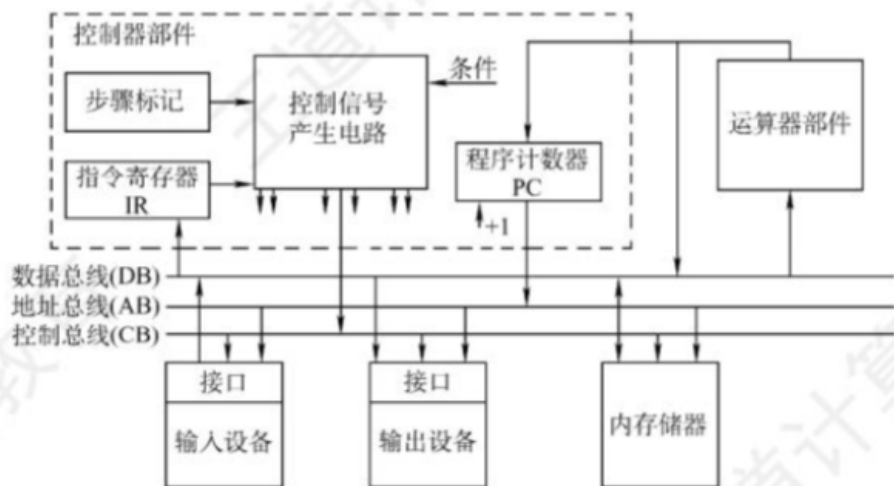


Lecture 20: 知识 - 控制器的功能和工作原理

控制器的结构

概念： 控制器结构

- 构成：
 - 1. 运算器部件通过数据总线与内存储器、输入设备和输出设备传送数据；
 - 2. 输入设备和输出设备通过接口电路与总线相连接；
 - 3. 内存储器、输入设备和输出设备从地址总线接收地址信息，从控制总线得到控制信号，通过数据总线与其他部件传送数据；
 - 4. 控制器部件从数据总线接收指令信息，从运算器部件接收指令转移地址，送出指令地址到地址总线，还要向系统中的部件提供它们运行所需要的控制信号；
- 图示：



概念： 控制器功能

- 功能：
 - 控制器是计算机系统的指挥中心，控制器的主要功能有：
 - 1. 从主存中取出一条指令，并指出下一条指令在主存中的位置；
 - 2. 对指令进行译码或测试，产生相应的操作控制信号，以便启动规定的动作；
 - 3. 指挥并控制 CPU、主存、输入设备和输出设备之间的数据流动方向；
- 分类：
 - 根据控制器产生微操作控制信号的方式的不同，控制器可分为硬布线控制器和微程序控制器，两类控制器中的 PC 和 IR 是相同的，但确定和表示指令执行步骤的办法及给出控制各部件运行所需要的控制信号的方案是不同的；

硬布线控制器

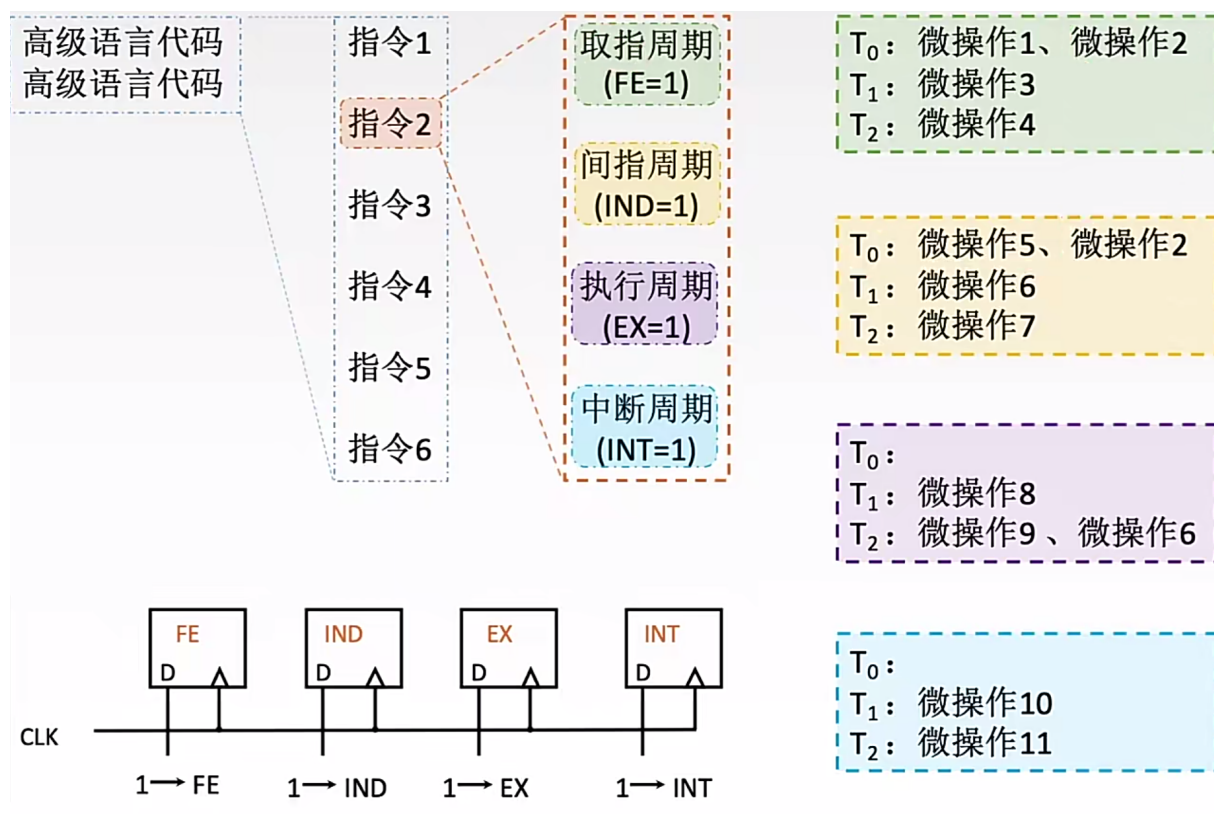
硬布线控制器基本概念

概念： 硬布线控制器下的指令周期

• 概念:

