计算机组织与体系结构

Computer Architectures

陆俊林

北京大学 Peking University

2024年春季学期

第七讲 控制器的原理和分类

本讲要点

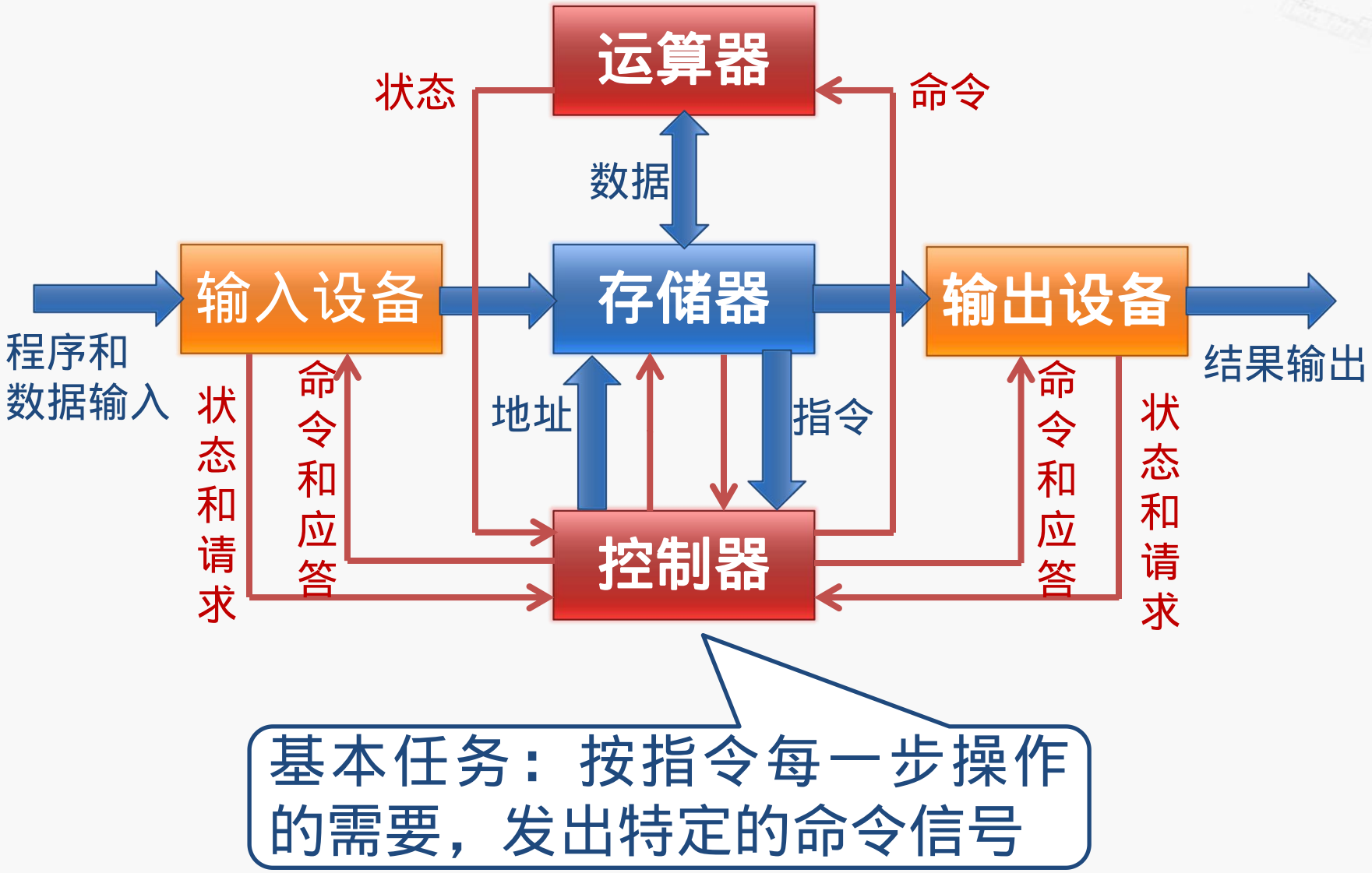
首先回顾冯·诺依曼结构，讲解其中控制器的基本原理，其次简要介绍两种控制器的基本特点，然后分别详细分析硬布线控制器和微程序控制器的原理和实现。

