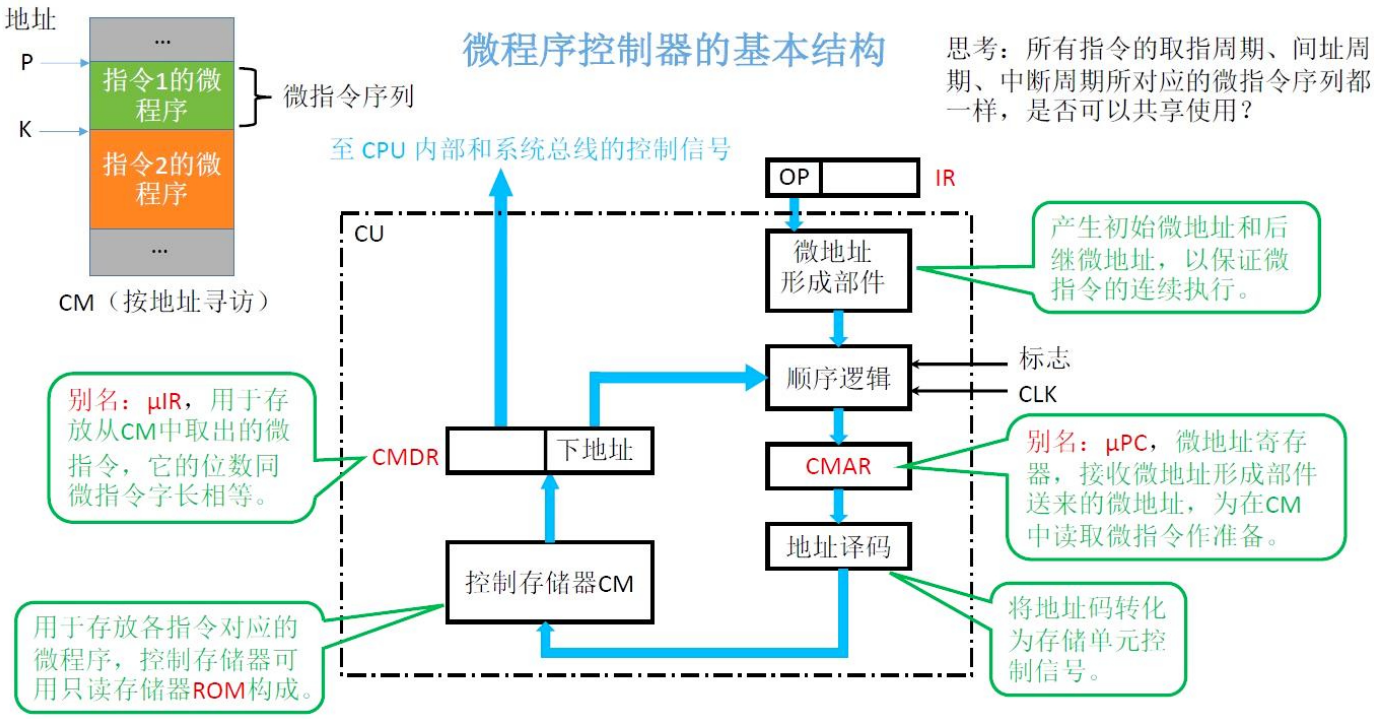
- 指令越多，设计和实现就越复杂，因此一般用于 RISC(精简指令集系统)
- 如果扩充一条新的指令，则控制器的设计就需要大改，因此扩充指令较困难
- 由于使用纯硬件实现控制，因此执行速度很快。微操作控制信号由组合逻辑电路即时产生

微程序控制器

基本原理

概念





- 程序vs微程序; 指令vs微指令; 主存储器vs控制器存储器(CM); MAR vs CMAR; MDR vs CMDR; PC VS μPC; IR VS μIR
- 微命令、微操作、微指令、微程序之间的关系
- 指令周期：从主存取出并执行一条机器指令所需的时间
- 微周期(微指令周期) : 从控制器存储器取出一条微指令并执行相应微操作所需的时间

- 程序由机器指令组成，微程序由微指令组成
- 微指令序列是对指令具体功能的描述，每一条微指令对应一个或多个微操作，微指令描述指令执行的各个阶段需要完成那些微操作
- 主存储器存放机器指令，控制器存储器存放微指令序列
- 微命令与微操作一一对应，微指令可能包含多个微命令，微程序由一系列的微指令组成，微程序与机器指令一对一的关系

CU的构成

- 微地址形成部件
  - 微地址即微指令在CM中的存放地址
  - 通过指令操作码形成对应微程序的第一条微指令的存放地址
- 顺序逻辑
  - 根据某些机器标志和时序信息确定下一条微指令的存放地址
- CMAR(μPC)
  - 指明接下来要执行的微指令的存放地址
- 地址译码器
  - 将CMAR内的地址信息译码为电信号，控制CM读出微指令
- 控制存储器CM
  - 存放所有机器指令对应的微程序(微指令序列)
  - 用ROM实现，按地址寻访。通常在CPU出厂时就把所有微程序写入

- CMDR(μIR)
    - 微指令寄存器，用于存放当前要执行的微指令。CM(μPC) -> μIR

工作原理

- 指令周期 = 取值周器 -> 间址周期 -> 执行周期 -> 中断周期。其中间址、中断周期可有可无
- 处理取指周期、间址周期、中断周期的微指令序列通常是公用的。执行周期的微指令序列各不相同
- 取指周期的微指令序列固定从#O开始存放。执行周期的微指令序列的存放根据指令操作码确定

微指令的设计

微指令格式

- 水平型微指令
    - 每条微指令能定义多个可并行的微命令
- 垂直型微指令
    - 每条微指令只能定义一个微命令，由微操作码指明
- 混合型微指令
    - 在垂直型微指令的基础上加上一些简单的并行操作

水平型微指令的编码方式

- 直接编码(直接控制)
    - 控制码的每个bit对应一个微命令，微指令执行速度最快
- 字段直接编码
    - 将互斥性的微命令分在同一个段内，相容的分在不同的段
    - 每个段留出一个状态表示“不操作”
    - 微指令操作码需要经过译码电路处理，因此执行速度更慢
- 字段间接编码
    - 一个字段的微命令需要用另一个字段的微命令解释
    - 可能需要多级译码电路处理，执行速度最慢

下一条微指令地址的形成方式

- 断定法(下地址法)：根据当前执行的微指令下地址找到下一条微指令
- 计数器法: μPC+1顺序找到下一条微指令
- 根据指令操作码确定执行周期微程序首地址
- 由专门的硬件指明取指/中断周期的微程序首地址

微程序控制单元的设计

- 1.分析每个阶段的微操作序列
- 2.写出对应机器指令的微操作命令及节拍安排
    - 写出每个周期所需要的微操作(参照硬布线)
    - 补充微程序控制器特有的微操作
- 确定微指令格式
- 编写微指令码点

微程序设计分类

- 静态微程序设计和动态微程序设计
  - 静态：微程序无需改变，采用ROM
  - 动态：通过改变微指令和微程序改变机器指令有利于仿真，采用EPROM
- 毫微程序设计
  - 微程序设计用微程序解释机器指令
  - 毫微程序设计用毫微程序解释微程序
  - 毫微指令与微指令的关系好比微指令与机器指令的关系

硬布线与微程序的比较

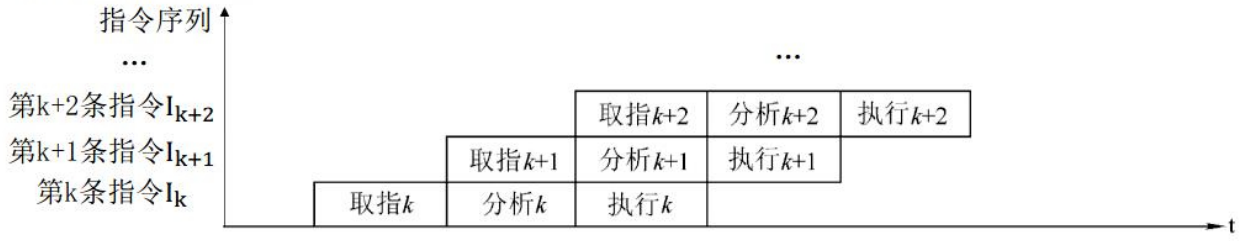
类别 / 对比项目	微程序控制器	硬布线控制器
工作原理	微操作控制信号以微程序的形式存放在控制存储器中，执行指令时读出即可	微操作控制信号由组合逻辑电路根据当前的指令码、状态和时序，即时产生
执行速度	慢	快
规整性	较规整	烦琐、不规整
应用场合	CISC CPU	RISC CPU
易扩充性	易扩充修改	困难

