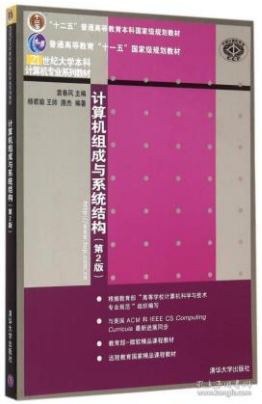
计算机组织结构

16 控制器

吴海军，任桐炜，刘博涵

2021年12月16日

教材对应章节



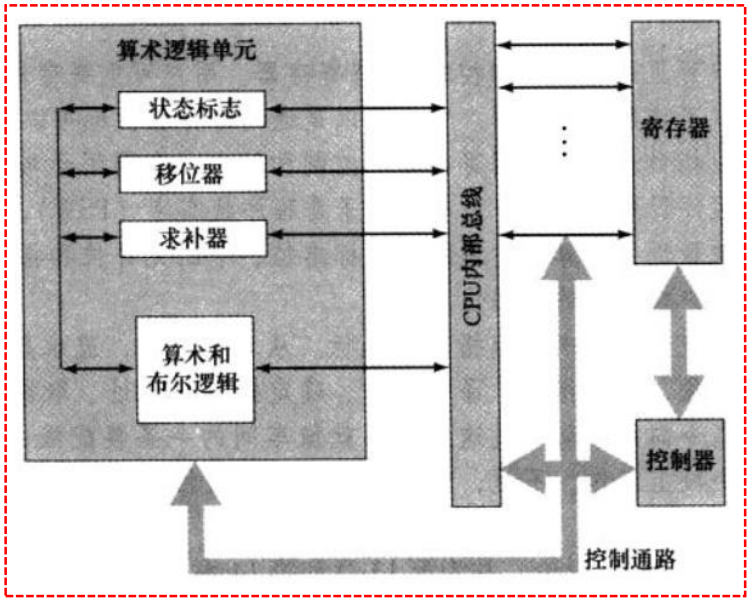
第5章 中央处理器

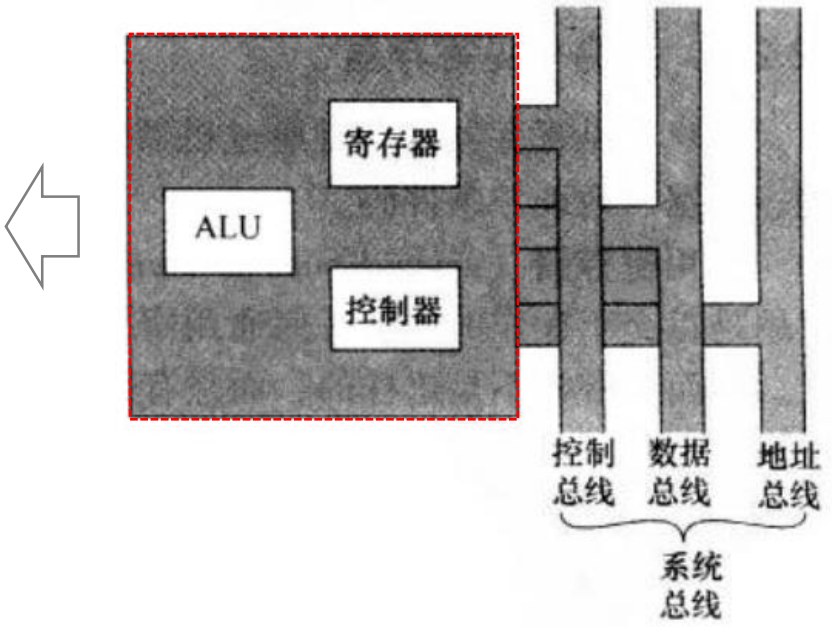
第12章 CPU结构和功能

第15章 控制器操作

第16章 微程序控制

处理器的结构


算术逻辑单元
状态标志
移位器
求补器
算术和
布尔逻辑
CPU内部总线
寄存器
控制器
控制通路


寄存器
ALU
控制器
控制
总线
数据
总线
地址
总线
系统
总线

寄存器分类

- 用户可见寄存器（user-visible register）
  - 允许编程人员通过机器语言或汇编语言访问，通过优化寄存器的使用而减少对主存的访问
- 控制和状态寄存器（control and status register）
  - 由控制器来控制CPU的操作，并由拥有特权的操作系统程序来控制程序的执行
  - 大多数控制和状态寄存器在大多数机器上是用户不可见的
    - 某些在控制或操作系统模式下执行的机器指令是用户可见的
- 两者的区分并不严格



用户可见寄存器

- 通用寄存器（general-purpose register）
  - 可被程序员指派各种用途
- 数据寄存器（data register）
  - 仅可用于保持数据而不能用于操作数地址的计算
- 地址寄存器（address register）
  - 可以是自身有某些通用性，或是专用于某种具体的寻址方式
  - 例如：段指针、变址寄存器、栈指针……
- 条件码寄存器（condition codes register） / 标志（flag）寄存器
  - CPU硬件设置这些条件位作为操作的结果
  - 至少是部分用户可见的

用户可见寄存器：设计出发点

- 使用完全通用的寄存器还是规定各寄存器的用途
- 寄存器数量
  - 太少的寄存器会导致更多的存储器访问
  - 太多的寄存器又不能显著地减少存储器访问
- 寄存器长度
  - 应能保存大多数数据类型的值
  - 某些机器允许两个相邻的寄存器作为一个寄存器来保持两倍长度的值




用户可见寄存器：保存和恢复

- 子程序调用会导致自动保存所有用户可见的寄存器，并在返回时自动取回
  - 这些保存和恢复是作为调用和返回指令执行功能的一部分，由CPU完成
  - 这允许各个子程序独立地使用用户可见寄存器
- 子程序调用之外保存用户可见寄存器的相关内容是程序员的责任，需要在程序中为此编写专门的指令

控制和状态寄存器

- 程序计数器（Program counter，PC）
  - 存有待取指令的地址
  - 通常在每次取指令之后，PC的内容即被CPU更改，转移或跳步指令也会修改PC的内容，因此总指向将被执行的下一条指令
- 指令寄存器（Instruction register，IR）
  - 存有最近取来的指令，在其中分析操作码和操作数
- 存储器地址寄存器（Memory address register，MAR）
  - 直接与地址总线相连，存有存储器位置的地址
- 存储器缓冲寄存器（Memory buffer register，MBR）
  - 直接与数据总线相连，存有将被写入存储器的数据字或从存储器读出的字，用户可见寄存器再与MBR交换数据
  - ALU可对MBR和用户可见寄存器直接存取

控制和状态寄存器（续）

- 程序状态字（Program status word，PSW）
  - 一个或一组包含状态信息的寄存器，包含条件码加上其他状态信息
  - 包含的字段或标志
    - 符号（Sign）：容纳算术运算结果的符号位
    - 零（Zero）：当结果是0时被置位
    - 进位（Carry）：若操作导致最高位有向上的进位（加法）或借位（减法）时被置位，用于多字算数运算
    - 等于（Equal）：若逻辑比较的结果相等，则置位
    - 溢出（Overflow）：用于表示算术溢出
    - 中断允许/禁止：用于允许或禁止中断
    - 监管（Supervisor）：指出CPU是执行在监管模式中还是在用户模式中

控制和状态寄存器（续）

- 一个指向含有附加状态信息的存储器块的指针寄存器
- 在使用向量式中断的机器中，可能提供有一个中断向量寄存器
- 若栈用于实现某些功能，需要有一个系统栈指针
- 对于虚拟存储器系统，会有一个页表指针寄存器

