第四章 指令系统

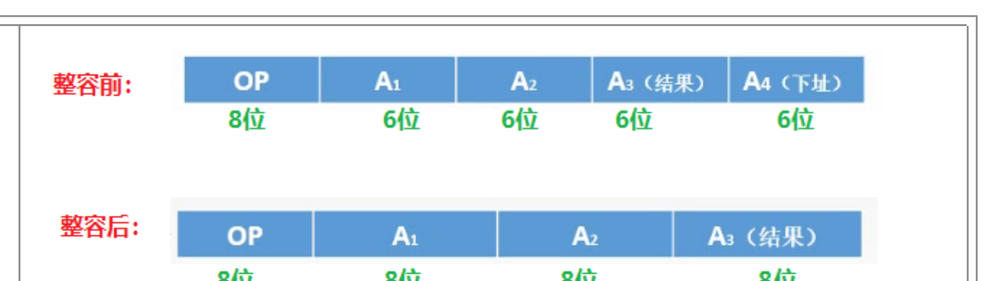
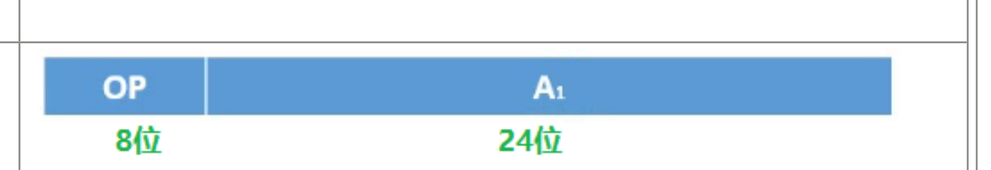
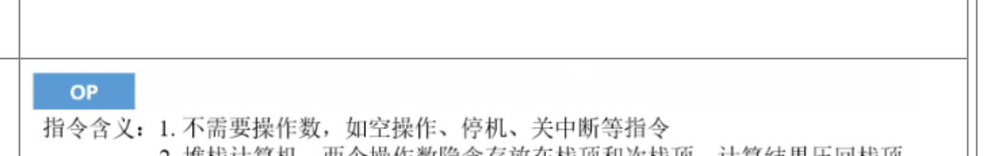
2022年9月17日 星期六 20:12

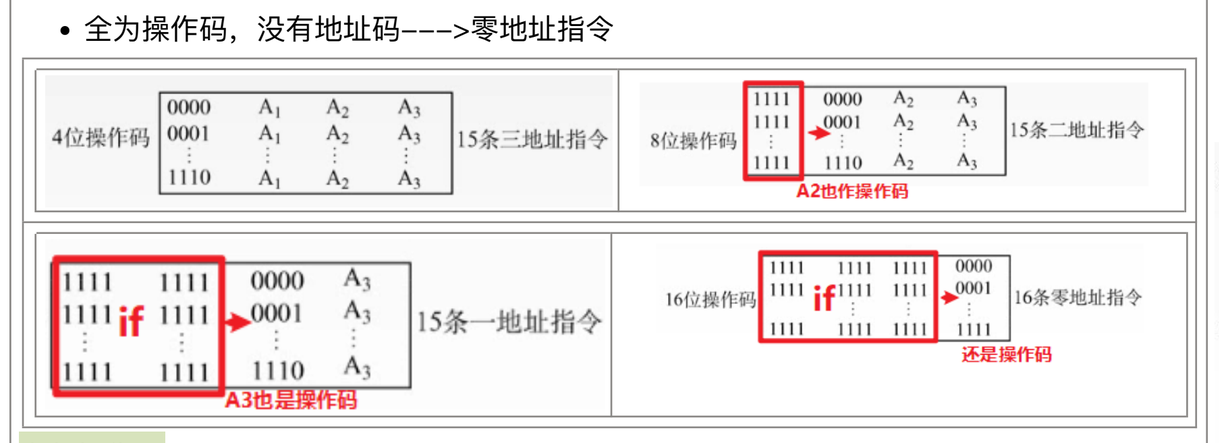
什么是指令?