指令流水线

基本概念

- 指令执行过程划分为不同阶段，占用不同的资源，就能使多条指令同时执行
- 表示方法
  - 指令流程图
    - 主要用于分析影响流水线的因素
  - 时空图
    - 主要用于分析流水线的性能

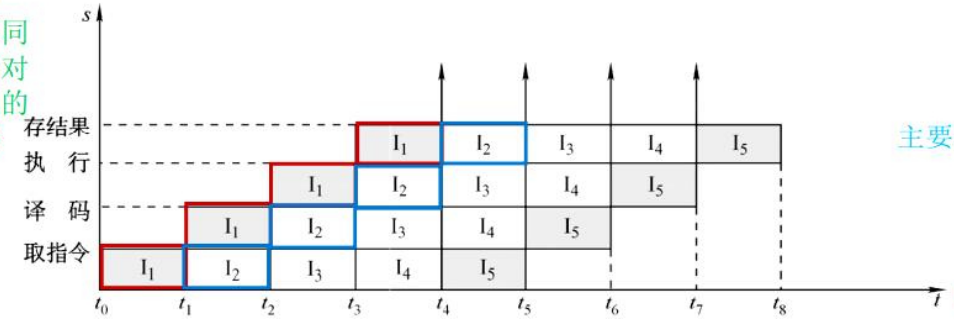
1. 指令执行过程图



主要用于分析指令执行过程以及影响流水线的因素(见下一个视频)

2. 时空图

空间：不同的阶段所对应的不同的硬件资源



主要用于分析流水线的性能

性能指标

吞吐率TP

- 在单位时间内流水线所完成的任务数量，或是输出结果的数量

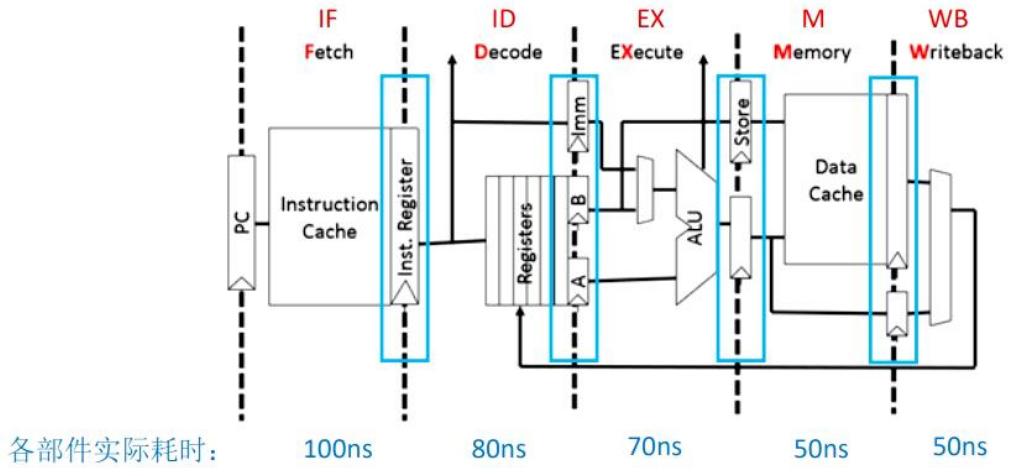
加速比S

- 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

效率E

- 流水线的设备利用率称为流水线的效率

影响因素



流水线**每一个功能段部件后面**都要有一个**缓冲寄存器**，或称为**锁存器**，其作用是**保存本流水段的执行结果**，提供给下一流水段使用。

各部件实际耗时：100ns 80ns 70ns 50ns 50ns

为方便流水线的设计，将每个阶段的耗时取成一样，以最长耗时为准。
即此处应将机器周期设置为100ns。

结构相关(资源冲突)

- 原因：多条指令争用同一资源
- 暂停相关指令
- 资源重复配置

数据相关(数据冲突)

- 原因：后续指令需要用到之前指令的执行结果
- 暂停相关指令
    - 硬件stall
    - 软件NOP
- 数据旁路技术
- 编译优化、调整指令顺序

控制相关(控制冲突)

- 原因：遇到转移指令和其它改变PC值的指令时发生
- 分支预测
- 预取两个方向的指令
- 加快和提前形成条件码
- 提高转移方向的猜准率

分类

- 按使用级别：部件功能级、处理机级、处理机间
- 按完成功能：单功能、多功能
- 按连接方式：动态、静态
- 按有无反馈信号：线性、非线性

流水线的多发技术

- 超标量流水线技术(空分复用)
- 超流水线技术(时分复用)
- 超长指令字技术

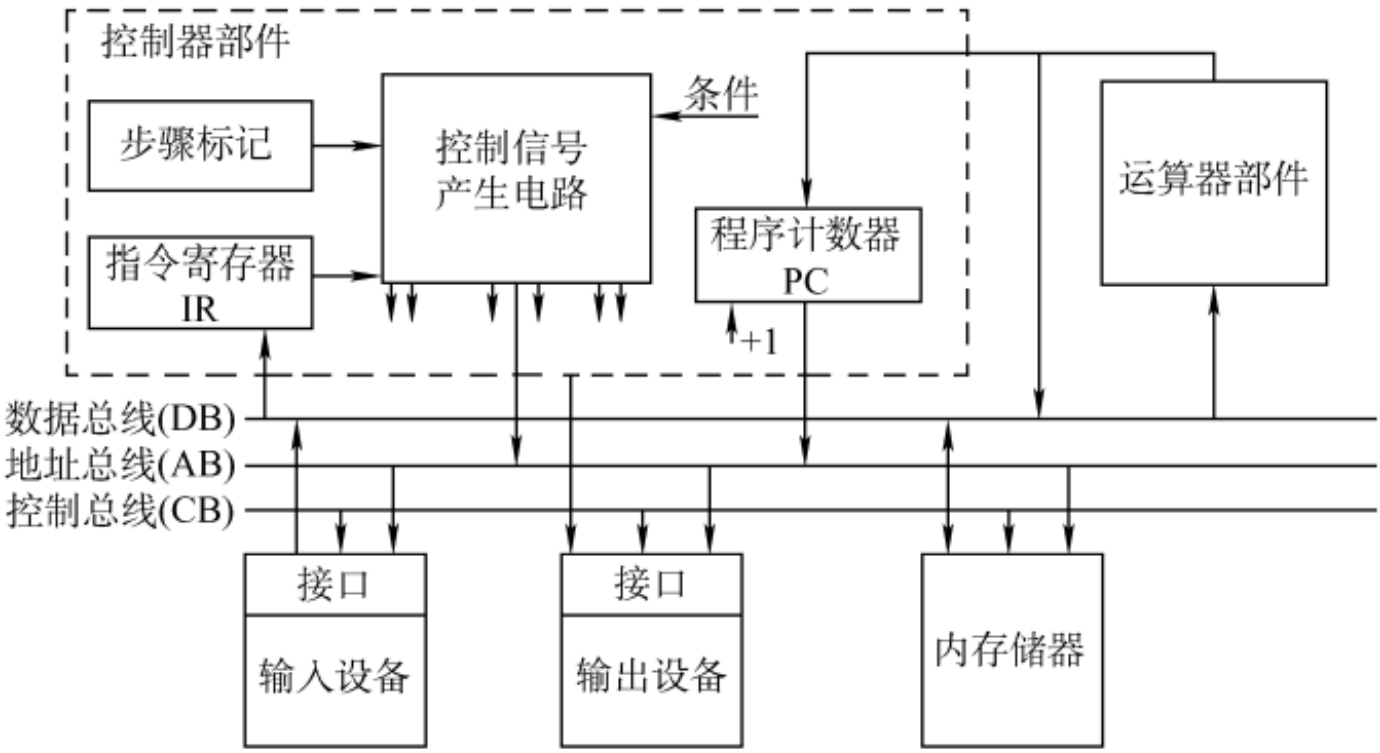
五段式指令流水线的执行过程

- 运算类指令
- LOAD指令
- STORE指令
- 条件转移指令
- 无条件转移指令

6 总线

基本概念

总线是一组能为多个部件分时共享的公共信息传送线路。



特性

- 1.机械特性：尺寸、形状、管脚数、排列顺序
- 2.电气特性：传输方向和有效的电平范围
- 3.功能特性：每根传输线的功能(地址、数据、控制)
- 4.时间特性：信号的时序关系