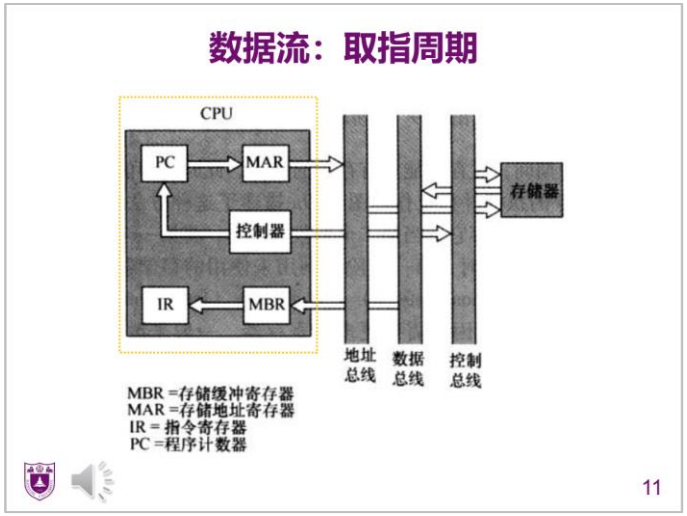
控制和状态寄存器：设计出发点

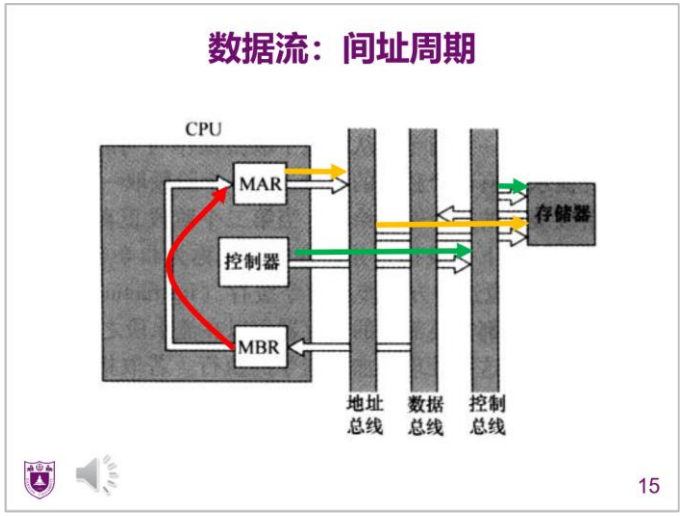
- 对操作系统的支持
  - 某些类型的控制信息是专门为操作系统使用的
  - 若CPU设计者对将要使用的操作系统有基本的了解，则寄存器的组织可能在一定程度上为该操作系统定制
- 控制信息在寄存器和存储器之间的分配
  - 一种普遍的做法是将存储器最前面（最低地址）的几百或几千个字用于控制目的
  - 在成本和速度之间进行权衡



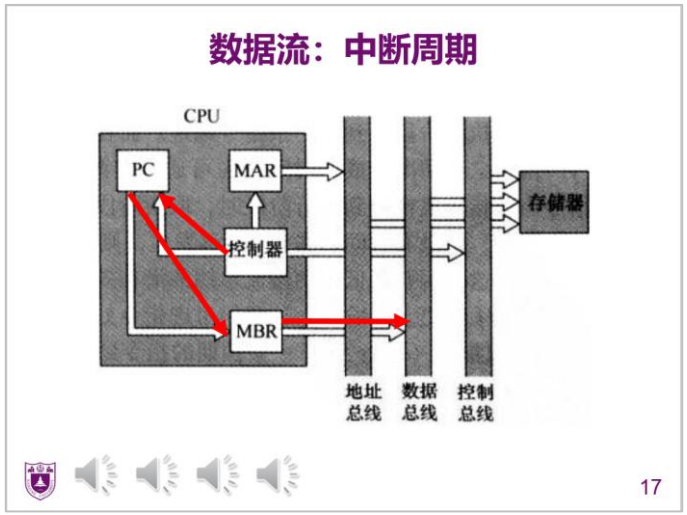
回顾：数据流

回顾：数据流



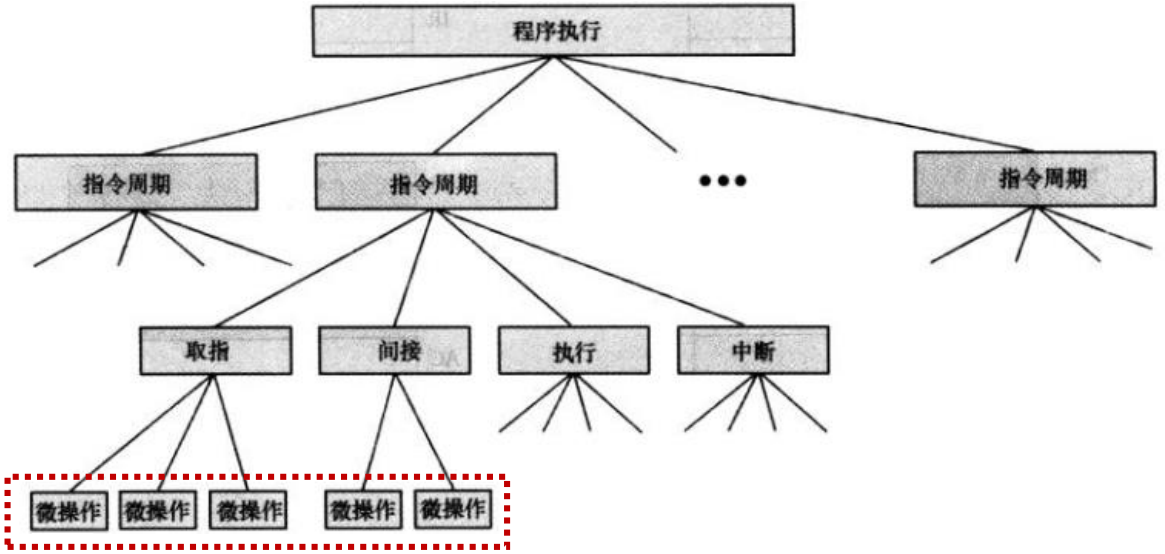


如何控制计算机的各种组件来完成这些功能?



微操作（micro-operation）

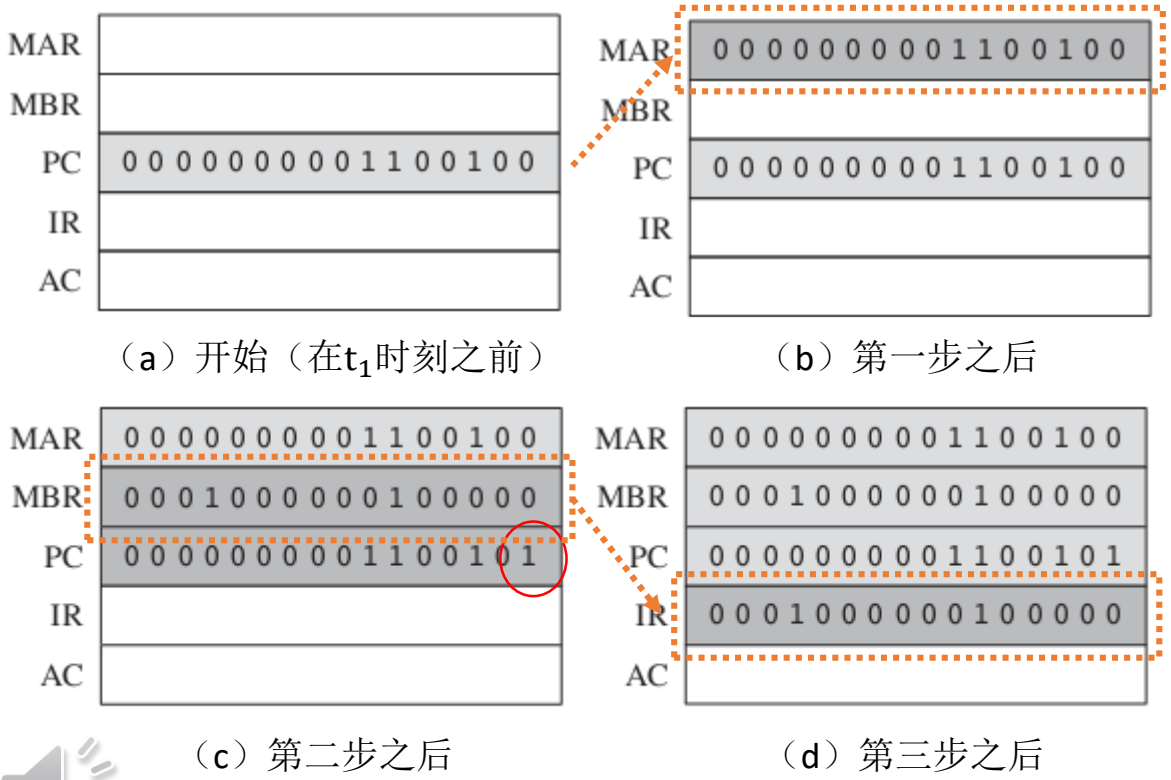
- 执行程序时，计算机操作是由一系列指令周期组成，每个周期执行一条机器指令
- 每个指令周期又可以看作是由几个更小的子周期组成
  - 包括：取指、间址、执行、中断
- 每个子周期由一系列涉及CPU寄存器操作的更小步骤组成，这些步骤称为**微操作**



取指周期

- 出现在每个指令周期的开始，将指令从存储器中取出

t_1: MAR ← (PC)
t_2: MBR ← Memory
PC ← (PC) + *I*
t_3: IR ← (MBR)