- 指令【又称机器指令】 是指示计算机执行某种操作的命令，是计算机运行的最小功能单位
- 一条指令就是机器语言的一个语句，它是一组有意义的二进制代码
- 一台计算机的所有指令的集合构成该机的指令系统，也称为指令集
- 指令系统是计算机的主要属性，位于硬件和软件的交界面上

指令格式

基本指令结构	• 一条指令通常要包括操作码字段和地址码字段 • 【操作码字段】告诉用户做什么操作 • 【地址码】告诉用户对谁操作? • 指令的地址由程序计数器给出	
指令的分类	按指令长度分类 • 单字长指令：指令长度 = 机器字长 • 双字长指令：指令长度 = 2个机器字长 • 半字长指令：指令长度 = 半个机器字长	按是否定长分类 • 【定长指令字结构】执行速度快，控制简单 • 【变长指令字结构】指令的长度随指令功能而变 • 主存一般按字节编制，所有指令字长多为字节的整数倍
具体指令结构	右图指令字长32位 • 【操作码(OP)8位】+【地址码(A)共4个，每个6位】 指令访问内存的过程 • 首先000000这个位置上存放着操作指令 • A1，A2上存着两串数 • 他们在000000指令的执行下，要进行加法操作，将结果填入到A3中 • A3中的数据就是A1+A2的和 • 最后再去A4读取出指令，开始下一轮工作 内存中既有操作码，又有地址码,这样把他们放在一起并不好可以优化他们 • 把操作码放一起，地址码放一块 • 通过程序计数器使操作码+1顺序执行 优化后的好处 • 将操作码放一块，我们可以让程序执行完一步就自动执行下一句指令 • 这样我们的指令就不用存放下一条指令的位置了 • 这样访存的次数少了一次，速度也会快点 • 如无例外（特例指的是跳转指令），执行完就直接下一条继续，也就是顺序执行	

地址码	【三，四地址指令】 • 原先每一条地址码最多只能寻址到2的6次方也就是64个地址 • 而现在地址搜寻的范围变成了2的8次方也就是256个地址 • 寻址的范围大大增加了 • (A1)OP(A2)---->A3 • A4 = 下一条将要执行指令的地址	
	【二地址指令】 • 对两个数进行操作完后结果覆盖到原来的地址上的数 • 如将A1和A2相加，结果返回到A1 • (A1)OP(A2)--->A1	
	【一/单地址指令】 第一种 • 只有目的操作数的单操作数指令 • 进行自身操作的数(比如自增、自减、取反，求补) • OP(A1)---->A1 第二种 • 蕴含约定目的地址的双操作数指令 • 地址码A1指明一个操作数 • 另一个操作数来自隐含寻址，由ACC提供 • (ACC)OP(A1)---->ACC	
	【零地址指令】 • 只给出操作码OP，没有显示地址 第一种 • 不需要操作数的指令：空操作指令，停机指令，关中断指令 第二种 • 堆栈计算机中，两个操作数来自堆栈的栈顶和次栈顶单元	