分类

按数据传输格式

- 串行总线
- 并行总线

按总线功能

数据通路表示的是数据流经的路径
数据总线是承载的媒介

1. 片内总线

片内总线是芯片内部的总线。
它是CPU芯片内部寄存器与寄存器之间、寄存器与ALU之间的公共连接线。

2. 系统总线

系统总线是计算机系统内各功能部件（CPU、主存、I/O接口）之间相互连接的总线。
按系统总线**传输信息内容**的不同，又可分为3类：**数据总线**、**地址总线**和**控制总线**。

1）**数据总线**用来传输各功能部件之间的数据信息，它是**双向**传输总线，其位数**与机器字长、存储字长有关**。

2）**地址总线**用来指出数据总线上的源数据或目的数据所在的主存单元或I/O端口的地址，它是**单向**传输总线，地址总线的位数**与主存地址空间的大小有关**。

3）**控制总线**传输的是控制信息，包括**CPU送出的控制命令**和**主存（或外设）返回CPU的反馈信号**。

3. 通信总线

通信总线是用于计算机系统之间或计算机系统与其他系统（如远程通信设备、测试设备）之间信息传送的总线，通信总线也称为外部总线。

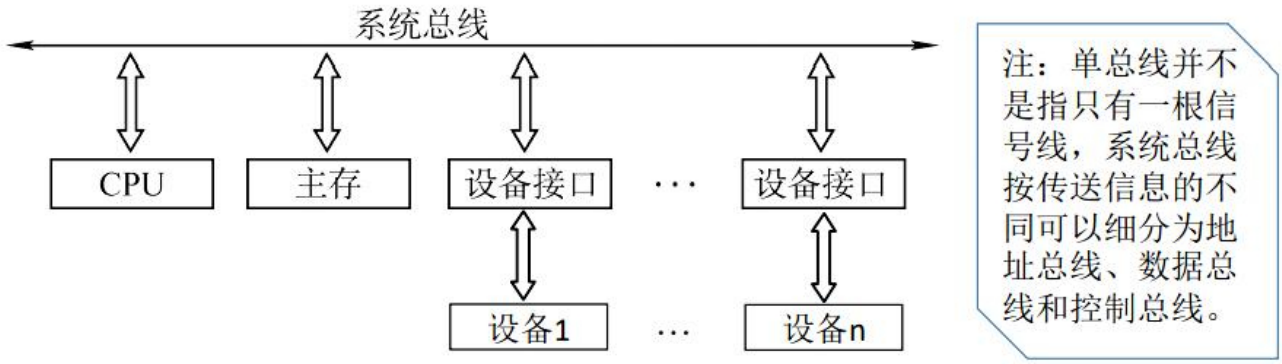
- 片内总线
- 系统总线
  - 数据总线
  - 地址总线
  - 控制总线
- 通信总线

按时序控制方式

- 同步
- 异步

总线结构

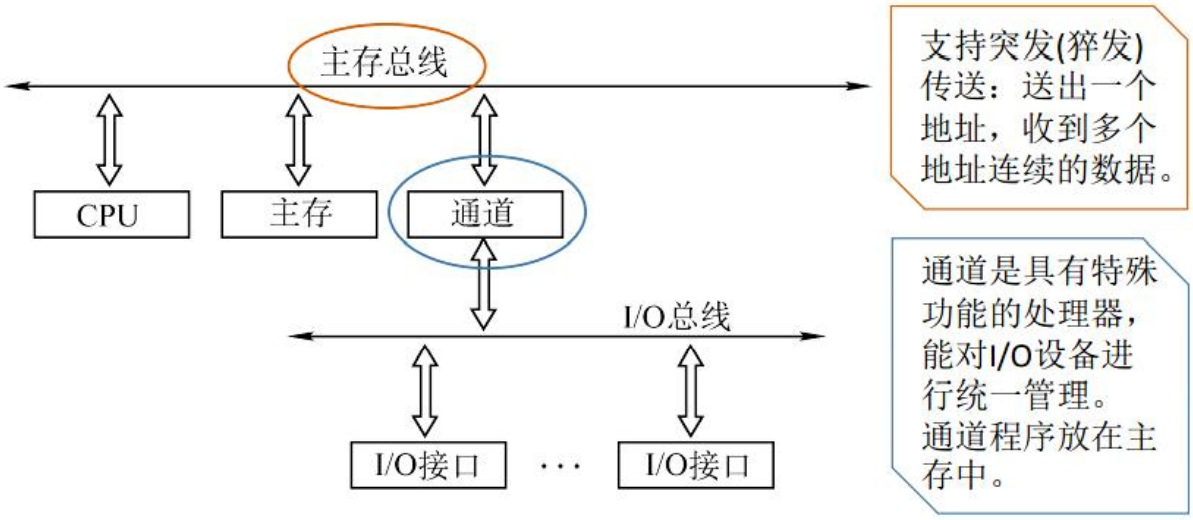
单总线结构



- **结构：**CPU、主存、I/O设备（通过I/O接口）都连接在**一组**总线上，允许I/O设备之间、I/O设备和CPU之间或I/O设备与主存之间直接交换信息。
- **优点：**结构简单，成本低，易于接入新的设备。
- **缺点：**带宽低、负载重，多个部件只能争用唯一的总线，且不支持并发传送操作。

- 系统总线

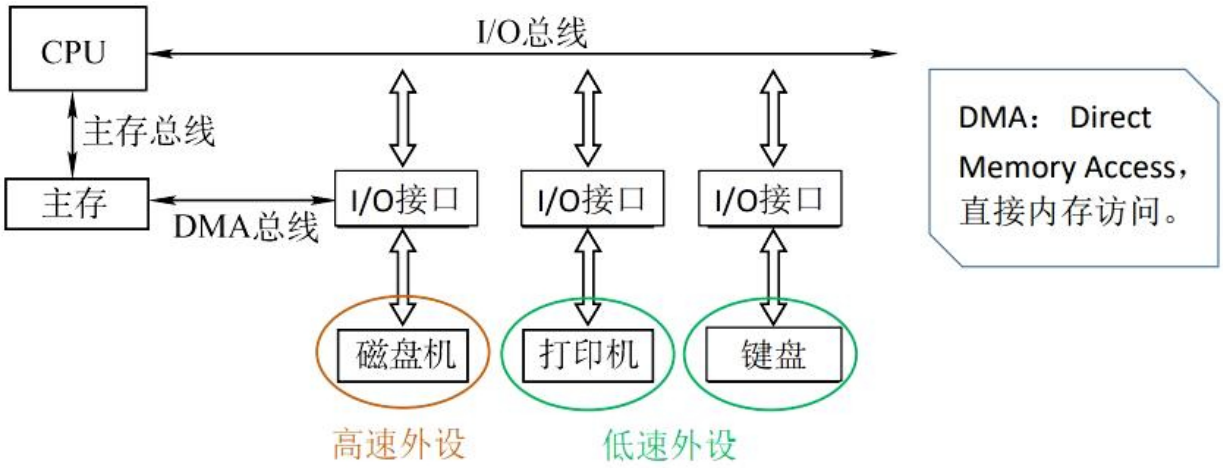
双总线结构



- **结构：**双总线结构有两条总线，一条是**主存总线**，用于CPU、主存和通道之间进行数据传送；另一条是**I/O总线**，用于多个外部设备与通道之间进行数据传送。
- **优点：**将较低速的I/O设备从单总线上分离出来，实现存储器总线和I/O总线分离。
- **缺点：**需要增加通道等硬件设备。