- 1. 一个节拍内可以并行的完成多个“相容的”微操作;
- 2. 同一个微操作可能在不同指令的不同阶段被使用;
- 3. 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);
- 4. 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;

• 图示:



概念: 硬布线控制器与微命令

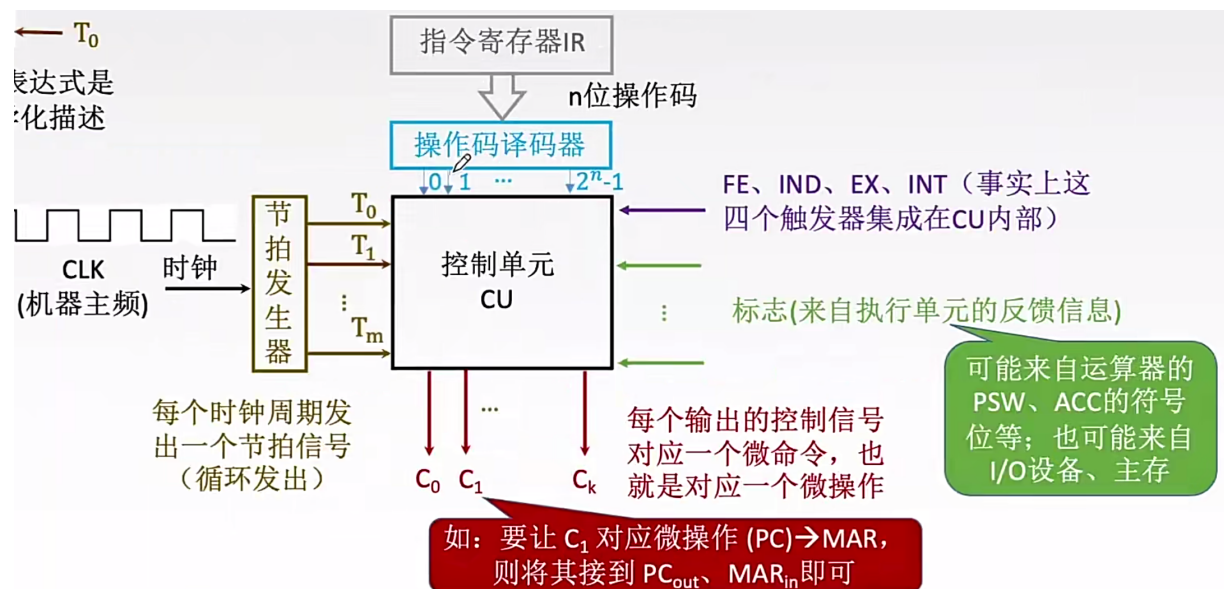
• 概念:

- 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”;
- 硬布线控制器由复杂的组合逻辑门电路和触发器构成, 也称组合逻辑控制器, 其原理是根据指令的要求、当前的时序及内外部的状态, 按时间的顺序发送一系列微操作控制信号;
- 指令的操作码是决定控制单元 (CU) 发出不同控制信号的关键;

• 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;

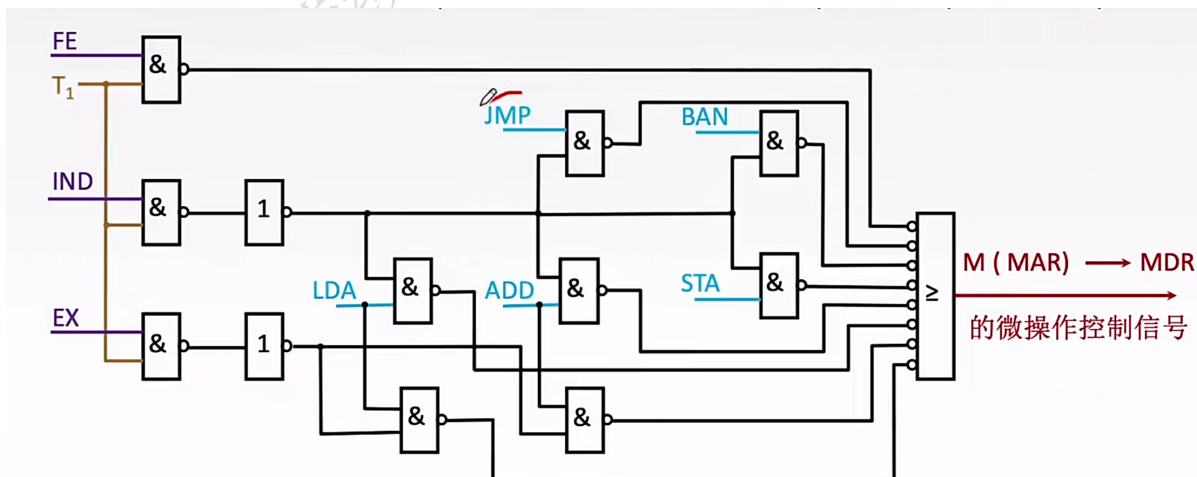
• 图示:



硬布线控制器微操作命令执行

示例：M(MAR)→MDR 微操作命令

- 命令：
 - $FE \cdot T_1 + IND \cdot T_1 (ADD + STA + LDA + JMP + BAN) + EX \cdot T_1 (ADD + LDA)$
- 解释：
 - FE：表示是在取指周期当中；
 - T1：表示是在 T1 这个指令周期当中；
 - IND：表示是在间址周期；
 - ADD+STA+LDA+JMP+BAN：执行的指令序列，来自指令译码器的对应的不同的线；
 - EX：执行周期
- 图示：



硬布线控制器的设计

方法：硬布线控制器的设计步骤

- 1. 分析每个阶段的微操作序列：取指、间址、执行、中断 四个阶段；
- 2. 选择 CPU 的控制方式：采用 定长 机器周期还是 不定长 机器周期？每个机器周期安排几个节拍？

- 3. 安排微操作时序;
- 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;

概念: 阶段一 - 分析每阶段的微操作序列

- 取指阶段:
 - $PC \rightarrow MAR$
 - $1 \rightarrow R$ // 发出"读"控制信号
 - $M(MAR) \rightarrow MDR$ // 取出指令
 - $MDR \rightarrow IR$ // 将取得的指令放到 IR 中
 - $OP(IR) \rightarrow ID$ // ID 是指令译码器, 从 IR 中取出操作码, 放入 ID ;
 - $(PC)+1 \rightarrow PC$ // PC 自加
- 间址周期:
 - $Ad(IR) \rightarrow MAR$ // 从 IR 中取出指令的地址码
 - $1 \rightarrow R$ // 发出"读"控制信号
 - $M(MAR) \rightarrow MDR$ // 根据 MAR 的地址信息去主存寻址
 - $MDR \rightarrow Ad(IR)$ // 将 MDR 中的数据放入 IR 的地址部分
- 执行周期:
 - CLA 指令:
 - $0 \rightarrow ACC$
 - $LDA X$: 把 X 所指向的内容取到 ACC
 - $Ad(IR) \rightarrow MAR$
 - $1 \rightarrow R$
 - $M(MAR) \rightarrow MDR$
 - $MDR \rightarrow ACC$
 - $JMP X$: 无条件转移
 - $Ad(IR) \rightarrow PC$
 - $BAN X$: 条件转移, 当 ACC 为负值时转移
 - $A0 \cdot Ad(IR) + A0 \cdot (PC) \rightarrow PC$

概念: 阶段二 - 选择 CPU 的控制方式

- 假设: 采取同步控制方式, 定长时间周期, 一个机器周期内安排三个节拍;

概念: 阶段三 - 安排微操作时序

- 原则:
 - 1. 微操作的先后顺序不得随意更改;
 - 2. 被控对象不同的微操作一个节拍完成尽量安排在一个节拍内完成;
 - 3. 占用时间较短的微操作尽量安排在一个节拍内完成, 并允许有先后顺序;
- 举例:
 - 排列前:

- (1) $PC \rightarrow MAR$
- (2) $1 \rightarrow R$ 存储器空闲即可
- (3) $M(MAR) \rightarrow MDR$ 在(1)之后
- (4) $MDR \rightarrow IR$ 在(3)之后
- (5) $OP(IR) \rightarrow ID$ 在(4)之后
- (6) $(PC) + 1 \rightarrow PC$ 在(1)之后

• 排列后：

- 因为 4 和 5 的操作都是在寄存器之间进行的，速度很快，符合原则三，所以可以放到一个周期当中；

- T_0 (1) $PC \rightarrow MAR$
- T_0 (2) $1 \rightarrow R$ 存储器空闲即可
- T_1 (3) $M(MAR) \rightarrow MDR$ 在(1)之后
- T_1 (6) $(PC) + 1 \rightarrow PC$ 在(1)之后
- T_2 (4) $MDR \rightarrow IR$ 在(3)之后
- T_2 (5) $OP(IR) \rightarrow ID$ 在(4)之后

概念：阶段四 - 电路设计

• 电路设计步骤：

- (1) 列出操作时间表：列出在 取指、间址、执行、中断周期 中， T_0 、 T_1 、 T_2 节拍内有可能用到的所有微操作；
- (2) 写出微操作命令的最简表达式；
- (3) 画出逻辑图；

• 第一步：列出操作时间表

• 取指周期 - 组合逻辑设计

工作周期标记	节拍	状态条件	微操作命令信号	CLA	COM	SHR	CSL	STP	ADD	STA	LDA	JMP	BAN
FE 取指	T_0		$PC \rightarrow MAR$	1	1	1	1	1	1	1	1	1	1
			$1 \rightarrow R$	1	1	1	1	1	1	1	1	1	1
	T_1		$M(MAR) \rightarrow MDR$	1	1	1	1	1	1	1	1	1	1
			$(PC) + 1 \rightarrow PC$	1	1	1	1	1	1	1	1	1	1
	T_2		$MDR \rightarrow IR$	1	1	1	1	1	1	1	1	1	1
			$OP(IR) \rightarrow ID$	1	1	1	1	1	1	1	1	1	1
		I	$1 \rightarrow IND$						1	1	1	1	1
		$\bar{I}$	$1 \rightarrow EX$	1	1	1	1	1	1	1	1	1	1