操作码	拓展操作码（考计算） • 保持指令字长度不变而增加寻址空间 • 如果我们将前面的4位全部用作操作码，则一共能发出0000~1111【16种操作】 • 现在舍弃一条操作（1111），只发出0000~1110 【15种操作】 • 将1111留着作为标记，如果是1111开头的，则代表A1也作操作码(下图标记有问题) • 将11111111留着作为标记，如果是1111 1111开头的，则代表A2也作操作码(下图标记有问题) • 全为操作码，没有地址码--->零地址指令  定长操作码 • 一般用在指令长度比较长的机器上，就不用那么拘束调来调去	总结 • 对使用频率较高的指令，分配较短的操作码 • 对使用频率较低的指令，分配较长的操作码 • 从而尽可能减少指令译码和分析的时间 • 各指令的操作码一定不能重复 • 拓展操作码不一定只能有一条，也就是说不一定只有1111作拓展操作码，对应15条地址，也可以1110、1111都做拓展码，留14条地址指令也行，甚至不要14条地址指令，只要13条、12条也可以，所有操作码的设计都符合下面的规则 • 设地址长度为n，上一层留出m条指令，下一层可扩展出$m \times 2^n$条指令 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：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：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，从而在满足需要的前提下，有效地缩短指令字长。 - 优：在指令字长有限的前提下仍保持比较丰富的指令种类； - 缺：增加了指令译码和分析的难度，使控制器的设计复杂化。
操作类型	【数据传送类】 • 进行CPU和主存之间的数据传送 • LOAD作用:把存储器中的数据放到寄存器中 • STORE作用:把寄存器中的数据放到存储器中 【运算类】 • 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 • 移位操作---算术移位、逻辑移位、循环移位(带进位和不带进位)	【程序控制类】 • 改变程序执行的顺序 • 「无条件转移JMP」「 条件转移BRANCH」 • 「调用CALL」「返回RETURN」「陷阱Trap」 • 调用指令和转移指令的区别：前者必须保存下一条指令的地址，当子程序执行结束时，根据返回地址返回到主程序继续执行，后者不需要返回 • 转移指令，子程序调用与返回指令用于解决变动程序中指令执行次序的需求，而不是数据调用次序的需求 【输入输出类】 • 进行CPU和I/O设备之间的数据传送 • 传送控制命令和状态信息

CISC和RISC

RISC相比于CISC的优点	• RISC更能充分利用VLSI芯片的面积 • RISC更能提高运算速度 • RISC便于设计，可降低成本，提高可靠性 • RISC有利于编译程序代码优化 • x86处理器属于CISC

	复杂指令系统计算机CISC	精简指令系统计算机RISC
特点	• 指令的长度不固定，指令格式多，寻址方式多 • 可以访存的指令不受限制 • 各种指令执行时间相差大，大多需要多个时钟周期才能完成 • 控制器大多数采用微程序控制	• 指令长度固定，指令格式种类少，寻址方式种类少，指令条数少 • 只有Load/Store指令访存，其他指令的操作在寄存器中进行 • CPU中通用寄存器的数量相当多 • 以硬布线控制为主，不用或少用微程序控制 • 有利于实现指令流水线的特点 　○ 指令格式规整且长度一致 　○ 指令和数据按边界对齐存放 　○ 只有Load/Store指令能访问存储器
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