- 主存总线
- I/O总线

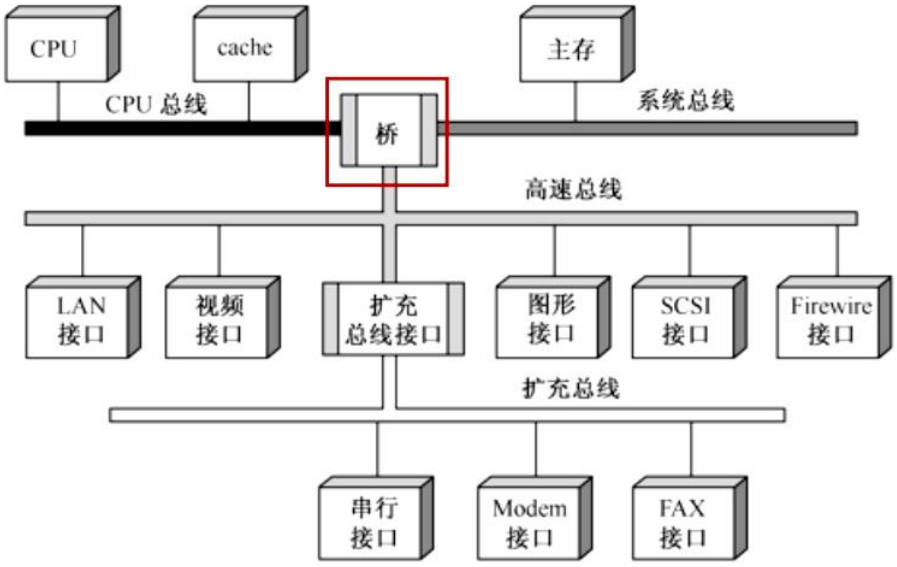
三总线结构



- **结构：**三总线结构是在计算机系统各部件之间采用3条各自独立的总线来构成信息通路，这3条总线分别为**主存总线**、**I/O总线**和直接内存访问**DMA总线**。
- **优点：**提高了I/O设备的性能，使其更快地响应命令，提高系统吞吐量。
- **缺点：**系统工作效率较低。

- 主存总线
- I/O总线
- DMA总线

四总线结构



1. 桥接器：用于连接不同的总线，具有数据缓冲、转换和控制功能。
2. 靠近CPU的总线速度较快。
3. 每级总线的设计遵循总线标准(见本章第4节)。

拓展：搜索“北桥芯片、南桥芯片”

性能指标

1.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。

2.总线时钟周期

- 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。

3.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。实际上指一秒内传送几次数据。

4.总线的时钟频率

- 即机器的时钟频率，为时钟周期的倒数。实际上指一秒内有多少个时钟周期。

5.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。

6.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)

7.总线复用

- 总线复用是指一种信号线在不同的时间传输不同的信息。可以使用较少的线传输更多的信息，从而节省了空间和
  成本。

8.信号线数

- 地址总线、数据总线和控制总线3种总线数的总和称为信号线数。

仲裁

- 多个主设备同时竞争主线控制权时，以某种方式选择一个主设备优先获得总线控制权称为总线仲裁。

仲裁方式 / 对比项目	链式查询	计数器定时查询	独立请求
控制线数	3 总线请求：1 总线允许：1 总线忙：1	$\lceil \log_2 n \rceil + 2$ 总线请求：1 总线允许：$\lceil \log_2 n \rceil$ 总线忙：1	2n+1 总线请求：n 总线允许：n 总线忙：1
优点	优先级固定 结构简单，扩充容易	优先级较灵活	响应速度快 优先级灵活
缺点	对电路故障敏感 优先级不灵活	控制线较多 控制相对复杂	控制线多 控制复杂

“总线忙”信号的建立者是**获得总线控制权的设备**

集中仲裁方式

- 链式查询
- 计数器定时查询
- 独立请求

分布仲裁方式

- 不需要中央仲裁器，每个潜在的主模块都有自己的仲裁器和仲裁号，多个仲裁器竞争使用总线

- 当设备有总线请求时，它们就把各自唯一的仲裁号发送到共享的仲裁总线上；
- 每个仲裁器将从仲裁总线上得到的仲裁号与自己的仲裁号进行比较；
- 如果仲裁总线上的号优先级高，则它的总线请求不予响应，并撤销它的仲裁号；
- 最后，获胜者的仲裁号保留在仲裁总线上。

总线操作和定时

总线传输的四个阶段

- 申请分配阶段
  - 传输请求
  - 总线仲裁
- 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。
- 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。
- 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。

定时

- 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

同步定时方式(同步通信)

- 由统一时钟控制数据传送
- 同步通信适用于总线长度较短及总线所接部件的存取时间比较接近的系统。

异步定时方式(异步通信)

- 在异步定时方式中，没有统一的时钟，也没有固定的时间间隔，完全依靠传送双方相互制约的“握手”信号来实现定时控制。
- 主设备提出交换信息的“请求”信号，经接口传送到从设备；从设备接到主设备的请求后，通过接口向主设备发出“回答”信号。
- 根据“请求”和“回答”信号的撤销是否互锁，分为以下3种类型。

- 不互锁方式
    - 速度最快 可靠性最差
- 半互锁方式
- 全互锁方式
    - 最可靠 速度最慢

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

- 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

标准

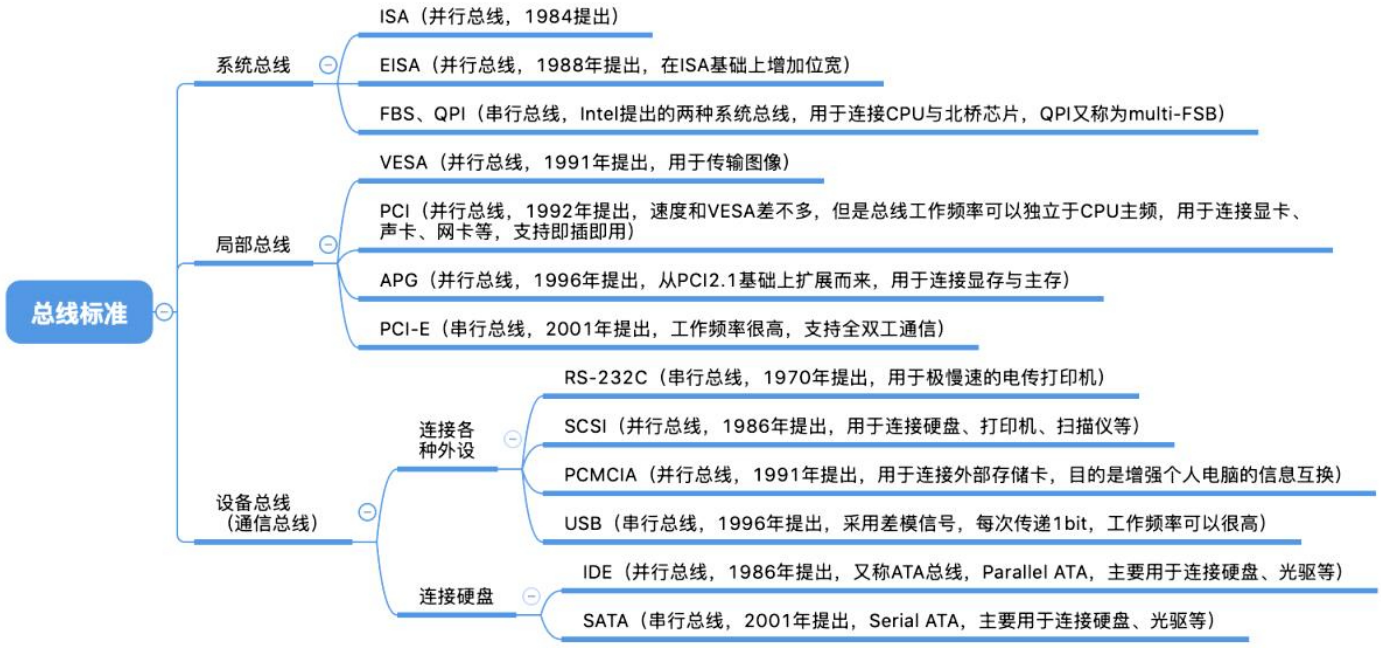
概念

- 系统总线：通常与CPU直接相连，用于连接CPU与北桥芯片、或CPU与主存等
- 局部总线：没有直接与CPU连接，通常是连接高速的北桥芯片，用于连接了很多重要的硬件部件(如显卡、声卡等)
- 设备总线、通信总线：通常由南桥芯片控制，用于连接计算机与计算机，或连接计算机与外部I/O设备