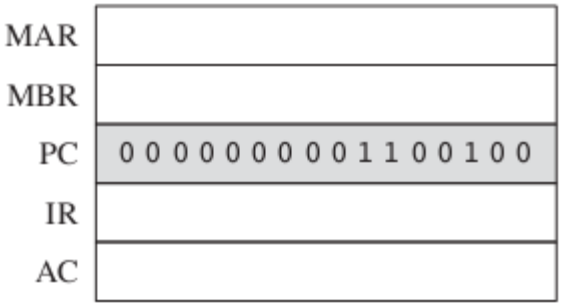
（a）开始（在t_1时刻之前）　（b）第一步之后

（c）第二步之后　（d）第三步之后

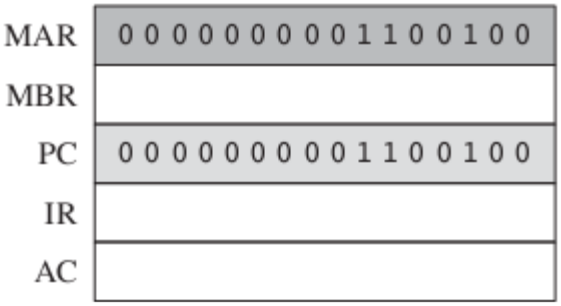
取指周期（续）

- 出现在每个指令周期的开始，将指令从存储器中取出

t_1: MAR ← (PC)
t_2: MBR ← Memory
PC ← (PC) + *I*
t_3: IR ← (MBR)

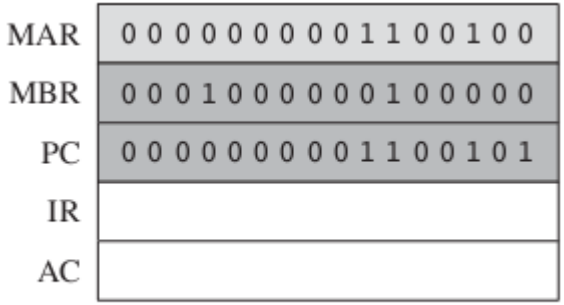


（a）开始（在t_1时刻之前）

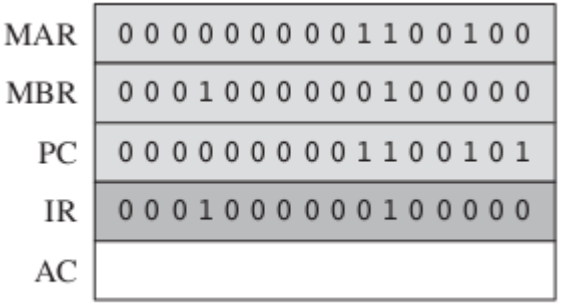


（b）第一步之后

t_1: MAR ← (PC)
t_2: MBR ← Memory
t_3: PC ← (PC) + *I*
IR ← (MBR)



（c）第二步之后



（d）第三步之后

微操作分组的原则

- 事件的流动顺序必须是恰当的
  - 例：MAR ← (PC) 必须先于MBR ← 内存，因为内存读操作要使用MAR中的地址
- 必须避免冲突
  - 例：MBR ← 内存 和IR ← MBR 这两个微操作不应出现在同一时间单位里
- 满足上述条件下，所用的时间单位尽可能少




间址周期

- 如果指令采用间接寻址，则在指令执行前有一个间址周期

t_1: MAR ← (IR(Address))
t_2: MBR ← Memory
t_3: IR(Address) ← (MBR(Address))

- 完成间址周期后，IR的状态与不使用间接寻址方式的状态是相同的，已经为执行周期准备就绪

执行周期

- 对于不同的操作码，会出现不同的微操作序列
- 例：加法指令

```
ADD R1, X     t1: MAR ← (IR(address))
              t2: MBR ← Memory
              t3: R1 ← (R1) + (MBR)
```

- 例：“转移并保存地址”指令

```
BSA X         t1: MAR ← (IR(address))
                  MBR ← (PC)
              t2: PC ← (IR(address))
                  Memory ← (MBR)
              t3: PC ← (PC) + I
```





中断周期

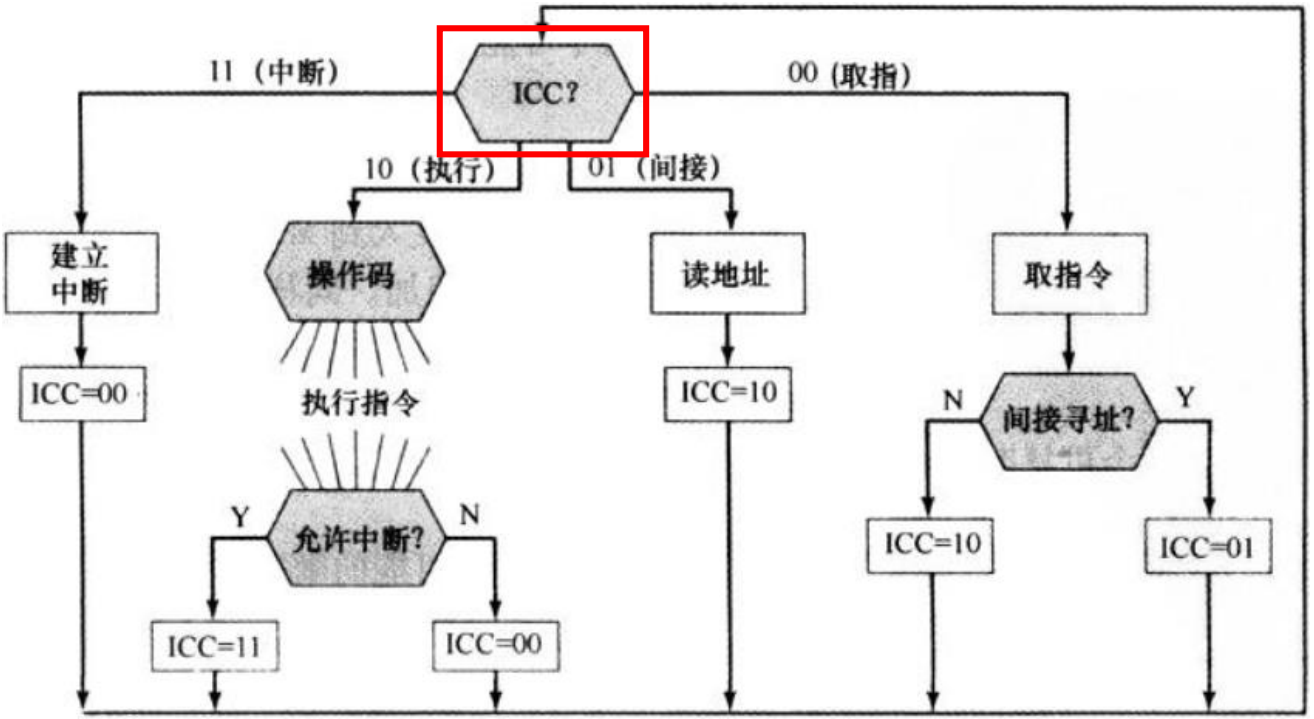
- 在完成执行周期时，要确定是否有允许的中断产生
- 如果有，则出现一个中断周期

t_1: MBR ← (PC)
~~t_2~~: MAR ← Save_Address
t2: PC ← Routine_Address
~~t_3~~: Memory ← (MBR)