• 间址周期 - 组合逻辑设计

工作周期标记	节拍	状态条件	微操作命令信号	CLA	COM	SHR	CSL	STP	ADD	STA	LDA	JMP	BAN
IND 间址	T ₀		Ad (IR) → MAR						1	1	1	1	1
			1 → R						1	1	1	1	1
	T ₁		M(MAR) → MDR						1	1	1	1	1
	T ₂		MDR → Ad (IR)						1	1	1	1	1
		$\overline{\text{IND}}$	1 → EX						1	1	1	1	1

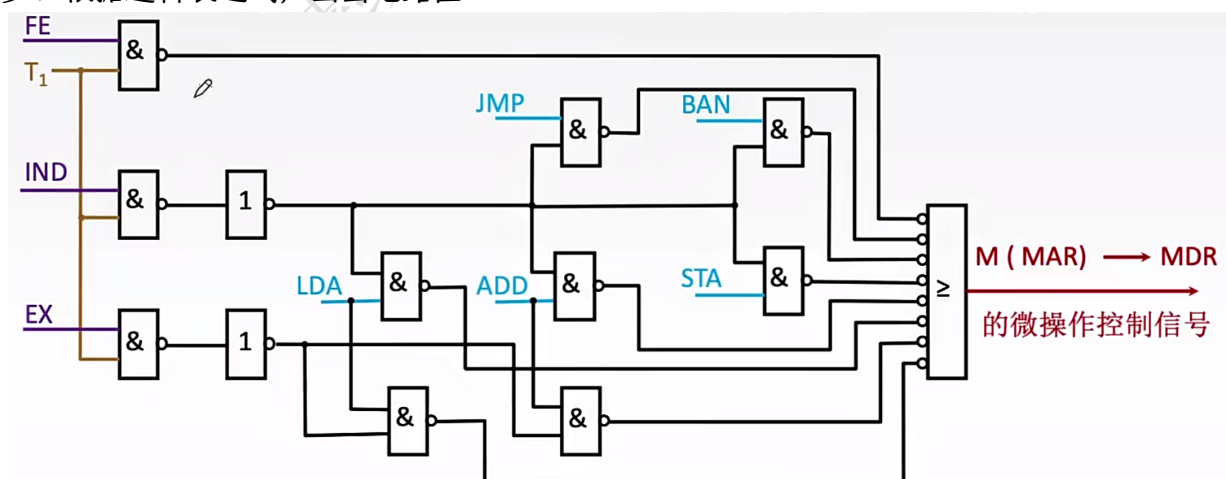
• 执行周期 - 组合逻辑设计

工作周期标记	节拍	状态条件	微操作命令信号	CLA	COM	ADD	STA	LDA	JMP	BAN
EX 执行	T ₀		Ad (IR) → MAR			1	1	1		
			1 → R			1		1		
			1 → W				1			
	T ₁		M(MAR) → MDR			1		1		
			AC → MDR				1			
	T ₂		(AC)+(MDR) → AC			1				
			MDR → M(MAR)				1			
			MDR → AC					1		
			0 → AC	1						
			$\overline{\text{AC}} \rightarrow \text{AC}$		1					
			Ad(IR) → PC						1	
	A ₀		Ad(IR) → PC							1

• 第二步：写出微操作命令的最简表达式

- $\text{FE} \cdot \text{T}_1 + \text{IND} \cdot \text{T}_1 (\text{ADD} + \text{STA} + \text{LDA} + \text{JMP} + \text{BAN}) + \text{EX} \cdot \text{T}_1 (\text{ADD} + \text{LDA} + \text{TFE} + \text{HND} (\text{ADD} + \text{STA} + \text{LDA} + \text{JMP} + \text{BAN}) + \text{EX} (\text{ADD} + \text{LDA}))$

• 第三步：根据逻辑表达式，画出电路图



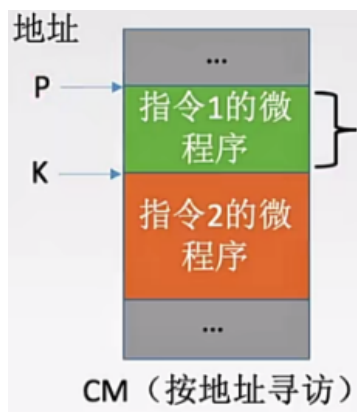
微程序控制器的基本原理

微指令与微指令控制器基本概念

概念：微指令

- 概念：

- 程序：
 - 由**指令序列**组成；
 - 由**一系列的机器指令**构成；
- 微程序：
 - 由**微指令序列**组成，每一种指令对应一个微程序；
 - **微程序和机器指令是一对一**的关系；
- 指令：
 - 是对程序执行步骤的描述；
- 微指令：
 - 微指令描述在执行当中都对应多少个微操作；
 - 一条**微指令**当中可能会包含多个**微操作**；
 - 真题：2024年
 - 微指令是存放在控制存储器（ROM）中由 0 和 1 组成的代码，可以由 CPU 直接执行；
- 微命令和微操作：
 - 微命令就等于微操作；
 - 一条微指令对应多个微操作；
- 微周期：
 - 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；
- 关系：
 - 程序：由指令序列构成 → 指令：对应一个微指令控制器中的**微程序** → 微程序：由**微指令序列**构成 → 微指令：由若干个**微操作**构成；
- 图示：



概念：微指令控制器的基本结构

- 概念：
 - 微地址形成部件
 - 根据当前要执行的机器指令的操作码，来确定当前其对应的微指令序列的起始地址位于哪；
 - 产生 **初始微地址** 和 **后继微地址**，以确保微地址指令的连续执行；
 - 控制存储器 CM
 - 真题：2017年
 - **存放微指令序列**；