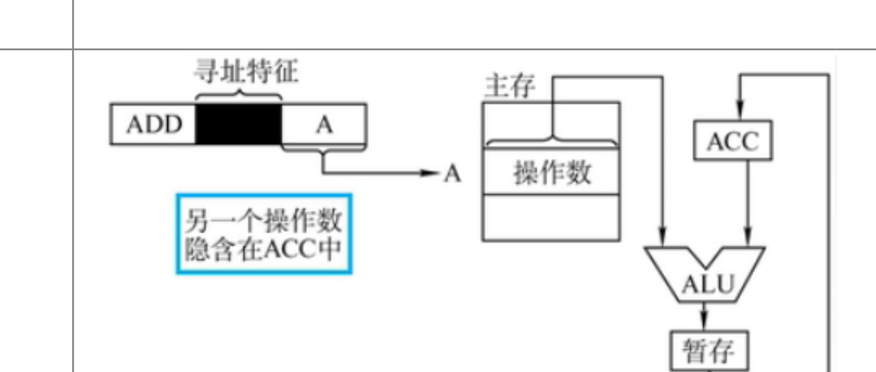
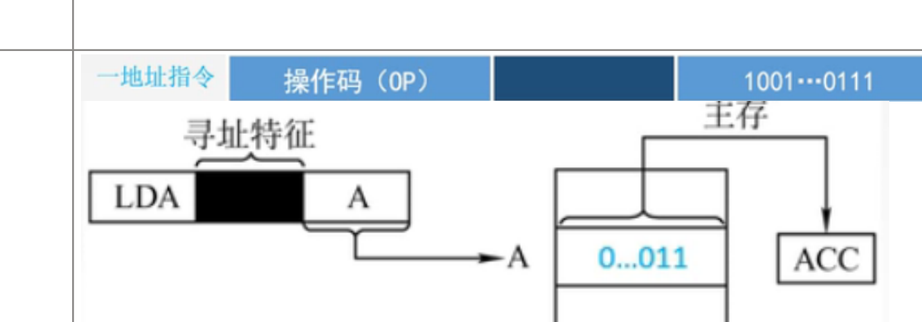
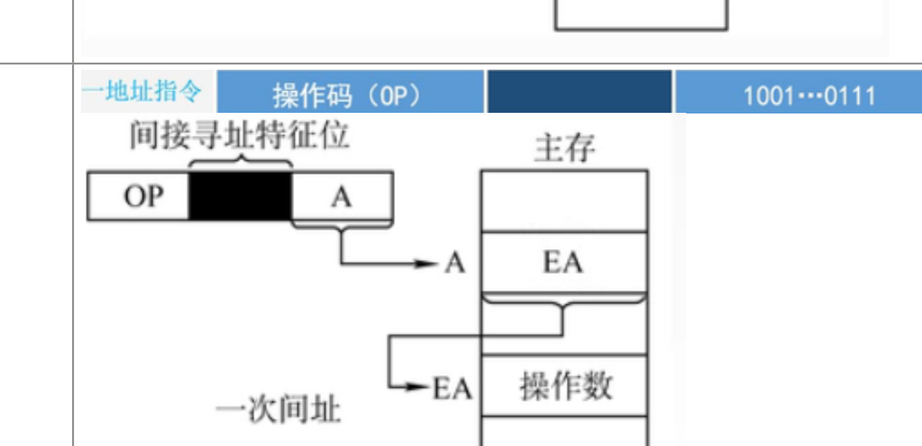
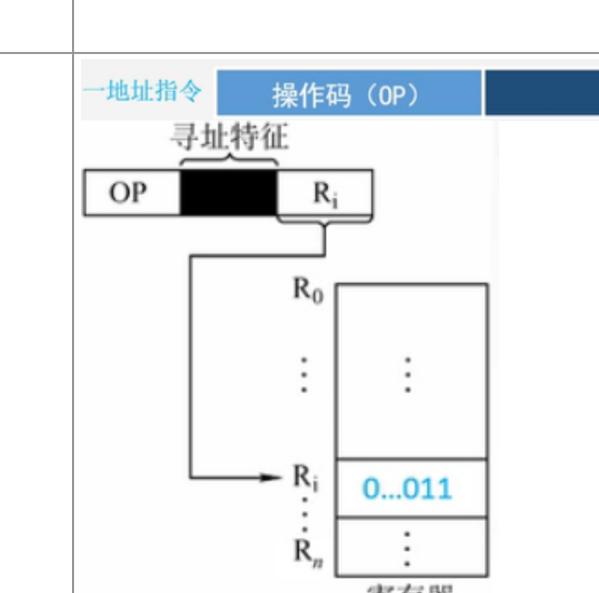
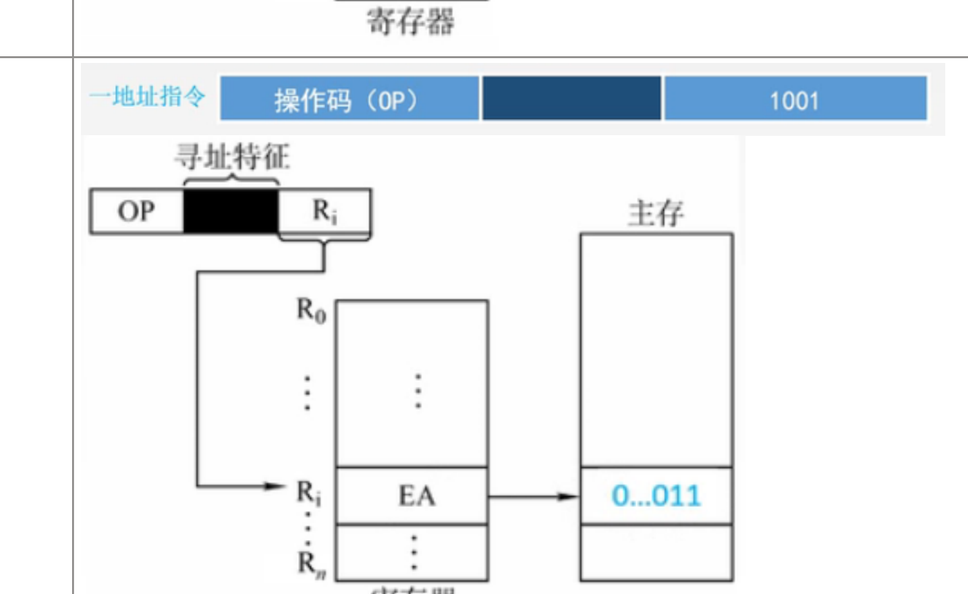