速度对比

总线标准	全称	工作频率	数据线	最大速度
ISA	Industry Standard Architecture	8MHz	8/16	8MB/s
EISA	Extended ISA	8MHz	32	32MB/s
PCI	Peripheral Component Interconnect	33MHz	32	133MB/s
AGP	Accelerated Graphics Port	-	-	X1：266MB/s X8：2.1GB/s
VESA	Video Electronics Standard Architecture	33MHz	32	132MB/s
PCI-E	PCI-Express（3GIO）	-	-	10GB/s以上
USB	Universal Serial Bus	-	-	1280MB/s
RS-232C	Recommended Standard	-	-	20Kbps
IDE（ATA）	Integrated Drive Electronics	-	-	100MB/s
SATA	Serial Advanced Technology Attachment	-	-	600MB/s
PCMCIA	Personal Computer Memory Card International Association	-	-	90Mbps
SCSI	Small Computer System Interface	-	-	640MB/s

标准总结



串行总线取代并行总线

并行总线：用m根线每次传送m个比特，用高/低电平表示1/0，通常采用同步定时方式，由于线间信号干扰，因此总线工作频率不能太高。另外，各条线不能有长度差，长距离并行传输时工艺难度大。

串行总线：用两根线每次传送一个比特，采用“差模信号”表示1/0，通常采用异步定时方式，总线工作频率可以很高。现在的串行总线通常基于包传输，如80bit为一个数据包，包与包之间有先后关系，因此可以用多个数据通路分别串行传输多个数据包。因此某种程度上现在的串行总线也有“并行”的特点。

7 输入输出系统

I/O系统基本概念

基本概念

IO硬件

- 输入设备、输出设备、外存设备、I/O接口(I/O控制器)

IO软件

- IO指令
  - CPU执行的指令，用于控制IO接口或控制通道
- 通道指令
  - 通道执行的指令，与CPU机器指令不同

I/O控制方式

- 程序查询方式：CPU“忙等”慢速设备完成工作，二者串行工作
- 程序中断方式：设备准备数据时，CPU继续工作。设备准备好之后向CPU发出中断请求，CPU在指令周期的末位检查中断并做出中断响应(执行中断处理程序)
- DMA方式：主存与IO交换信息时由DMA控制器控制，传输完一整块数据才需要中断
- 通道方式：通过IO指令启动通道，通道执行通道指令序列，通道程序存放在主存中

外部设备

输入设备

- 鼠标
- 键盘

输出设备

显示器

- 分类
  - 阴极射线管(CRT)
    - 字符、图形、图像
    - 光栅扫描、随机扫描
  - 液晶(LCD)
  - 发光二极管(LED)
- 参数
  - 屏幕大小、分辨率、灰度级、刷新频率
  - 显示存储器(VRAM)
    - 容量 = 分辨率 x 灰度级位数
    - 带宽 = 分辨率 x 灰度级位数 x 帧频

打印机

- 击打式和非击打式
- 串行和行式
- 针式、喷墨式、激光

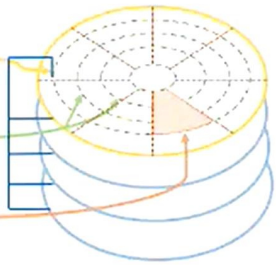
外存储器

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。

① 磁盘的容量：一个磁盘所能存储的字节总数称为磁盘容量。磁盘容量有非格式化容量和格式化容量之分。

非格式化容量是指磁记录表面可以利用的磁化单元总数。
格式化容量是指按照某种特定的记录格式所能存储信息的总量。

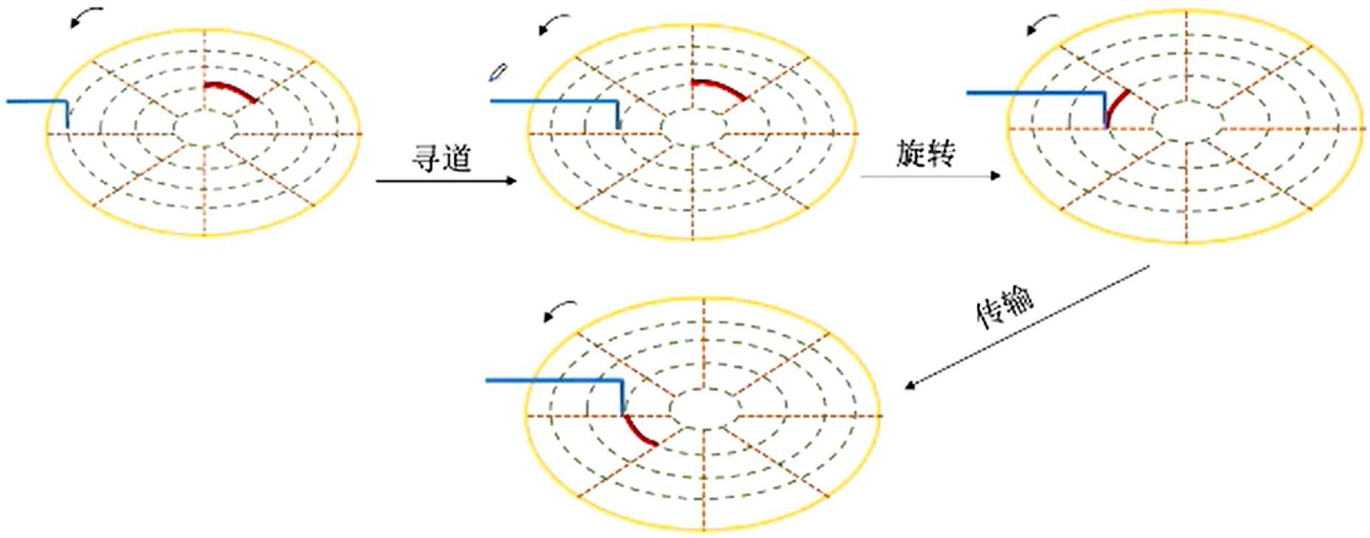
② 记录密度：记录密度是指盘片单位面积上记录的二进制的信息量，通常以道密度、位密度和面密度表示。

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。

注意：磁盘所有磁道记录的信息量一定是相等的，并不是圆越大信息越多，故每个磁道的位密度都不同。

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）+
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，称为数据传输率。

假设磁盘转数为r（转/秒），每条磁道容量为N个字节，则数据传输率为$D_r=rN$

磁盘存储器

- 磁记录原理
- 磁盘设备的组成
  - 存储区域：磁头(Heads)、柱面(Cylinders)、扇区(Sectors)
  - 硬盘存储器：磁盘驱动器、磁盘控制器、盘片
- 性能指标
  - 容量：格式化和非格式化
  - 记录密度：道密度、位密度、面密度
  - 平均存取时间：寻道时间 + 旋转延迟时间 + 传输时间
  - 数据传输率
- 磁盘地址：驱动器号+柱面(磁道)号+盘面号+扇区号
- 工作过程：寻址、读盘、写盘

磁盘阵列RAID

- RAIDO：无冗余和无校验的磁盘阵列。
- RAID1：镜像磁盘阵列。
- RAID2：采用纠错的海明码的磁盘阵列。
- RAID3：位交叉奇偶校验的磁盘阵列。
- RAID4：块交叉奇偶校验的磁盘阵列。
- RAID5：无独立校验的奇偶校验磁盘阵列。

光盘存储器

- CD-ROM：只读型光盘，只能读出其中内容，不能写入或修改。
- CD-R：只可写入一次信息，之后不可修改。
- CD-RW：可读可写光盘，可以重复读写。
- DVD-ROM：高容量的CD-ROM，DVD表示通用数字化多功能光盘。