- 控制存储器不等于主存，CM 在 CPU 的内部，主存在 CPU 外部；
- 用于存放各指令对应的微程序，控制存储器可用只读存储器 ROM 构成；
- 注意：CM 按地址寻址，不是按内容寻址。在 CPU 出厂时，就把所有的微地址写入；

- 微地址寄存器 CMAR

- 普通的地址寄存器（用于主存）：MAR
- 在 CPU 内部使用的微地址寄存器：CMAR
- 微地址寄存器，接收微地址形成部件送来的微地址，为在 CM 中读取微指令作准备；

- 微指令寄存器 CMDB

- 普通的地址寄存器（用于主存）：MDR
- 用于存放从 CM 中取出的微指令，它的位数同微指令字长相等；

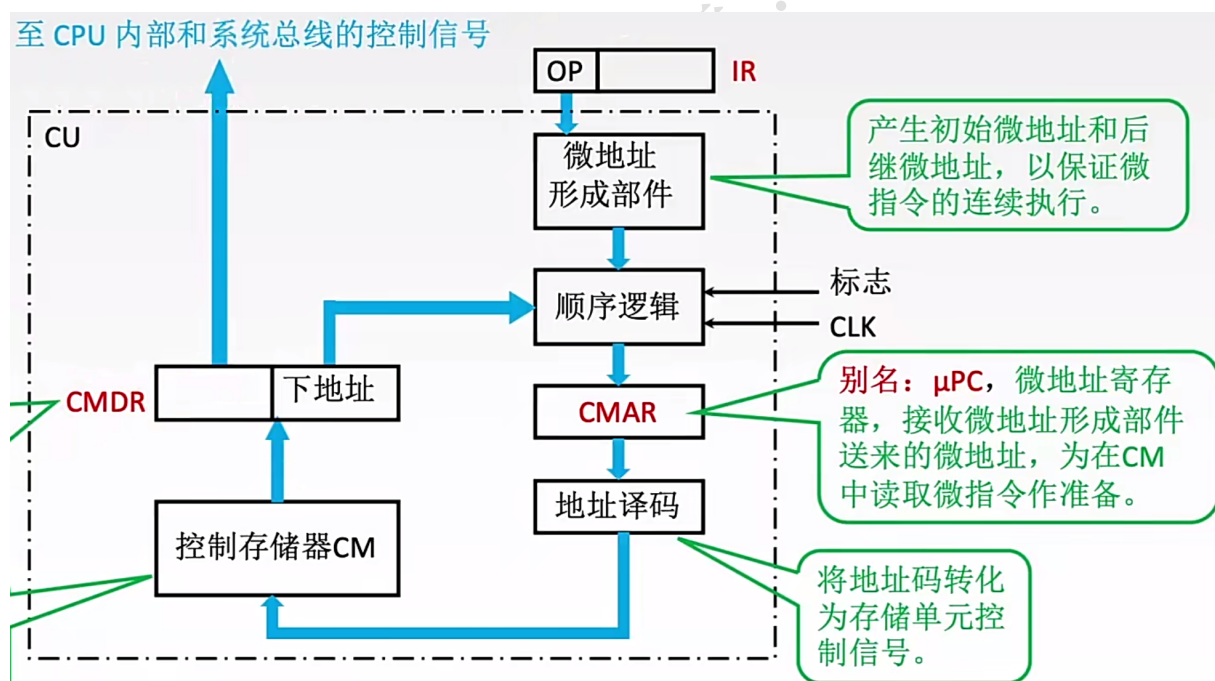
- 地址译码：

- 地址译码器，将地址码转化为存储单元控制信号；

- 顺序逻辑：

- 用于控制微指令的执行顺序的；

- 图示：

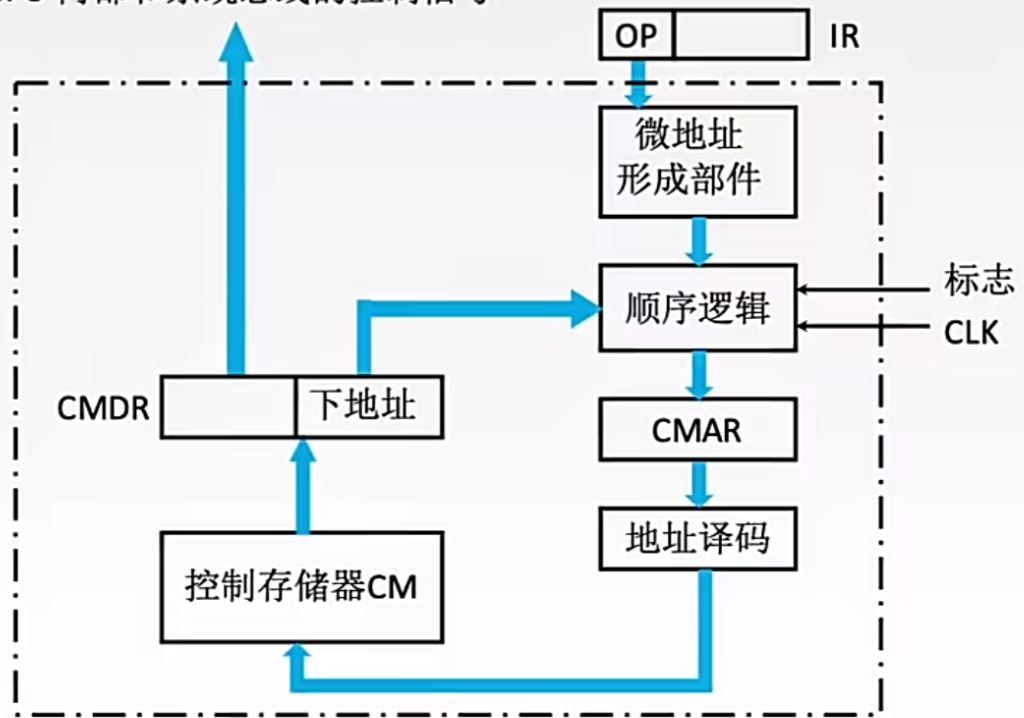


微程序控制器工作原理

概念：微程序控制器原理

- 概念：
 - 取指周期和中断周期的大部分指令都是共通的；
- 图示：
 - 微指令控制器工作过程

至 CPU 内部和系统总线的控制信号



概念：微指令控制器工作原理

- 概念：
 - 存放地址：
 - 控存的地址信息；
 - 下地址：
 - 下一条要执行的指令的地址；
- 分析：四个周期的执行
 - 取值周期：
 - 取址周期的指令从 #1 开始存放；
 - 执行 #1, #2, #3 这三条指令，然后进入下一条指令；
 - 注意：下一条指令并不一定是 #4，因为有很多的指令并不需要间接寻址，因此可以从其他指令开始执行；
 - 程序根据顺序逻辑中、输入的 标志 字段信息，了解到当前的寻址特征、是否需要跳过间址周期；
 - 间址周期：
 - 间址周期结束之后，会有一条指令、进入执行周期；
 - 执行周期：
 - 执行周期的微指令序列的存放，根据指令操作码确定；
 - 微地址形成部件后，知道执行指令的首地址，比如 #13 是 LDA X 指令的首地址，比如 #16 是 STA 指令的首地址；
 - 假设执行的是 LDA 指令，执行完成 #13 后，接下来执行 #14, #15
 - 中断周期：
 - 在中断周期结束后，会有一条指令、进入取指周期；

• 注意：

- 1. **取指周期、中断周期、间址周期**的微程序通常都是**公用**的，如果某指令系统中有 n 条机器指令，则 **CM** 中、微程序的个数至少为 $n+1$ 个；
- 2. 早期的 **CPU** 还有现在的一些物联网设备上，可能没有**间址周期**和**中断周期**；
- 3. 物理上，取指周期、执行周期看起来像是两个微程序，但逻辑上应该把它们看作是一个整体，可以总结为："一条指令对应一个微程序"；