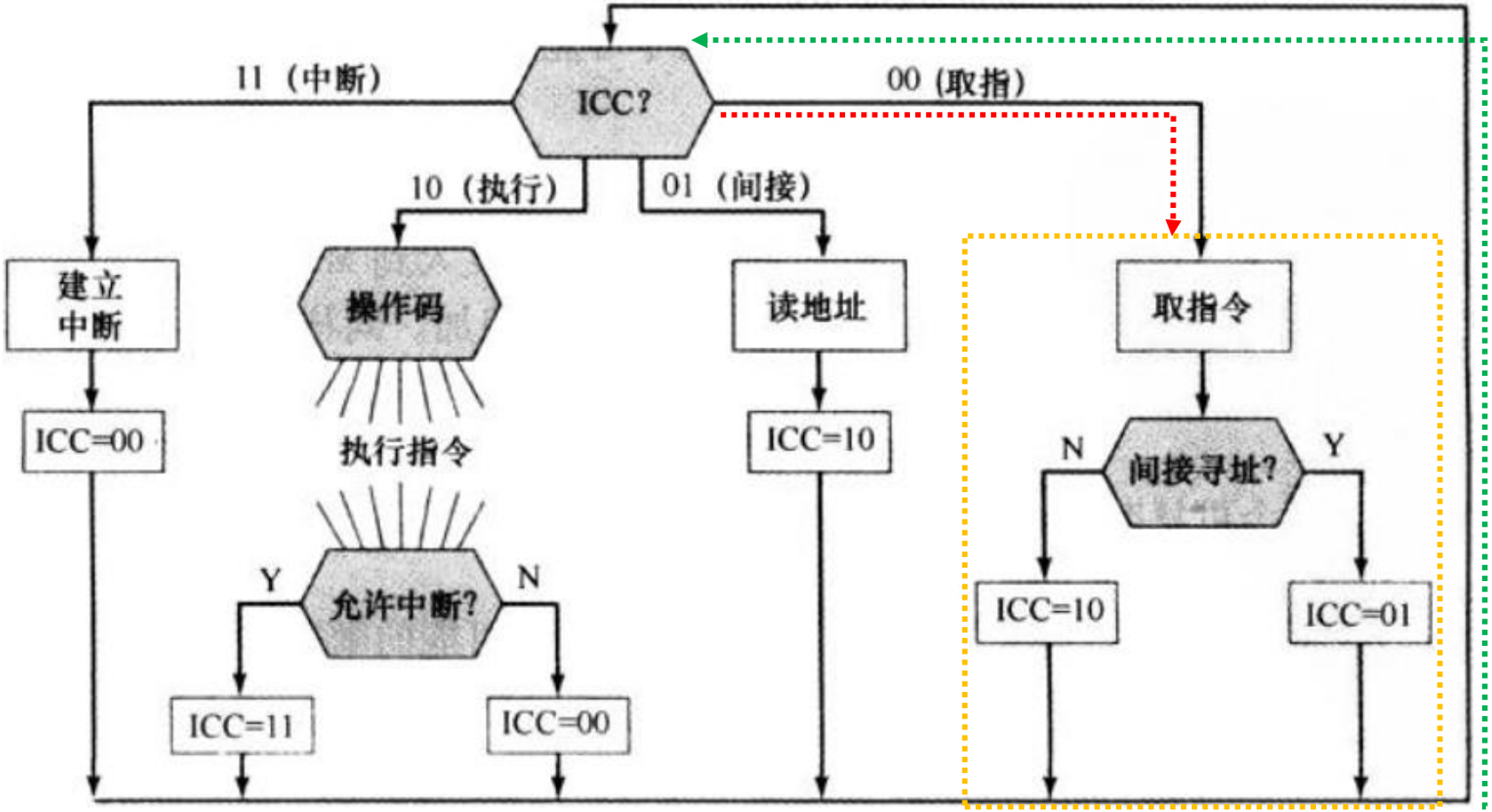


指令周期代码（Instruction Cycle Code，ICC）

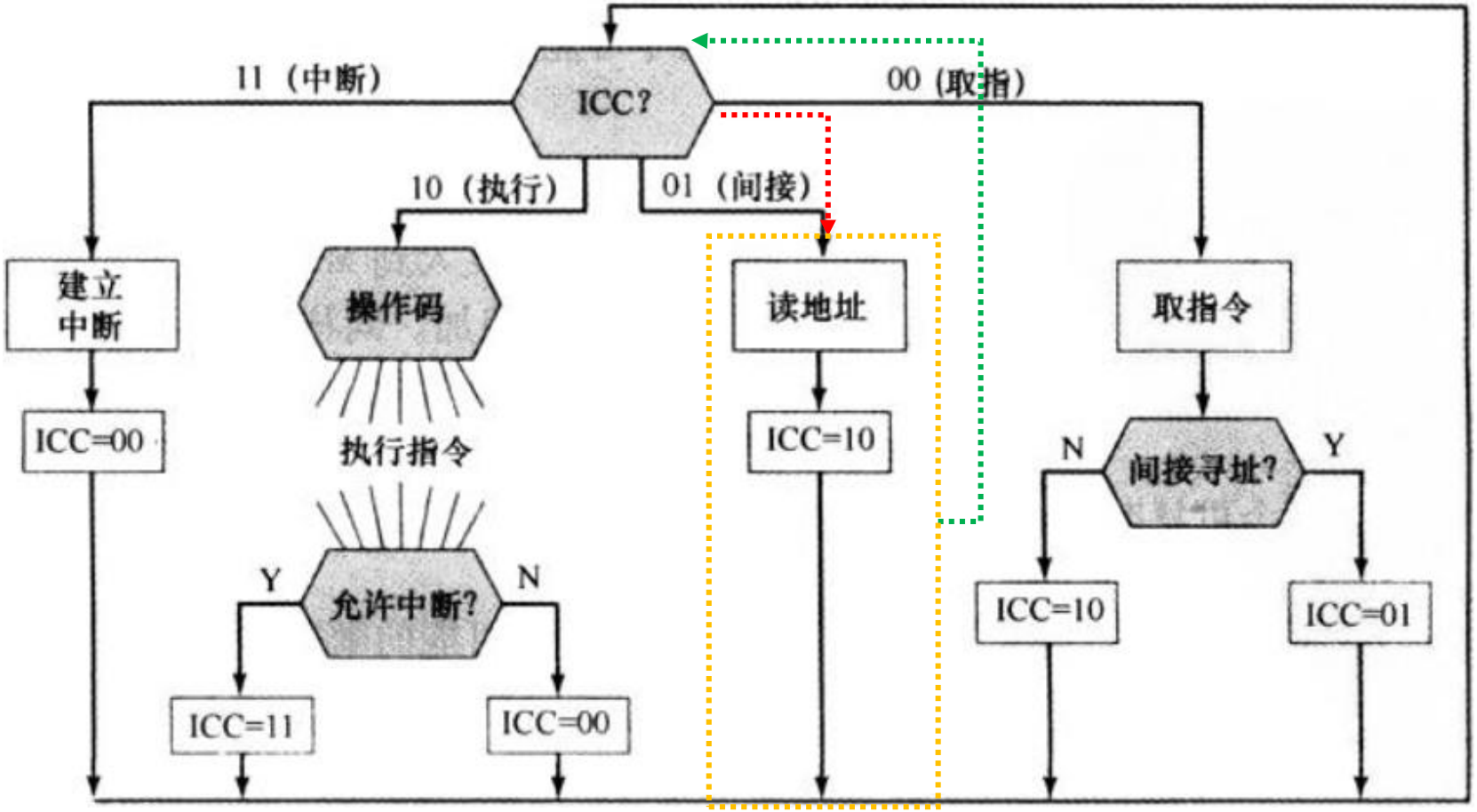
- 取指、间址、中断周期各有一个微操作序列，执行周期则对于每个操作码有一个微操作序列
- 指令周期代码：假设一个2位的ICC寄存器，明确CPU处于指令周期哪个阶段
  - 00：取指
  - 01：间址
  - 10：执行
  - 11：中断



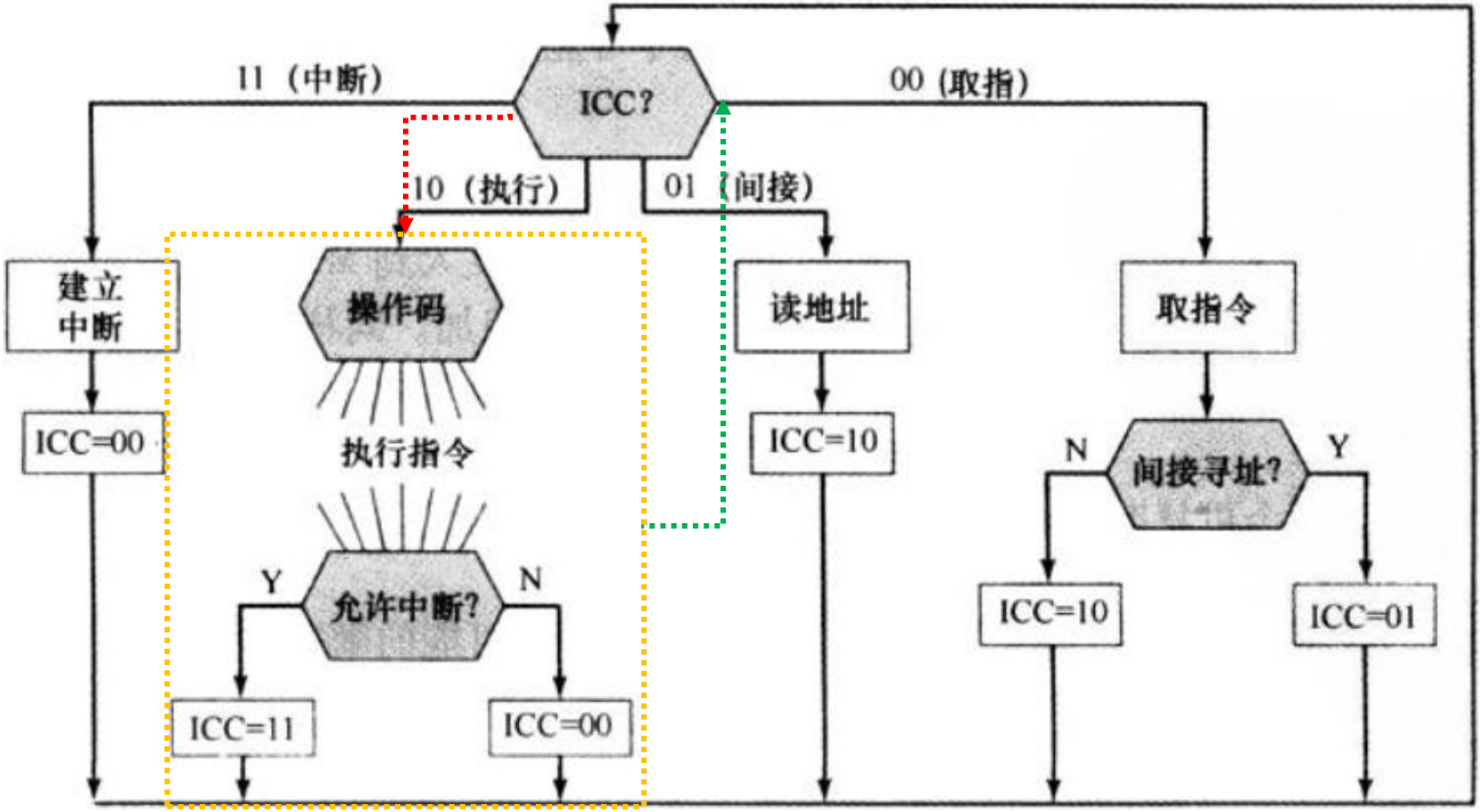
指令周期代码：取指周期


ICC?
11（中断）
00（取指）
10（执行）
01（间接）
建立中断
ICC=00
操作码
执行指令
允许中断?
Y
N
ICC=11
ICC=00
读地址
ICC=10
取指令
间接寻址?
N
Y
ICC=10
ICC=01