固态硬盘SSD

- 采用Flash Memory记录数据，由E2PROM发展而来

IO接口

结构和作用

- 数据缓冲寄存器(DBR)
  - 暂存即将输入输出的数据
  - 主机和外设的速度匹配
- 状态/控制寄存器
  - 命令字：CPU对设备发出的具体命令
  - 状态字：设备的状态信息，供CPU检查
- 串并转换机构
  - 数据格式的转换
- I/O控制逻辑

    - 根据命令向设备发出控制信号
- 地址译码逻辑
    - 将地址信号映射到指定I/O端口

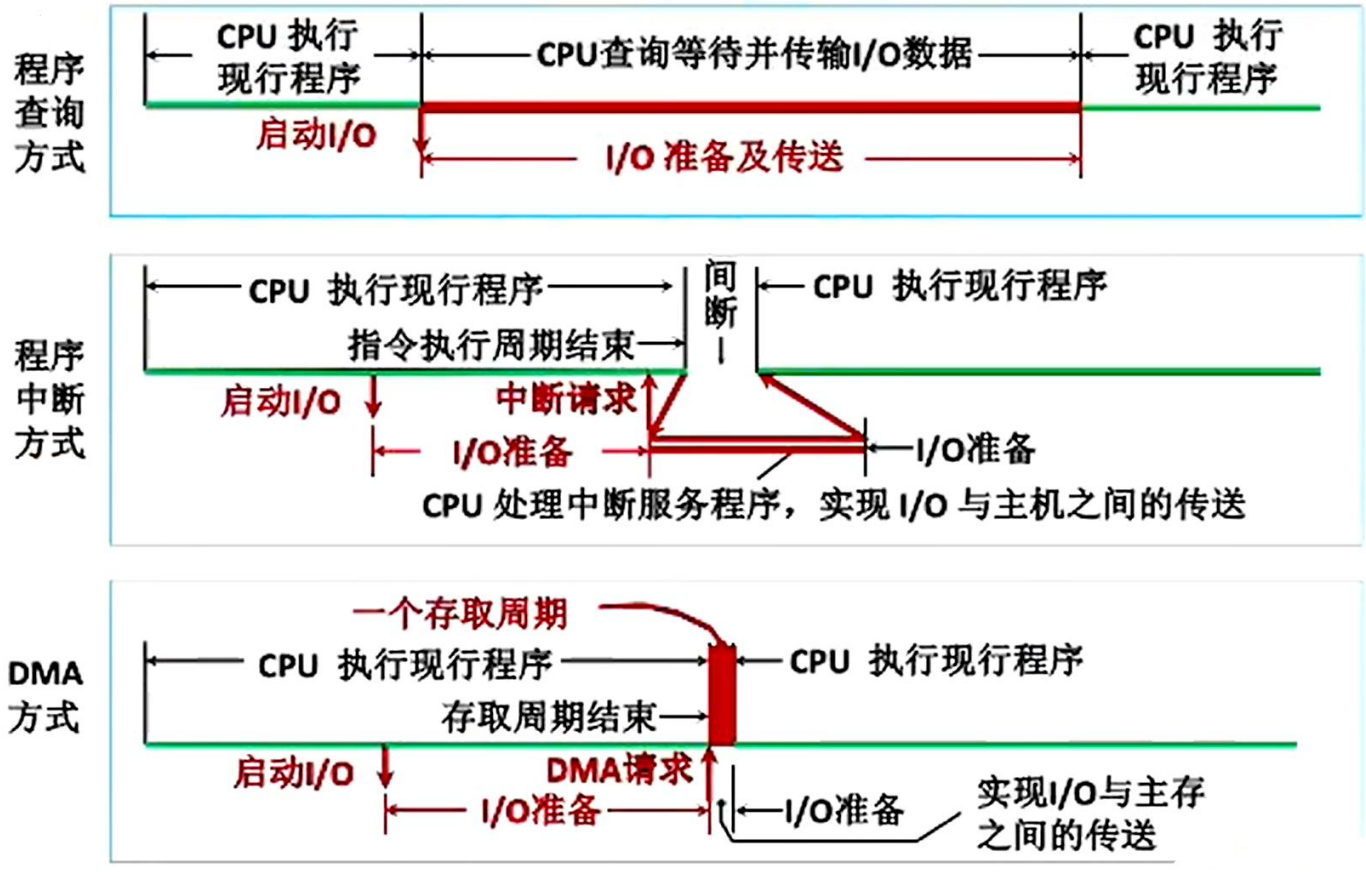
I/O端口

- 指IO控制器中可被CPU访问的寄存器
- 统一编制: IO端口和主存地址空间统一，用访存指令访问IO端口
- 独立编址: IO端口地址与主存地址相互独立，用IO指令访问IO端口

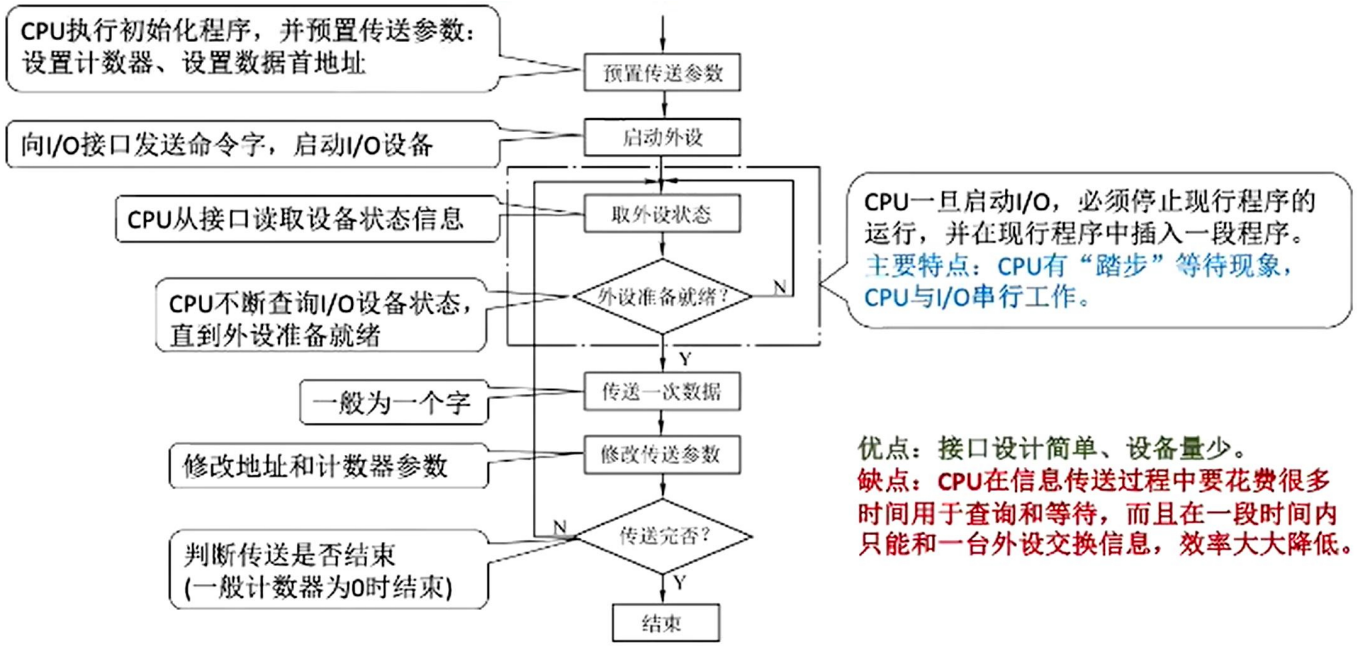
分类

- 并行接口、串行接口
- 程序查询接口、中断接口、DMA接口
- 可编程接口、不可编程接口

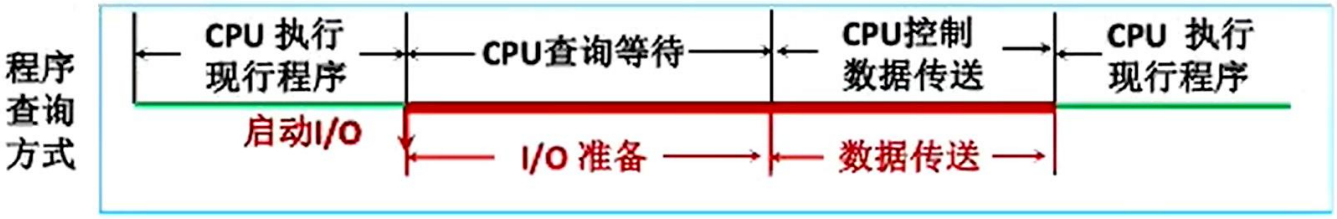
I/O方式



程序查询方式



优点：接口设计简单、设备量少。
缺点：CPU在信息传送过程中要花费很多时间用于查询和等待，而且在一段时间内只能和一台外设交换信息，效率大大降低。



CPU一旦启动I/O，必须停止现行程序的运行，并在现行程序中插入一段程序。
主要特点：CPU有“踏步”等待现象，CPU与I/O串行工作。
优点：接口设计简单、设备量少。
缺点：CPU在信息传送过程中要花费很多时间用于查询和等待，而且如果采用独占查询，则在一段时间内只能和一台外设交换信息，效率大大降低。