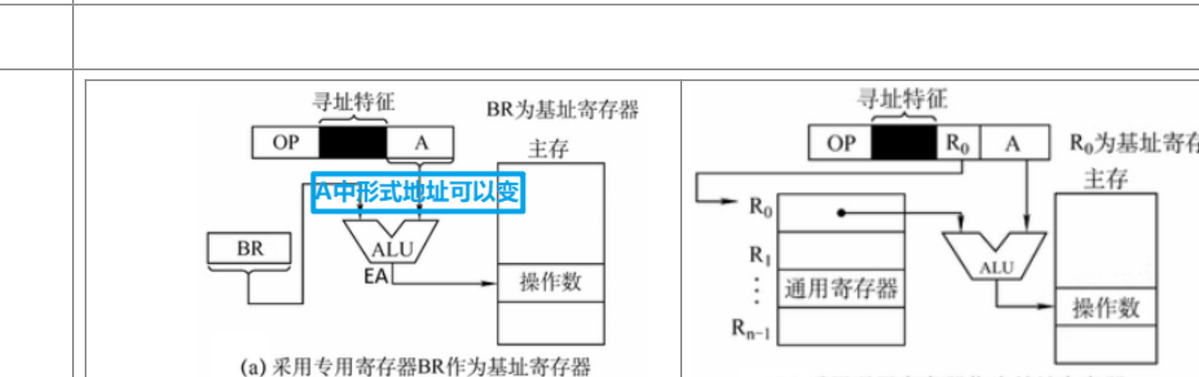
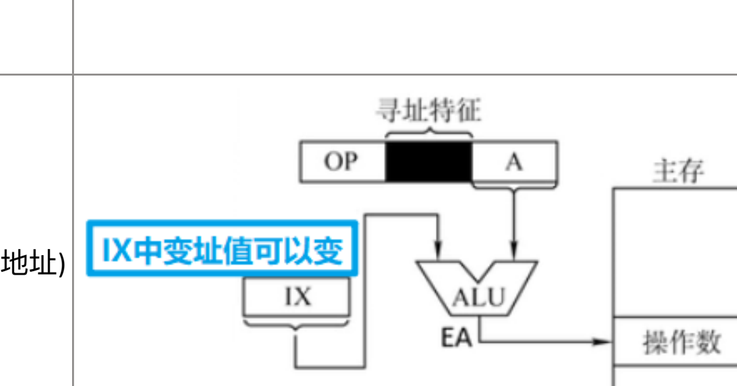
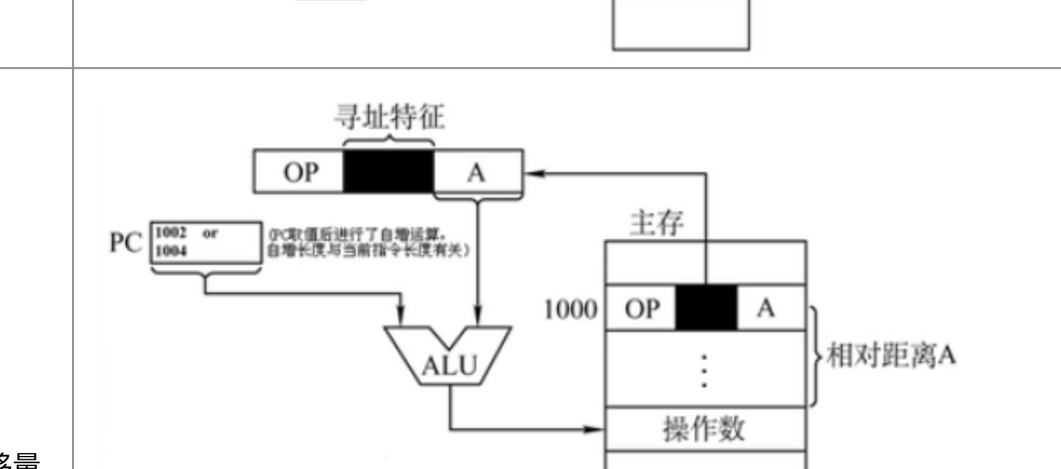
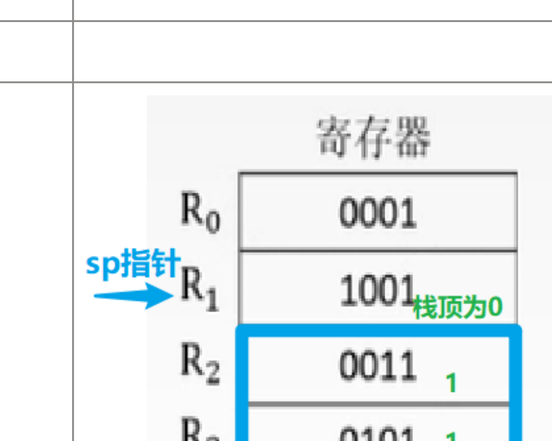
寻址方式