阅读参考书《微型计算机……》：第2章

主要内容

通过学习本课程
了解计算机的发展历程，理解计算机的组成原理，掌握计算机的设计方法

冯·诺依曼计算机结构


运算器
状态
命令
数据
输入设备
存储器
输出设备
程序和
数据输入
结果输出
状态和请求
命令和应答
地址
指令
命令和应答
状态和请求
控制器
基本任务：按指令每一步操作的需要，发出特定的命令信号

某假想计算机的指令系统

运算类指令

- ADD　R, M
  - 功能：将R的内容与M中的内容相加后存入R
- LOAD　R, M
  - 功能：将M中的内容装入R
- STORE　M, R
  - 功能：将R的内容存入M中
- JMP　L
  - 功能：无条件转向L处

传送类指令

转移类指令

*注：M和L为存储器地址，R为寄存器编号

程序示例

- 程序功能包含两个步骤：
  1. 将M1的内容与M2的内容相加后存入M3，可表示为：(M1)+(M2)→(M3)
  2. 将程序转向L处的指令继续执行
- 程序代码

汇编语言程序	机器语言程序	程序的功能
LOAD R1, M1	00001011 00000101	将M1的内容送入R1
ADD R1, M2	00011011 00000110	将R1的内容加上M2的内容再送回R1
STORE M3, R1	00101011 00000111	将R1的内容送入M3中
JMP L	00110000	转向L处继续执行指令