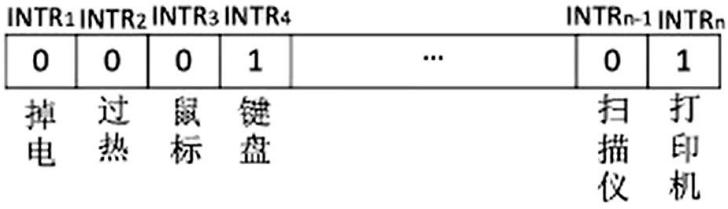
独占查询：CPU 100%的时间都在查询I/O状态，完全串行

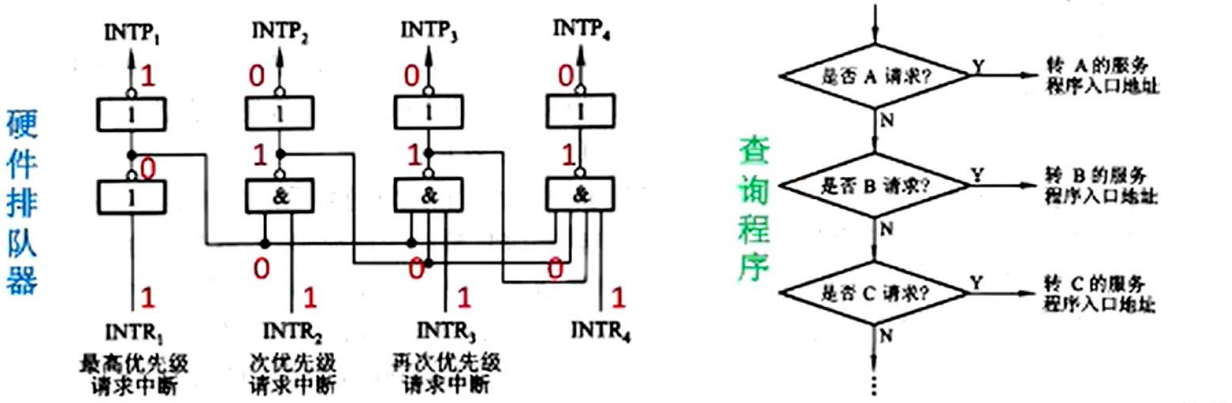
定时查询：在保证数据不丢失的情况下，每隔一段时间CPU就查询一次I/O状态。查询的间隔内CPU可以执行其他程序

程序中断方式

中断判优

中断判优既可以用硬件实现，也可用软件实现：
硬件实现是通过**硬件排队器**实现的，它既可以设置在CPU中，也可以分散在各个中断源中；
软件实现是通过**查询程序**实现的。