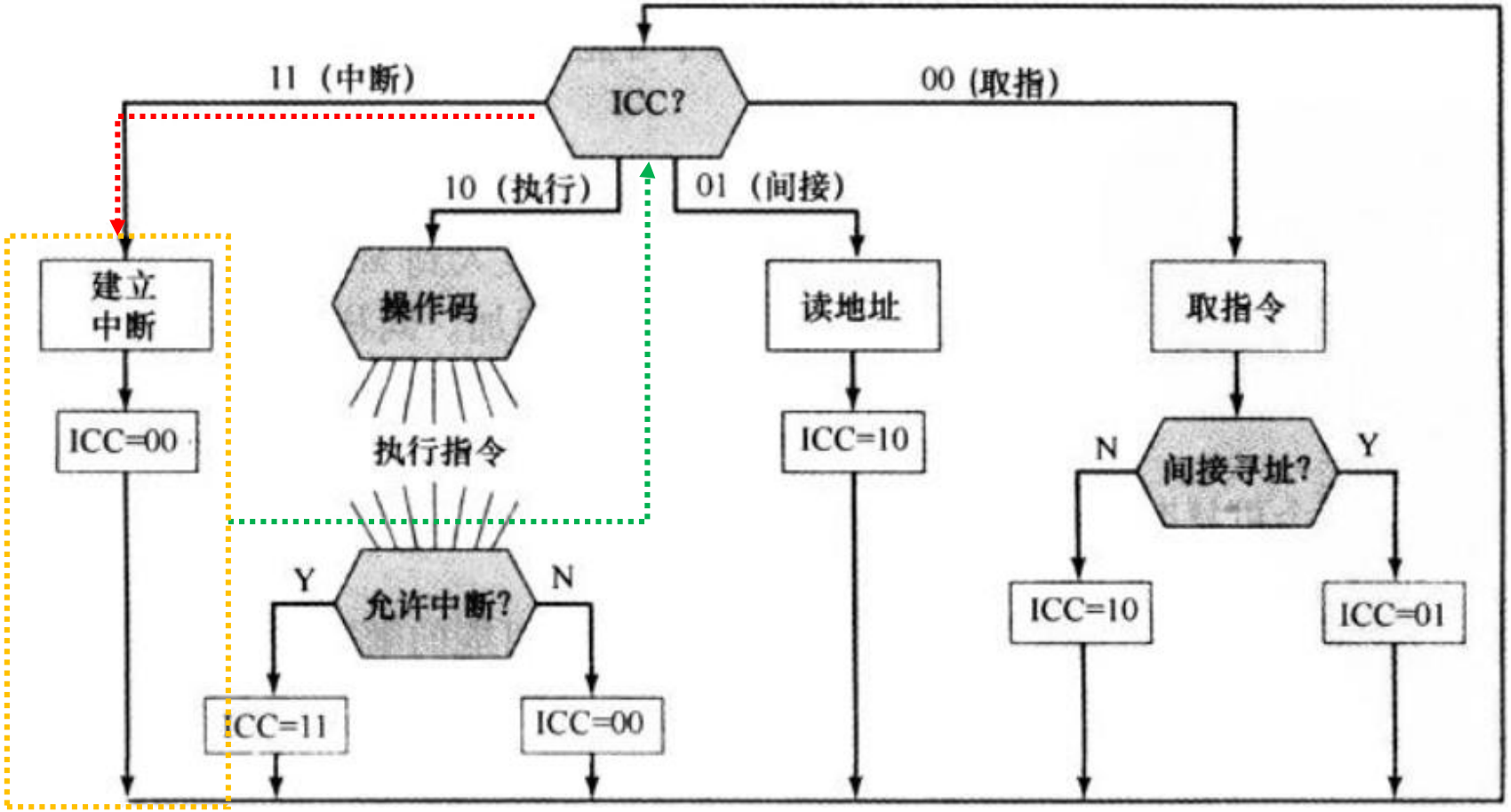
指令周期代码：间址周期


ICC?
11（中断）
00（取指）
10（执行）
01（间接）
建立中断
ICC=00
操作码
执行指令
允许中断?
Y
N
ICC=11
ICC=00
读地址
ICC=10
取指令
间接寻址?
N
Y
ICC=10
ICC=01

指令周期代码：执行周期


ICC?
11（中断）
10（执行）
01（间接）
00（取指）
建立中断
ICC=00
操作码
执行指令
允许中断?
Y
N
ICC=11
ICC=00
读地址
ICC=10
取指令
间接寻址?
N
Y
ICC=10
ICC=01

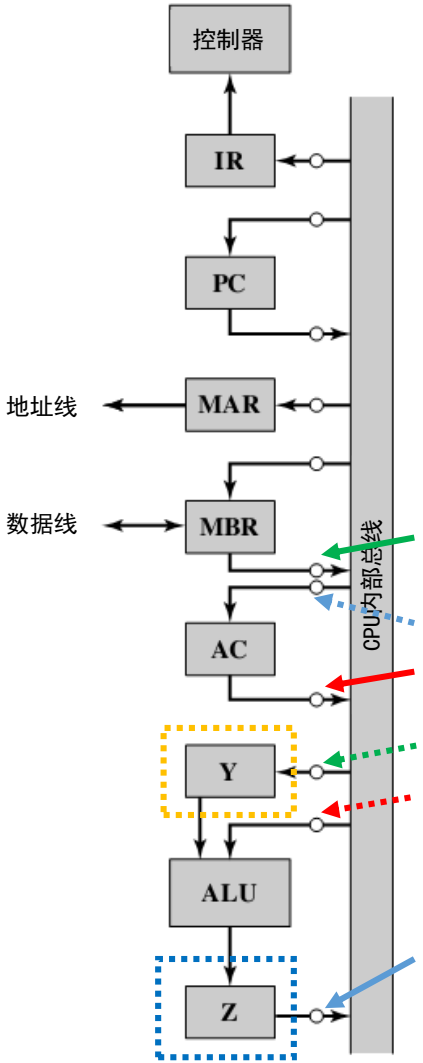
指令周期代码：中断周期


11（中断）
ICC?
00（取指）
10（执行）
01（间接）
建立中断
操作码
读地址
取指令
ICC=00
执行指令
ICC=10
N
间接寻址?
Y
Y
允许中断?
N
ICC=10
ICC=01
ICC=11
ICC=00

CPU内部总线

- ALU和寄存器都连接到CPU内部总线上
- 为了数据在该内部总线和各寄存器之间传递，内部总线和寄存器之间有门和控制信号
- 控制线控制着数据和系统总线（外部）的交换以及ALU的操作

t_1: MAR ← (IR(address))
t_2: MBR ← Memory
t_3: Y ← (MBR)
t_4: Z ← (AC) + (Y)
t_5: AC ← (Z)