程序示例说明

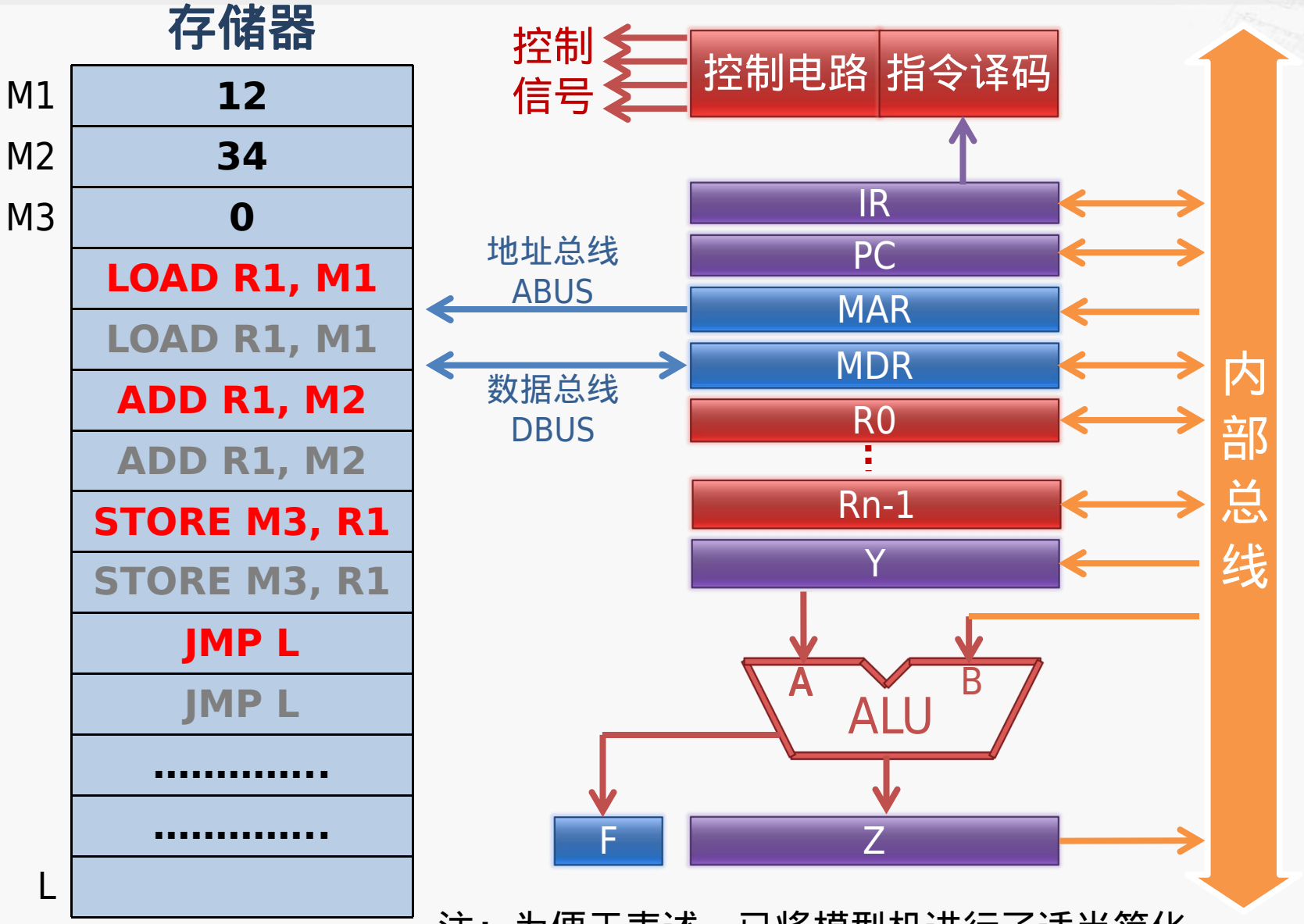
汇编语言程序	机器语言程序	程序的功能
LOAD R1, M1	00001011 00000101	将M1的内容送入R1
ADD R1, M2	00011011 00000110	将R1的内容加上M2的内容再送回R1
STORE M3, R1	00101011 00000111	将R1的内容送入M3中
JMP L	00110000 00010010	转向L处继续执行指令

- 四条指令LOAD、ADD、STORE、JMP的操作码分别为0000、0001、0010、0011
- 每条机器指令的第二个字节都是存储单元地址，第一个字节左边4位是操作码，右边4位是寄存器号或0000
- 假设存储单元的地址M1、M2、M3分别为5、6、7，其内容分别为12、34、0；地址L为18；R1寄存器的编号为1011_{two}
- 上述机器语言程序可被计算机识别，但不能被执行，还需要为其分配存储器地址