• 图示：



微指令的设计

知识大纲

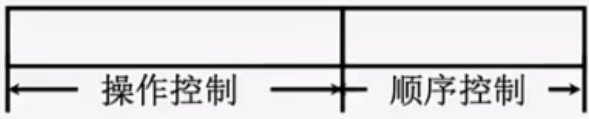
- 微指令格式：
 - 水平型；
 - 垂直型；
 - 混合型；
- 水平型微指令的编码方式：
 - 直接编码；
 - 字段直接编码；
 - 字段间接编码；
- 下一条微指令地址的形成方式：
 - 断定法；
 - 计数器法；
 - 指令操作码生成；
 - 硬件存储；

微指令的格式

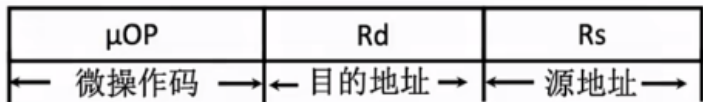
概念：微指令的格式

- 概念：
 - 微命令与微操作一一对应，一个微命令对应一根输出线；
 - 有的微命令可以**并行**执行，因此一条微指令可以包含多个微命令；
- 三种格式：

- 水平型微指令：
 - 概念：
 - 一条微指令能定义**多个**可并行的微命令；
 - 特点：
 - 微程序短，执行速度快；
 - 微指令长，编写微程序较麻烦；
 - 图示：



- 垂直型微指令：
 - 概念：
 - 一条微指令只能定义**一个**微命令，由微操作码字段规定具体功能；
 - 特点：
 - 优点：微指令短、简单、规整，便于编写微程序；
 - 缺点：微程序长，执行速度慢，工作效率低；
 - 图示：



- 混合型微指令：
 - 概念：
 - 在垂直型的基础上增加一些不太复杂的并行操作；

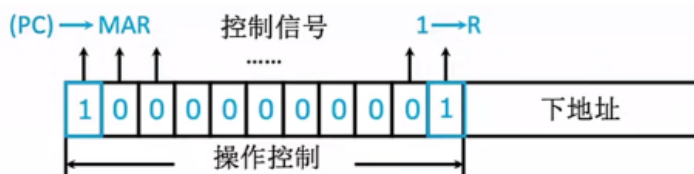
水平型微指令的编码方式

概念：水平型微指令的编码方式

- 概念：
 - 目的是**形成控制信号**；
 - 编码的目标是在保证速度的情况下，尽量**缩短微指令字长**；
- 方式：直接编码（直接控制）
 - 概念：
 - 在微指令的操作控制字段中，每一位代表一个微操作命令
 - 某位为“1”表示该控制信号有效；
 - 特点：

- 优点：简单、直观，执行速度快，**并行性好**；
- 缺点：微指令字长过长， n 个微命令就要求微指令的操作字段有 n 位，造成**控存容量极大**；

- 图示：



- 方式：字段直接编码

- 概念：

- 将微指令的控制字段分为若干个“段”，每段进行译码之后、发出控制信号；

- 补充：微命令字段分段原则

- 是否互斥：

- 互斥性微命令分在同一段内；
- 相容性微命令分在不同段内；
- 方便不同类的操作可以并行指令；

- 段信息：

- 每个小段中包含的信息位不能太多，否则将增加译码线路的复杂性和译码时间；

- 段留出状态：

- 一般每个小段还要留出一个状态表示**本字段不发出任何微命令**；
- 因此，当某字段的长度为 3 位时，最多只能表示 7 个互斥的微命令，通常用 000 表示不操作；