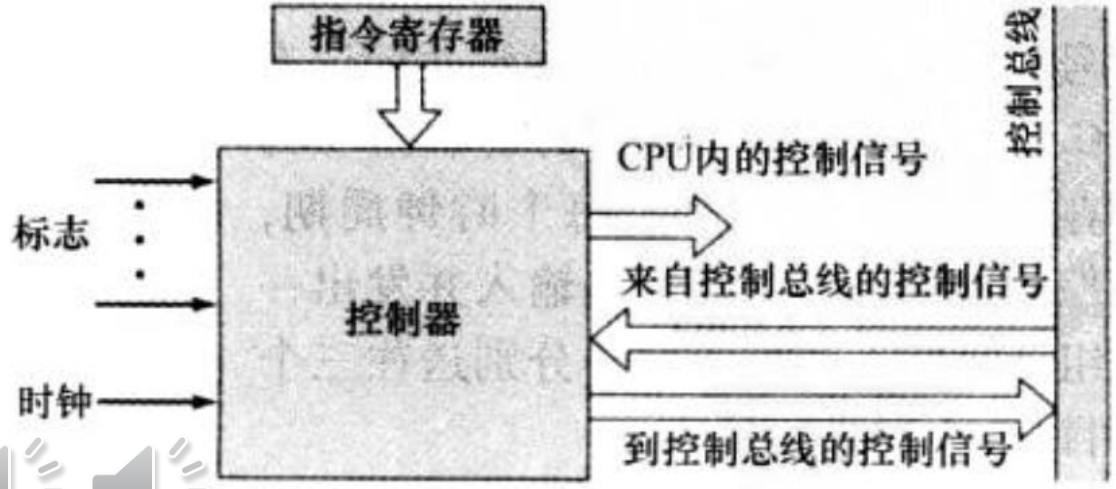
控制CPU的功能需求

- CPU的基本元素
  - ALU，寄存器组，内部数据通路，控制器，外部数据通路
- CPU需要完成的微操作
  - 在寄存器之间传送数据
  - 将数据由寄存器传送到外部接口（如系统总线）
  - 将数据由外部接口传送到寄存器
  - 将寄存器作为输入和输出，完成算术和逻辑运算
- 控制器的两个基本任务
  - 定序（sequencing）：根据正被执行的程序，控制器使CPU以正确的顺序通过一系列微操作
  - 执行（execution）：控制器使每个微操作得以完成



控制器的输入

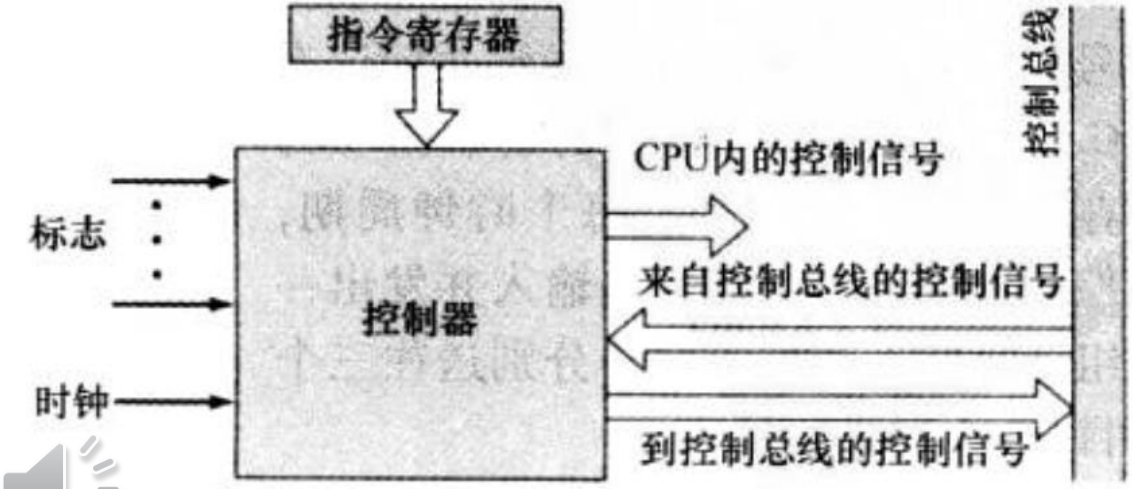
- 指令寄存器：当前指令的寻址方式和操作码
- 标志：确定CPU的状态和前一个ALU操作的结果
- 时钟：控制器要在每个时钟脉冲完成一个或一组同时的微操作
- 来自控制总线的控制信号：向控制器提供控制信号
  - 例：中断请求



控制器的输出

- CPU内的控制信号：
  - 用于寄存器之间传送数据
  - 用于启动特定的ALU功能
- 到控制总线的控制信号：
  - 到存储器的控制信号
  - 到I/O模块的控制信号

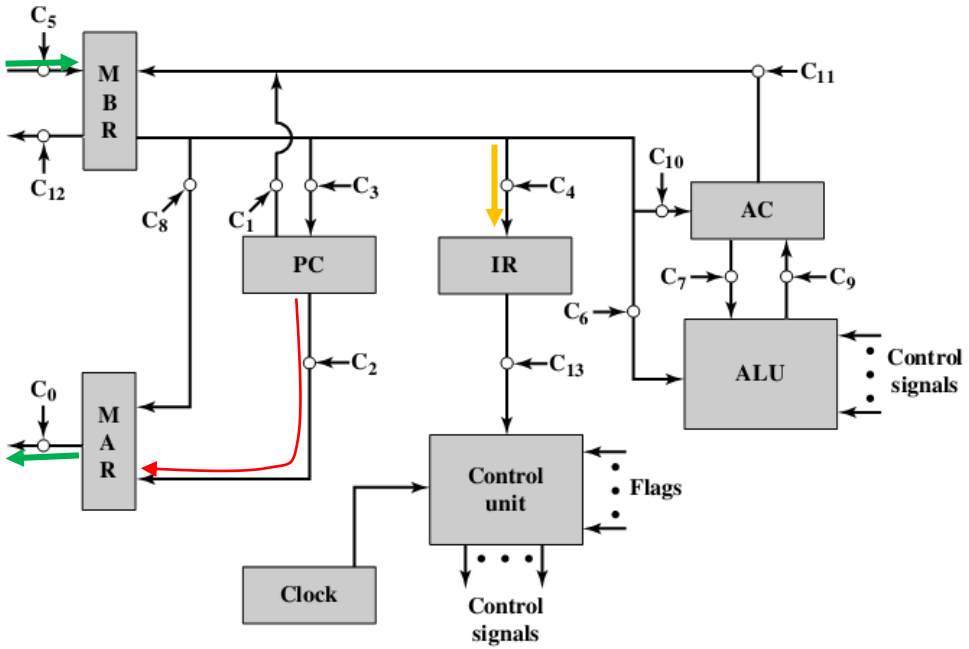
所有的控制信号最终作为二进制输入量
直接输入到各个逻辑门上



控制信号示例：取指周期

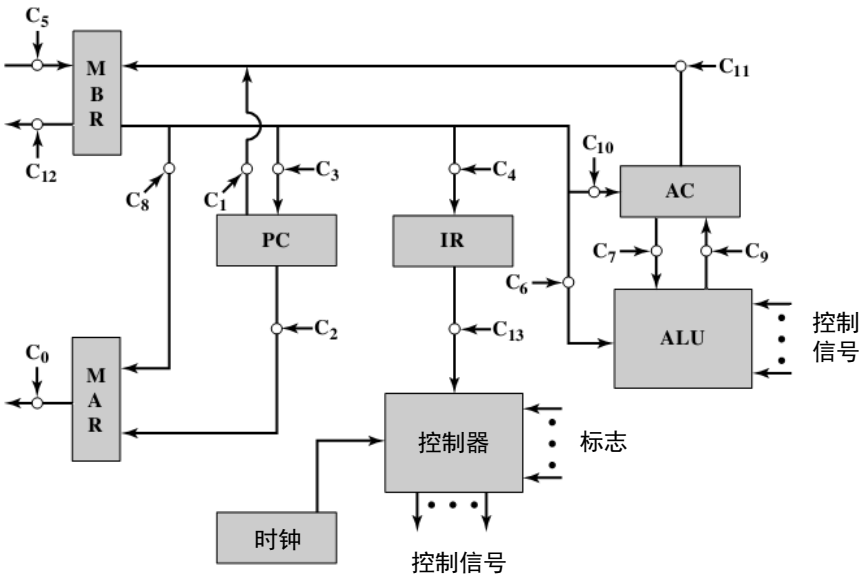
- 传送PC的内容到MAR
  - 打开C_2：PC传到MAR
- 由存储器读一条指令装入MBR，并且递增PC
  - 打开C_0：MAR的内容送到地址总线上
  - 存储器读控制信号C_R送到控制总线上
  - 打开C_5：数据总线上的内容存入MBR
  - 控制信号对PC内容加1（指令长度）并把结果存回PC
- 传送MBR的内容到IR
  - 打开C_4：MBR的内容送到IR

	微操作	有效的控制信号
取指	t_1：MAR←（PC）	C_2
	t_2：MBR←Memory PC←（PC）+1	C_5，C_R
	t_3：IR←（MBR）	C_4



控制信号示例：更多的子周期

	微操作	有效的控制信号
取指	t_1：MAR←（PC）	C_2
	t_2：MBR←Memory PC←（PC）+1	C_5，C_R
	t_3：IR←（MBR）	C_4
间接	t_1：MAR←（IR（地址））	C_8
	t_2：MBR←Memory	C_5，C_R
	t_3：IR（地址）←（MBR（地址））	C_4
中断	t_1：MBR←（PC）	C_1
	t_2：MAR←保存_地址 PC←子程序_地址	
	t_3：Memory←（MBR）	C_{12}，C_W



控制器的最小特性

- 它只需要知道将被执行的指令和算术、逻辑运算结果的性质（如正负、溢出等），而不需要知道正被处理的数据或得到的实际结果具体是什么
- 它只是以少量的送到CPU内的和送到系统总线上的控制信号来实现控制