- 指令系统采用不同寻址方式的目的：缩短指令字长，扩大寻址空间，提高编程的灵活性

什么是指令寻址?	• 指令寻址就是寻找下一条将要执行的指令地址
什么是程序计数器	• 程序计数器pc是指让程序执行完一步就自动执行下一句指令的物理硬件 • 机器按字寻址，PC给出下一条指令字的访存地址(指令在内存中的地址)，因此PC的位数取决于存储器的字数 • 机器按字寻址，指令寄存器IR用于接收取得的指令，因此IR的位数取决于指令字长
指令寻址的种类	顺序寻址：• 通过程序计数器加1(1是指指令字长)，自动形成下一条指令的地址 跳跃寻址：• 通过转移类指令（如相对寻址）实现，可用来实现程序的条件或无条件转移 • 跳跃：指下条指令的地址不由PC自动给出，而由本条指令给出下条指令地址的计算方式 • 跳跃的地址分为绝对地址【由标记符直接得到】和相对地址【相对于当前指令地址的偏移量】 • 跳跃的结果是当前指令修改PC值，所以下一条指令仍然通过PC给出 例图： 

什么是数据寻址?	• 数据寻址就是确认本条指令的操作数【是操作数不是操作码】地址
地址码的组成	• 地址码 = 寻址特征+形式地址 　○ 寻址特征 = 存的就是每个寻址方式上的蓝色小标，表示一种方式 　○ 形式地址A = 就是不是直接对应到存储器中的地址，是需要根据寻址特征的要求转换为对应存储器的地址 　○ 有效地址EA = 通过寻址特征和形式地址求出来的真正对应到存储器的地址 操作码（OP） 寻址特征 形式地址（A） ★A表示A这个地址，(A)表示地址为A里面的内容 ★EA=A表示形式地址A就是真实地址EA ★EA=(A)表示形式地址A的内容是真实地址EA