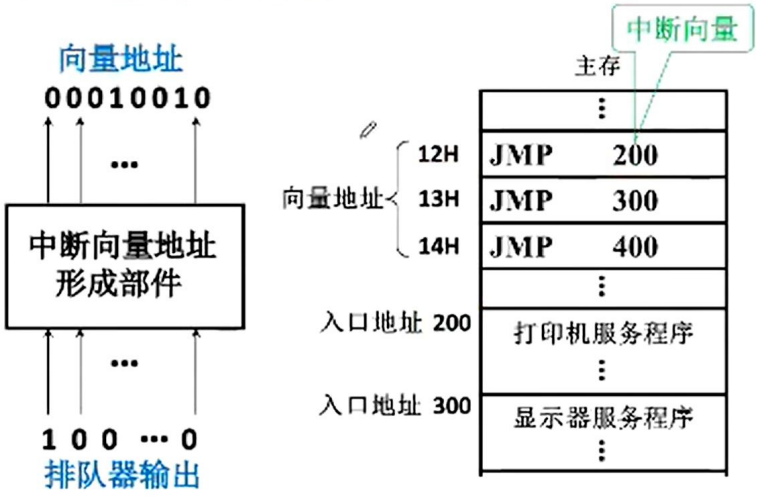




中断隐指令

- 保存原程序的PC值，并让PC指向中断服务程序的第一条指令

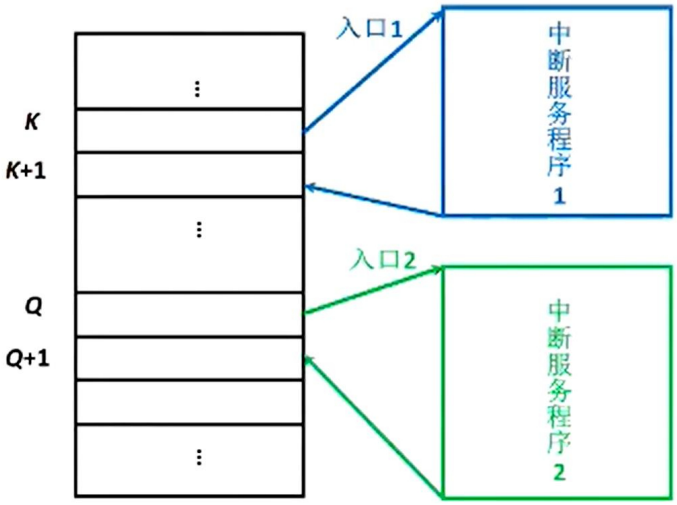
由 硬件 产生 向量地址
再由 向量地址 找到 入口地址



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）。

- 软件查询法
- 硬件向量法

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）

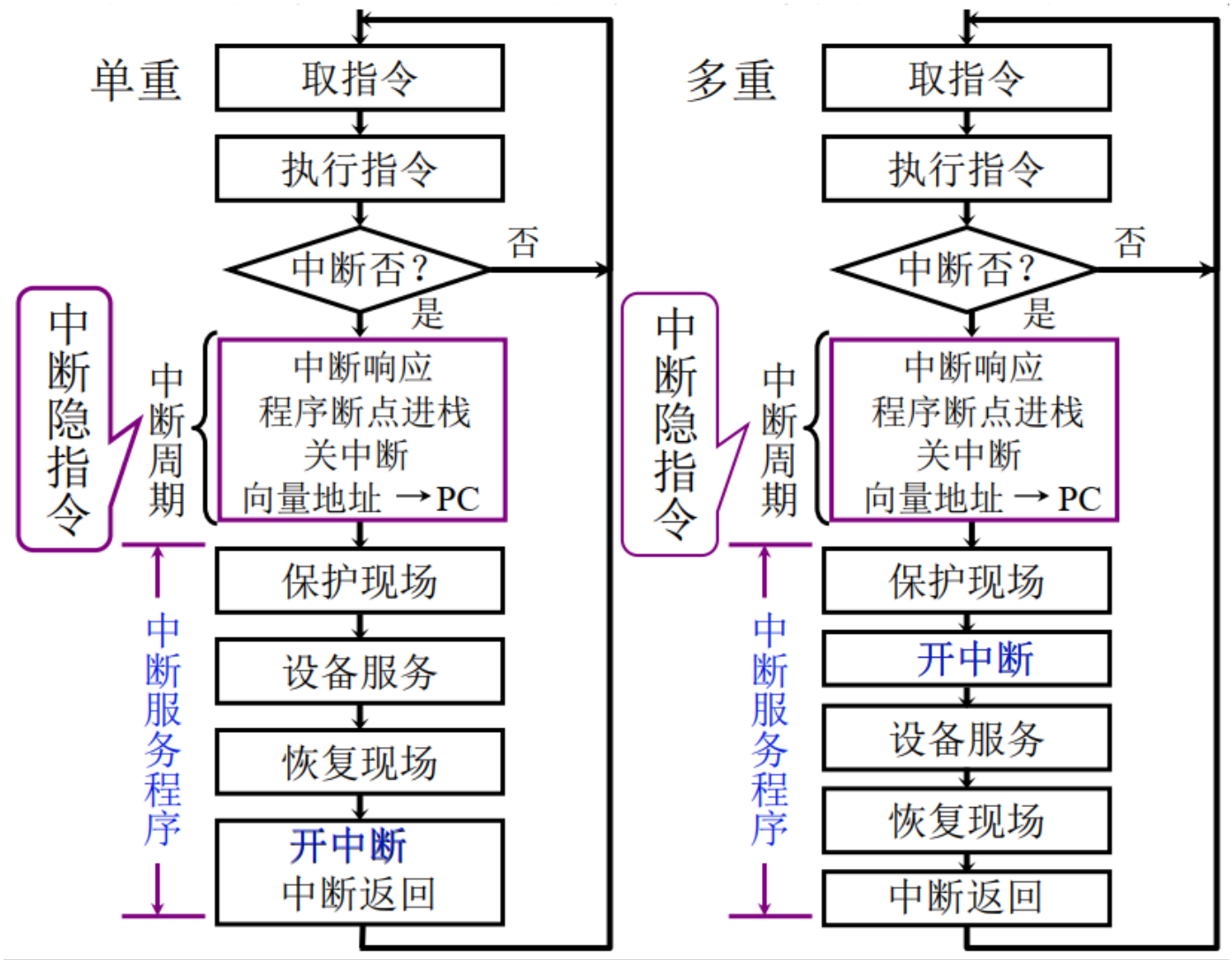
③ **恢复现场**

通过出栈指令或取数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。

中断处理过程



多重中断

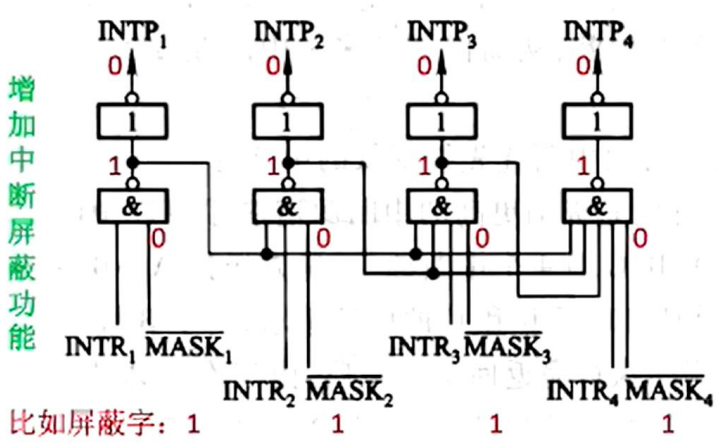
- 单重中断：执行中断服务程序时不响应新的中断请求。
- 多重中断：又称中断嵌套，执行中断服务程序时可响应新的中断请求。

	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

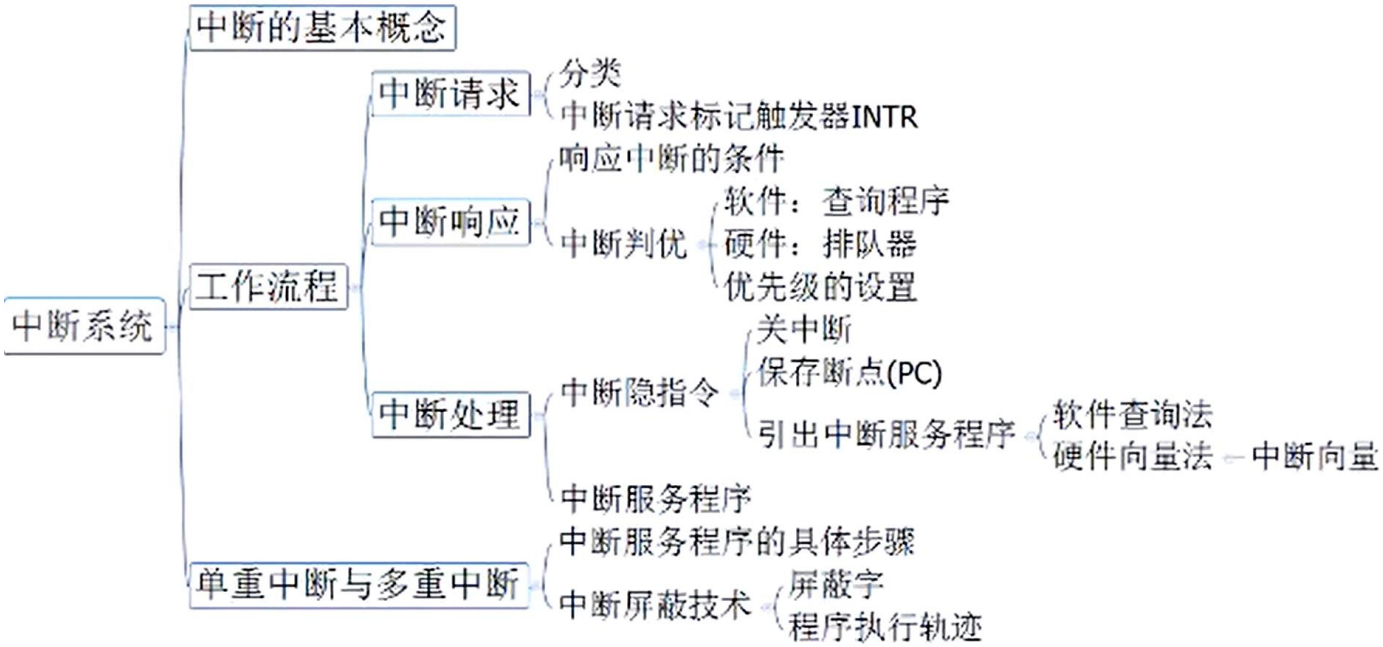
中断屏蔽技术

中断屏蔽技术主要用于多重中断，CPU要具备多重中断的功能，须满足下列条件。
① 在中断服务程序中提前设置开中断指令。
② 优先级别高的中断源有权中断优先级别低的中断源。
每个中断源都有一个屏蔽触发器，1表示屏蔽该中断源的请求，0表示可以正常申请，所有屏蔽触发器组合在一起，便构成一个屏蔽字寄存器，屏蔽字寄存器的内容称为屏蔽字。

屏蔽字设置的规律：
1. 一般用'1'表示屏蔽，'0'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)。



总结



DMA方式

DMA控制器

- 主要功能
  - 传送前：接受外设的DMA请求，向CPU发出总线请求，接管总线控制权
  - 传送时：管理总线，控制数据传送，确定主存单元地址及长度，能自动修改对应参数
  - 传送后：向CPU报告DMA操作的结束
- 组成
  - 主存地址计数器：存放要交换数据的主存地址
  - 传送长度计数器：记录传送数据的长度
  - 数据缓冲寄存器：暂存每次传送的数据
  - DMA请求触发器：设备准备好数据后将其置位

- 控制/状态逻辑：由控制和时序电路及状态标志组成
- 中断机构：数据传送完毕后触发中断机构，提出中断请求

传送过程

- 预处理：CPU完成寄存器初值设置等准备工作
- 数据传送：CPU继续执行主程序，DMA控制器完成数据传送
- 后处理：CPU执行中断服务程序做DMA结束处理

传送方式

- 停止CPU访存：需要数据传送时，停止CPU访存，总线控制权交给DMA控制器
- 交替访存：将CPU周期分为DMA访存和CPU访存两个部分
- 周期挪用(周期窃取)：I/O设备需要访存时，挪用一个或几个存取周期

中断和DMA对比

	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