控制器实现

- 硬布线实现（hardwired implementation）
  - 控制器是一个组合电路，把输入逻辑信号转换为一组输出逻辑信号，即控制信号
- 微程序实现（microprogrammed implementation）
  - 控制逻辑是微程序指定的，控制器是一个相对简单的逻辑电路，通过执行每条微指令来产生控制信号

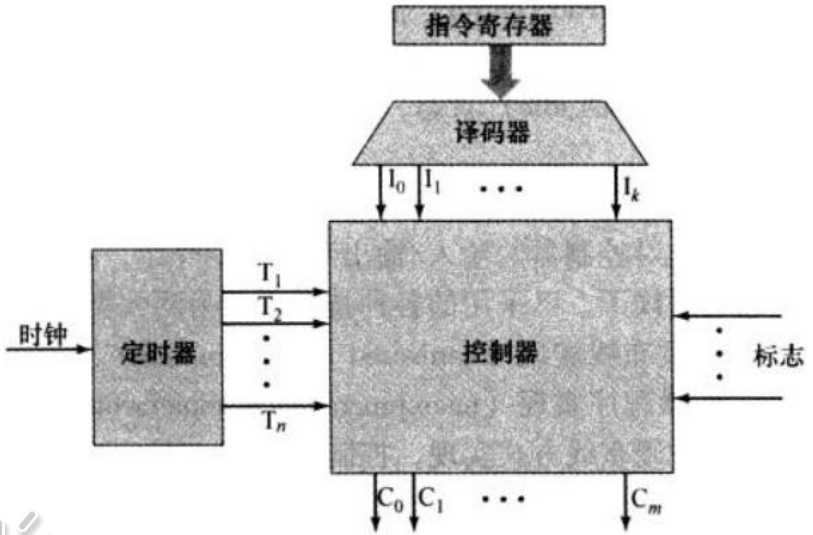
硬布线实现：控制器输入

- 标志和控制总线信号
  - 每位都有特定的意义
- 指令寄存器
  - 通过译码，使每一操作码有一个唯一的逻辑输入
  - 译码器有 n 个输入和 2^n个输出
    - 控制器要考虑变长的操作码，译码器会更复杂些

I1	I2	I3	I4	O1	O2	O3	O4	O5	O6	O7	O8	O9	O10	O11	O12	O13	O14	O15	O16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

硬布线实现：控制器输入（续）

- 时钟
  - 在一个指令周期内，控制器要在不同时间单位发送不同的控制信号
  - 使用一个定时器作为控制器的输入，并且控制器在指令周期结束时必须通知定时器以使其重新开始计数



硬布线实现：控制器逻辑

- 为每个输出的控制信号设计一个关于控制器输入的布尔表达式
- 定义两个新的控制信号P和Q
  - PQ = 00：取指周期，PQ = 01：间接周期，PQ = 10：执行周期，PQ = 11：中断周期
  - 示例：C5：使外部数据总线上的数据读入MBR
    - C5在取指和间接周期的第二个时间单位有效

$$C_5 = \bar{P} \cdot \bar{Q} \cdot T_2 + \bar{P} \cdot Q \cdot T_2$$

    - C5在执行周期也需要

$$C_5 = \bar{P} \cdot \bar{Q} \cdot T_2 + \bar{P} \cdot Q \cdot T_2 + P \cdot \bar{Q} \cdot (LDA + ADD + AND) \cdot T_2$$

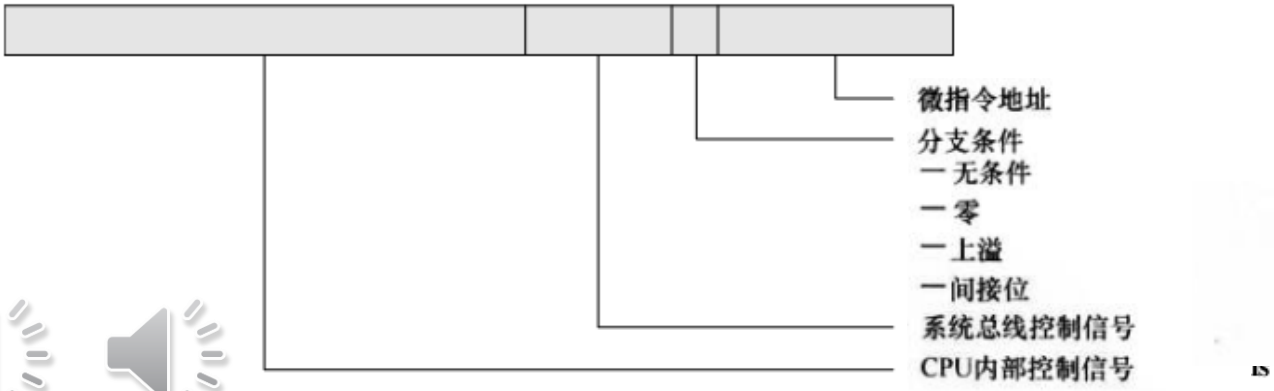
微程序实现：基本概念和思路

- 微程序（固件）介于硬件与软件之间
  - 设计固件比设计硬件容易，但写固件程序比软件程序困难
  - 微指令序列
- 微指令：每行描述一个时间内出现的一组微操作
- 基本思路
  - 对于每个微操作，控制器的任务是产生一组控制信号，即控制器发出的每根控制线或开或关（每根控制线由一个二进制数字表示）
  - 构造一个控制字，每位代表一根控制线，这样每个微操作能用控制字中的不同的0和1的样式来表示
  - 将这些控制字串在一起，可以表示控制器需要完成的微操作序列



微程序实现：基本概念和思路（续）

- 由于微操作序列不是固定的，把控制字放入一个存储器单元中，每个字有自己唯一的地址
  - 添加少数几位用于指示条件的真假
    - 若条件位指示的条件为假，则顺序执行下一条指令
    - 若条件位指示的条件为真，则地址字段指向的微指令是将被执行的下一条微指令
  - 给每个控制字添加一个地址字段，以指示某种条件为真时，将要执行的下一控制字的位置



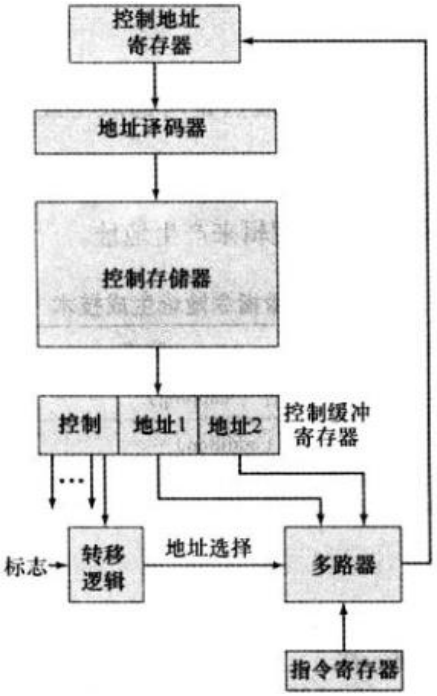
微程序实现：微程序执行


转移至间接或执行
转移至执行
转移至取指
转移至操作码例程
转移至取指或中断
转移至取指或中断
转移至取指或中断
取指周期例程
间接周期例程
中断周期例程
执行周期例程
AND例程
ADD例程
IOF例程

微程序控制器：任务

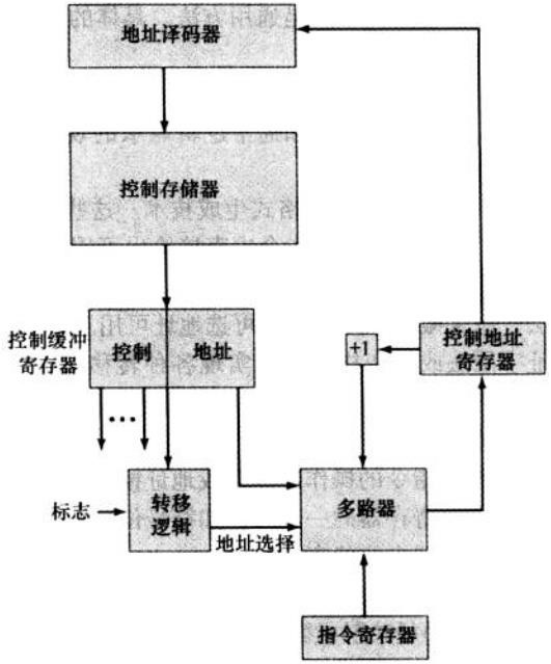
- 微指令定序（microinstruction sequencing）
  - 根据当前的微指令、条件标志和指令寄存器的内容，产生下一微指令的控制存储器地址
  - 设计考虑
    - 微指令的大小：减小微指令的大小就能节省控制存储器的成本
    - 地址生成时间：尽可能快地执行微指令
- 微指令执行（microinstruction execution）
  - 产生控制信号：发往CPU内部，送往外部控制总线或其他外部接口