常见的数据寻址方式

	有效地址	知识点	示例图	例题
访问主存空间的👇				
隐含寻址	程序指定	【定义】 • 不直接给出操作数的地址，而是在指令中就隐含操作数的地址 【寻址过程】 • 形式地址A取出对应的一个操作数 • 而另一个操作数则是通过隐含寻址方式的指令设置，隐含在了ACC中 【特点】 • 隐地址不给出明显的操作数地址，而在指令中隐含操作数的地址 • 可以简化地址结构，如零地址指令		
立即寻址	A就是操作数	【定义】 • 把我们实际要操作的数，直接存放在形式地址中 【寻址过程】 • 寻址特征为#，代表立即寻址的意思 • 形式地址写的是操作数3的补码（011） 【特点】 • 立即寻址主要执行取指令访存1次，不需要执行指令访存 • 立即寻址速度第一，指令直接给出操作数	一地址指令 操作码（OP） # 0…011	
直接寻址	EA=A	【定义】 • 地址码字段给的是操作数的有效位置 • 可以根据这个有效位置去内存中寻找操作数 【特点】 • 直接寻址主要执行取指令访存1次，还有执行指令访存1次		第28题
间接寻址	EA=(A)	【定义】 • 地址码字段给的是：操作数有效地址所在存储单元的地址 • 我们需要去这个单元取操作数的地址码，再拿这个地址码去找操作数 【特点】 • 与直接寻址相比：间接寻址执行取指令访存1次，还要执行指令访存2次		第24题
访问寄存器的👇				
寄存器寻址	EA=R_i	【定义】 • 和直接寻址原理一样，只是把访问主存改为访问寄存器 【特点】 • 寄存器寻址主要执行取指令访存1次 • 由于访问的是寄存器因此不需要执行指令访存 • 访问寄存器会比访问主存快得多 • CPU中寄存器不是很多，用很短的编码就可以指定寄存器，能有效地缩短地址段的位数		
寄存器间接寻址	EA=(R_i)	【定义】 • 和访问主存的间接寻址原理相同 • 地址码字段给的是操作数所在的寄存器位置 • 可以根据这个地址去寄存器中找到操作数的有效地址 • 再去内存中寻找操作数 【特点】 • 寄存器间接寻址主要执行取指令访存1次 • 还有一次是寄存器执行指令访存1次		
转/偏移类寻址👇				
基址寻址	EA=(BR)+A	【定义】 • CPU中基址寄存器BR的内容+形式地址A＝有效地址 【特点】 • 基址寄存器不变(作为基地址)，改变的是形式地址A中的值(作为偏移量) • 不用专门的BR(基址寄存器)也行，可以用通用寄存器 • 原理一样，只不过需要给个编码定位到通用寄存器 【用处】 • 可以扩大寻址范围 • 原先只能寻址A的位数范围内的地址，有了基址寻址的方式 • 可以通过加上一个基地址从而在更大范围的空间内设计程序		第23题 第27题
变址寻址	EA=(IX)+A	【定义】 • 通过修改变址值IX从而达到取不同操作数的目的 【特点】 • 变址寄存器的内容可以改变(作为偏移量)，而形式地址A保持不变(作为基地址) • 变址寻址常用在一些有规律的操作上，比如遍历字符串，遍历数组 • 1条指令实现了基址寻址多条的功能		第22题 第24题 第26题
相对寻址	EA=(PC)+A	【定义】 • 相对寻址是基址寻址的变种，将基址寄存器BR改为程序计数器PC 【特点】 • 地址码中的A是相对于当前指令地址的位移量，用补码表示 • A的位数决定操作数的寻址范围 • 相对寻址有利于程序浮动，广泛用于转移指令和多道程序设计中 • 执行本条指令时，PC已完成加1操作，PC中保存的是下一条指令的地址 • 所以相对寻址的相对地址是以下条指令在内存中首地址为基准位置的偏移量 【见右边的例题！！！】		第13题 第17题
不常见的👇				
堆栈寻址		【定义】 • 把操作数存放在堆栈中，隐含的使用堆栈指针(SP)作为操作数地址 • SP指针指向栈顶的空单元 【特点】（与正常栈的出入栈顺序相反） • 入栈，先压入数据，再修改指针 • 出栈，先修改指针，再弹出数据		

常考点

1. 速度方面：立即寻址 > 寄存器寻址 > 直接寻址 > 寄存器间接寻址 > 间接寻址
2. 基址寻址和变址寻址的区别

	基址寻址	变址寻址
有效地址	EA=(BR)+A	EA=(IX)+A
寄存器内容	由操作系统或管理程序确定	由用户设定
程序执行过程中值是否可变	不可变	可变
特点	多用于多道程序设计和编制浮动程序	有利于处理数组问题和编制循环程序

程序的机器级代表表示【2022新增考点】【感觉就是学汇编语言🤣】【暂且放弃】