- 分析：控制字段位数

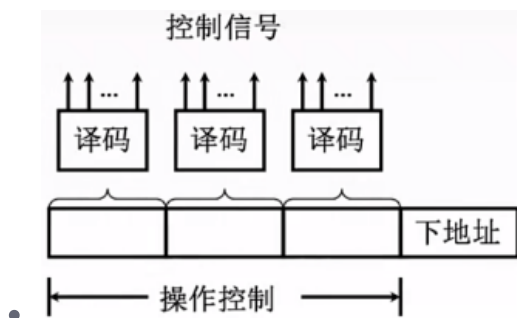
- 题目：

- **真题：**2012年
- 某计算机的控制器采用微程序控制方式，微指令中的操作控制字段采用字段直接编码法，共有 33 个微命令，构成 5 个互斥类，分别包含 7 、 3 、 12 、 5 和 6 个微命令，则操作控制字段至少有多少位？

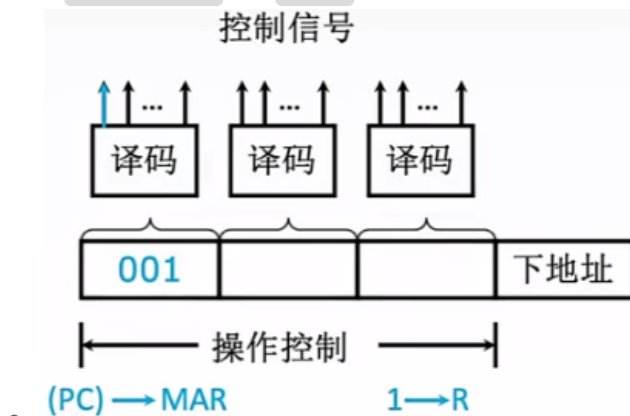
- 解析：

- 字段直接编码法将微命令字段分成若干小字段，互斥性微命令组合在同一字段中，相容性微命令分在不同字段中，每个字段还要留出一个状态，表示本字段不发出任何微命令；
- 5 个互斥类，分别包含 7 、 3 、 12 、 5 和 6 个微命令，需要 3 、 2 、 4 、 3 和 3 位，共 15 位；

- 图示：



- 举例: (PC)→MAR 和 1→R



- 方式: 字段间接编码

- 概念:

- 一个字段的某些微命令, 需由另一个字段中的某些微命令来解释;
- 由于不是靠字段直接译码发出的微命令, 故称为字段间接编码, 又称**隐式编码**;

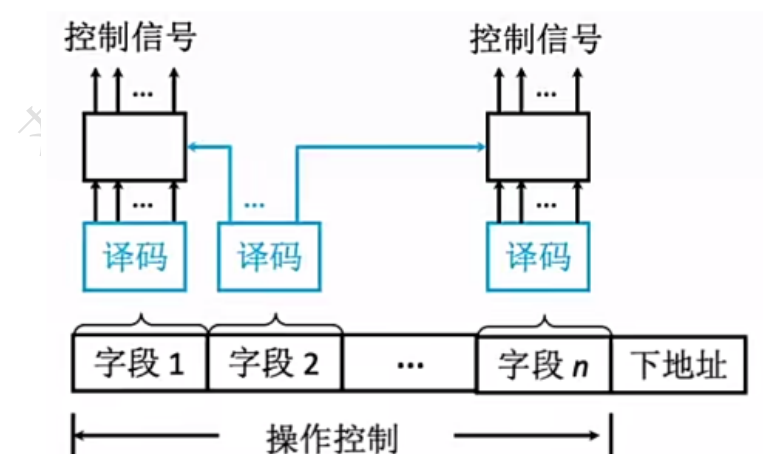
- 概念: 两次间接译码

- 字段 → 译码器 → 控制信号;

- 特点:

- 优点: 可进一步**缩短微指令字长**;
- 缺点: **削弱了微指令的并行控制能力**, 故通常作为字段直接编码方式的一种辅助手段;

- 图示:



微指令的地址形成方式

方法: 下地址字段 (断定方式)

- 概念：
 - 微指令格式中设置一个下地址字段，由微指令的下地址字段直接指出后继微指令的地址，这种方式又称为**断定方式**；
- 分析：
 - **真题**：2014年
 - 题目：
 - 某计算机采用微程序控制器，共有 32 条指令，公共的取指令微程序包含 2 条微指令，各指令对应的微程序平均由 4 条微指令组成，采用断定法（后继地址字段法）确定下条微指令地址，则微指令中后继地址字段的位数至少是多少？
 - 解析：
 - 由于每条指令平均对应 4 条微指令，因此 32 条指令共有 128 条微指令；
 - 由于公共的取指令微程序有两条微指令，因此一共有 $128+2=130$ 微指令；
 - 为了能够表示 130 条指令，需要至少 $2^8 = 256$ 即 8 位来表示所有指令，因此下地址字段至少需要 8 位；

方法：根据操作码形成

- 概念：当机器指令取至指令寄存器后，微指令的地址由**操作码经过微地址形成部件后形成地址**；

方法：增量计数器法

- 概念： $(CMAR)+1 \rightarrow CMAR$

方法：分支转移

- 概念：
 - 指明判别条件；
- 举例：
 - 转移地址 $\rightarrow$ 指明转移成功后的去向；

方法：由硬件产生微程序入口地址

- 第一条微指令地址：由专门硬件产生（用专门硬件记录取指周期微程序首地址）
- 中断周期：由硬件产生中断周期微程序首地址，用专门的硬件记录；

微指令控制单元设计

设计步骤

概念：设计步骤：

- 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
- 3. 确定微指令格式
- 4. 编写微指令码点