微程序控制器：定序



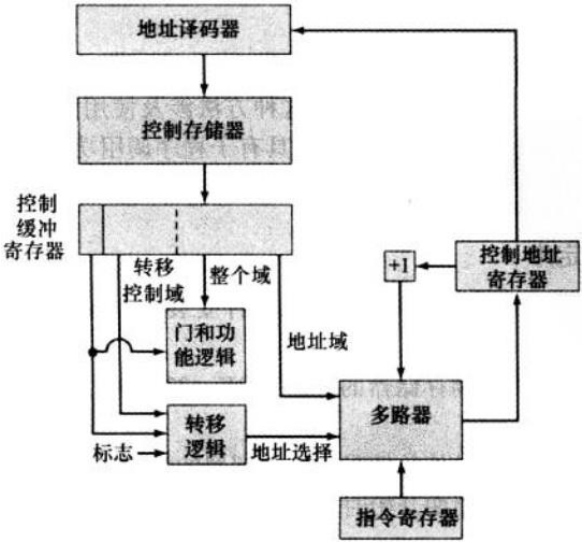
双地址字段

在每条微指令中提供两个地址字段，选择并发送其中某一个地址或操作码到控制地址寄存器



单地址字段

下一个地址得选择可以是地址字段或下一个顺序地址

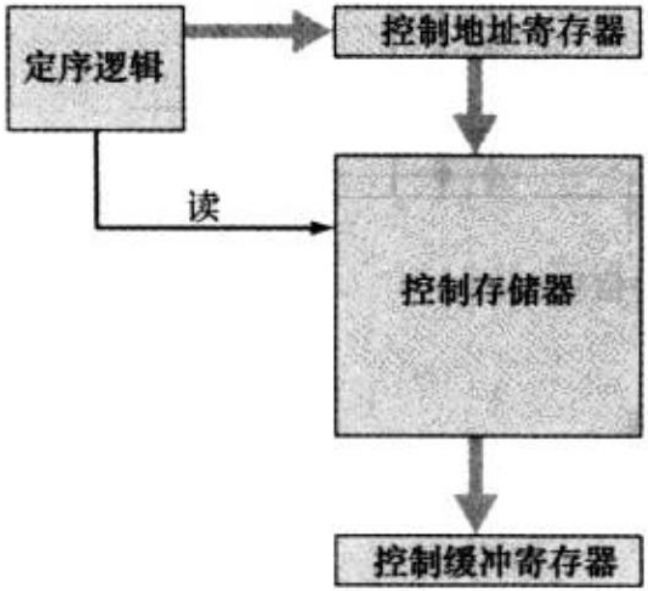


可变格式

提供两种完全不同的指令格式，一位字段用于指定哪种格式被使用

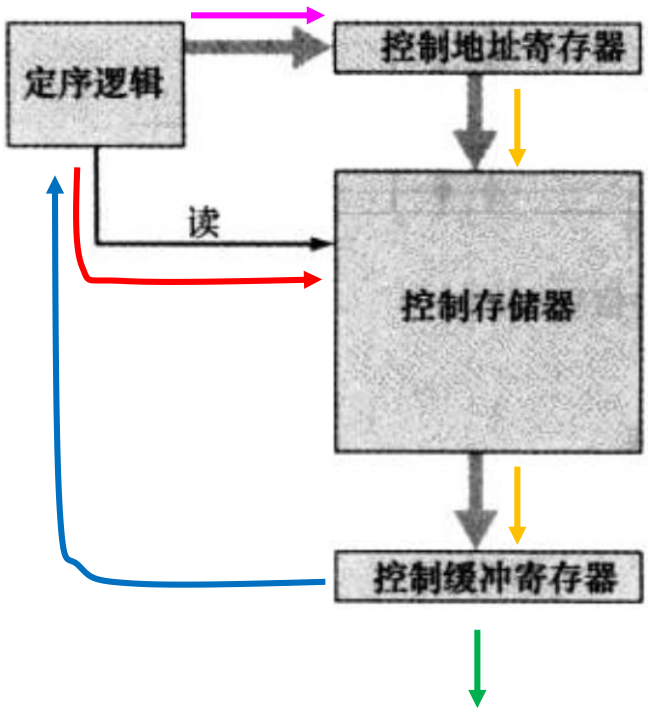
微程序控制器：构成

- 定序逻辑：向控制地址寄存器装入地址，并发出读命令
- 控制地址寄存器：含有下面即将被读取的微指令地址
- 控制存储器：存有一组微指令
- 控制缓冲寄存器：存放被读出的微指令



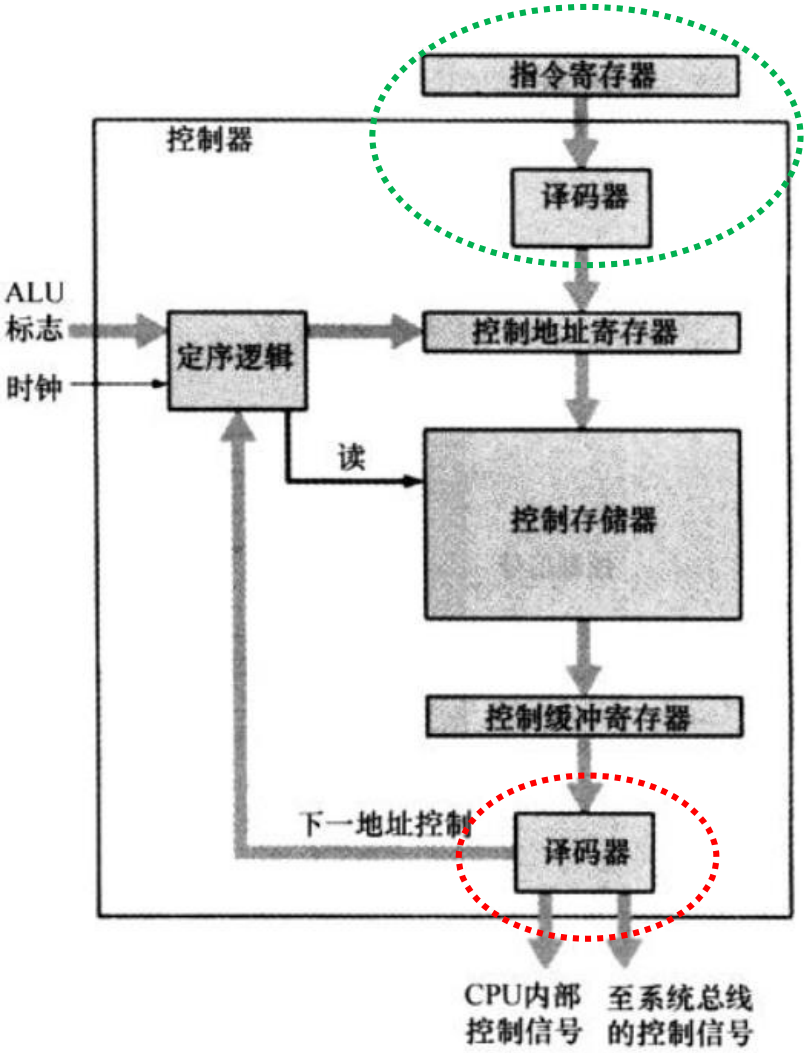
微程序控制器：工作流程

- 为执行一条指令，定序逻辑发出一个读命令给控制存储器
- 当一条微指令由控制存储器读出后，即被传送到控制缓冲寄存器
- 控制缓冲寄存器的内容生成控制信号，并为定序逻辑提供下一条地址信息
  - 控制缓冲寄存器的左半部分与控制器发出的控制线相连
  - 由控制存储器读一条微指令等同于执行这条微指令
- 定序逻辑根据这个地址信息和ALU标志，将新的地址装入到控制地址寄存器中



微程序控制器：工作流程（续）

- 生成新地址的三个选择
  - 取顺序下一条微指令：加1到控制地址寄存器
  - 基于跳转微指令转移到一个新的例程：将控制缓冲寄存器的地址字段装入控制地址寄存器
  - 转移到一个机器指令例程：根据IR中的操作码向控制地址寄存器装入机器指令例程的第一条微指令



微程序实现：优点与缺点

- 优点
  - 简化了控制器的设计任务
  - 实现起来既成本较低，也能减少出错机会
- 缺点
  - 要比采用相同或相近半导体工艺的硬布线控制器慢一些

总结

- 寄存器
  - 用户可见寄存器：常见类型，设计出发点，保存和恢复
  - 控制和状态寄存器：常见类型，设计出发点
- 微操作
- 控制器：输入和输出，控制信号
- 控制器实现
  - 硬布线实现：控制器输入，控制器逻辑
  - 微程序实现
    - 基本概念和思路：微程序，微指令
    - 微程序控制器：任务，构成，工作流程

谢谢

rentw@nju.edu.cn

南京大學
NANJING UNIVERSITY