分析：阶段一 - 分析每个阶段的微操作序列

- 图示:

取指周期-硬布线控制器的节拍安排

T_0 PC $\rightarrow$ MAR
 T_0 1 $\rightarrow$ R
 T_1 M (MAR) $\rightarrow$ MDR
 T_1 (PC) + 1 $\rightarrow$ PC
 T_2 MDR $\rightarrow$ IR
 T_2 OP (IR) $\rightarrow$ ID

取指周期-微程序控制器的节拍安排

T_0 PC $\rightarrow$ MAR
 T_0 1 $\rightarrow$ R
 T_1 M (MAR) $\rightarrow$ MDR
 T_1 (PC) + 1 $\rightarrow$ PC
 T_2 MDR $\rightarrow$ IR
 T_2 OP (IR) $\rightarrow$ 微地址形成部件

分析: 阶段二 - 写出对应机器指令的微操作命令及节拍安排

- 步骤:

- 2.1 写出每个周期所需要的微操作;
- 2.2 补充微程序控制器特有的微操作;

- 子步骤 2.1 - 写出每个周期所需要的微操作

- 分析:

- Ad(CMDR) $\rightarrow$ CMAR :

- 当前执行的指令存在 CMDR 当中, 当需要执行下一条指令时, 可以从当前指令的“下一条地址”部分取出地址信息, 即 Ad(CMDR), 然后放入 CMAR, 进行下一条指令的执行;

- 取指周期最后指令:

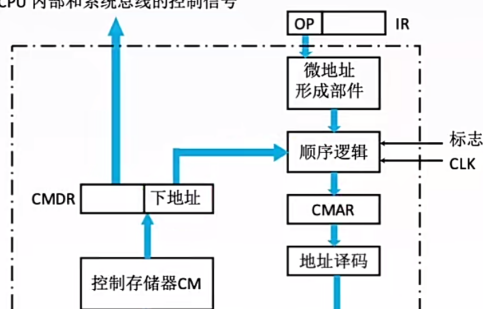
- 微地址形成部件 $\rightarrow$ CMAR

- 图示:

取指周期-硬布线控制器的节拍安排

T_0 PC $\rightarrow$ MAR
 T_0 1 $\rightarrow$ R
 T_1 M (MAR) $\rightarrow$ MDR
 T_1 (PC) + 1 $\rightarrow$ PC
 T_2 MDR $\rightarrow$ IR
 T_2 OP (IR) $\rightarrow$ ID

至 CPU 内部和系统总线的控制信号



取指周期-微程序控制器的节拍安排

T_0 PC $\rightarrow$ MAR
 T_0 1 $\rightarrow$ R
 T_1 Ad (CMDR) $\rightarrow$ CMAR
 T_2 M (MAR) $\rightarrow$ MDR
 T_2 (PC) + 1 $\rightarrow$ PC
 T_3 Ad (CMDR) $\rightarrow$ CMAR
 T_4 MDR $\rightarrow$ IR
 T_4 OP (IR) $\rightarrow$ 微地址形成部件
 T_5 微地址形成部件 $\rightarrow$ CMAR

微指令a

微指令b

微指令c

- 子步骤 2.2 - 补充微程序控制器特有的微操作

- (1) 取指周期:

- 1. 每条指令的取指中都需要有 Ad(CMDR) $\rightarrow$ CMAR
- 2. OP(IR) $\rightarrow$ 微操作形成部件 $\rightarrow$ CMAR

- (2) 执行周期：
 - 1. 执行周期的首地址是由取指周期指出；
 - 2. 执行周期中、每个微程序的最后一条微指令，其都会指明下一条地址的未知到 #0 位置；

分析：阶段三 - 确定微指令格式

- 概念：
 - 根据微操作个数决定采用何种编码方式，以确定微指令的操作控制字段的位数；
 - 根据 CM 中存储的微指令总数，确定微指令的顺序控制字段的位数；
 - 最后按操作控制字段位数和顺序控制字段位数，就可确定微指令字长；

分析：阶段四 - 确定微指令码点

- 概念：根据操作控制字段每一位代表的微操作命令，编写每一条微指令的码点；

微程序设计分类

概念：静态/动态微程序设计

- 静态微程序：
 - 无需改变；
- 动态微程序：
 - 采用 ROM ；
 - 通过改变微指令和微程序来改变机器指令，有利于仿真，采用 EPROM ；

概念：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；

两类控制器比较

分析：硬布线控制器与微程序控制器

- 真题：
 - 题目

【2009 统考真题】相对于微程序控制器，硬布线控制器的特点是 ()。

A. 指令执行速度慢，指令功能的修改和扩展容易

B. 指令执行速度慢，指令功能的修改和扩展难

C. 指令执行速度快，指令功能的修改和扩展容易

D. 指令执行速度快，指令功能的修改和扩展难
 - 解析：

- 微程序控制器采用了“存储程序”的原理，每条机器指令对应一个微程序，因此修改和扩充容易，灵活性好，但每条指令的执行都要访问控制存储器，所以速度慢；
- 硬布线控制器采用专门的逻辑电路实现，其速度主要取决于逻辑电路的延迟，因此速度快，但修改和扩展困难，灵活性差；

• 总结：

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

总结：微程序控制器