机器语言在存储器中

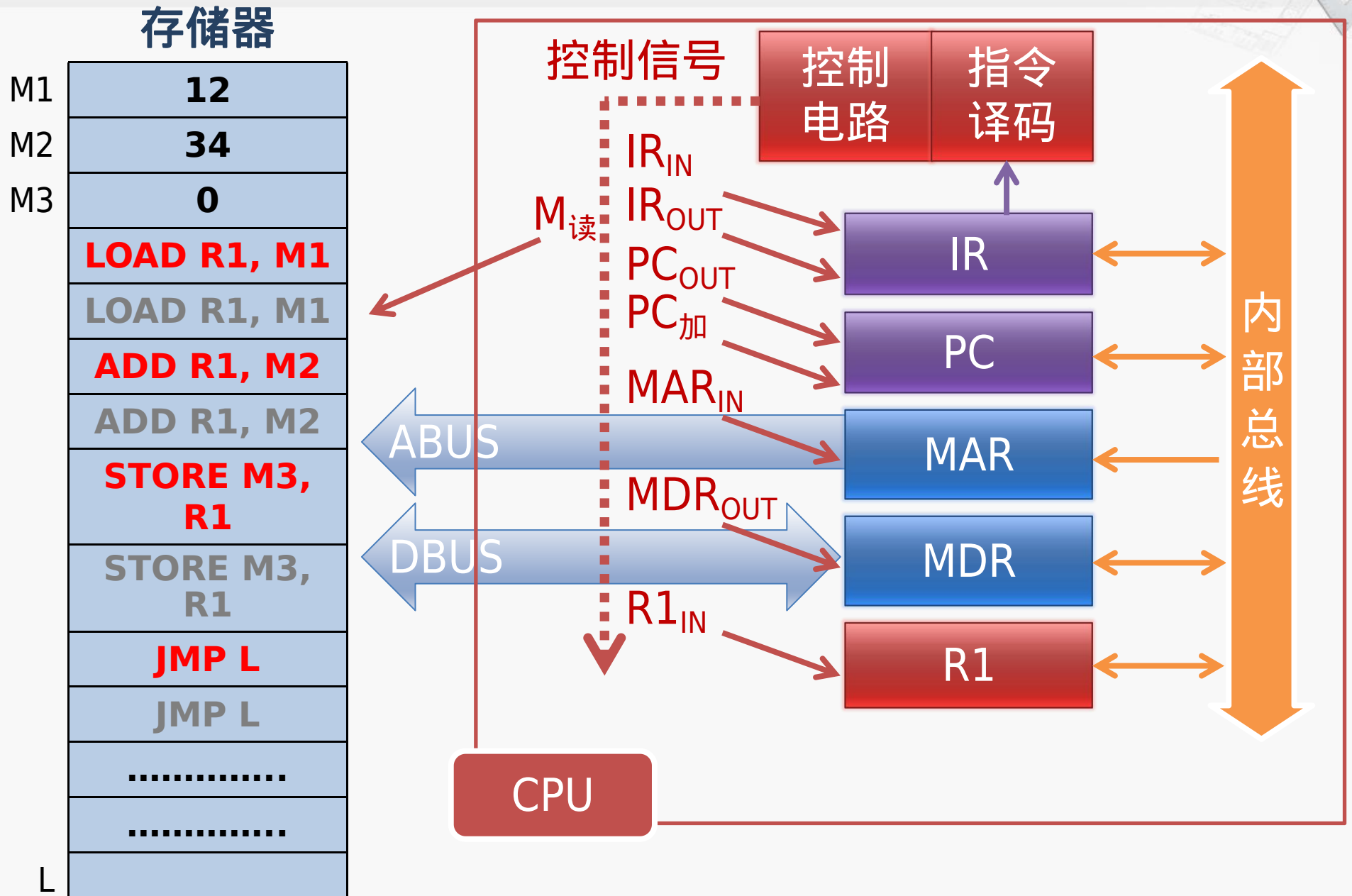
存储器地址	存储器内容	说明
0000 0101	**00001100**	地址M1指向的字节
0000 0110	**00100010**	地址M2指向的字节
0000 0111	**00000000**	地址M3指向的字节
0000 1000	**00001011**	“**LOAD R1, M1**”的第一个字节
0000 1001	**00000101**	“**LOAD R1, M1**”的第二个字节
0000 1010	**00011011**	“ADD R1, M2”的第一个字节
0000 1011	**00000110**	“ADD R1, M2”的第二个字节
0000 1100	**00101011**	“STORE M3, R1”的第一个字节
0000 1101	**00000111**	“STORE M3, R1”的第二个字节
0000 1110	**00110000**	“JMP L”的第一个字节
0000 1111	**00010001**	“JMP L”的第二个字节
0001 0000	**.....**	第五条指令的第一个字节
0001 0001	**.....**	第五条指令的第二个字节
0001 0010	**.....**	第六条指令的第一个字节（地址L指向的字节）

计算机基本结构（模型机）



注：为便于表述，已将模型机进行了适当简化

经过简化的控制信号示例



指令“LOAD　R1, M1”的执行过程

① 取指令并形成下一条指令的地址

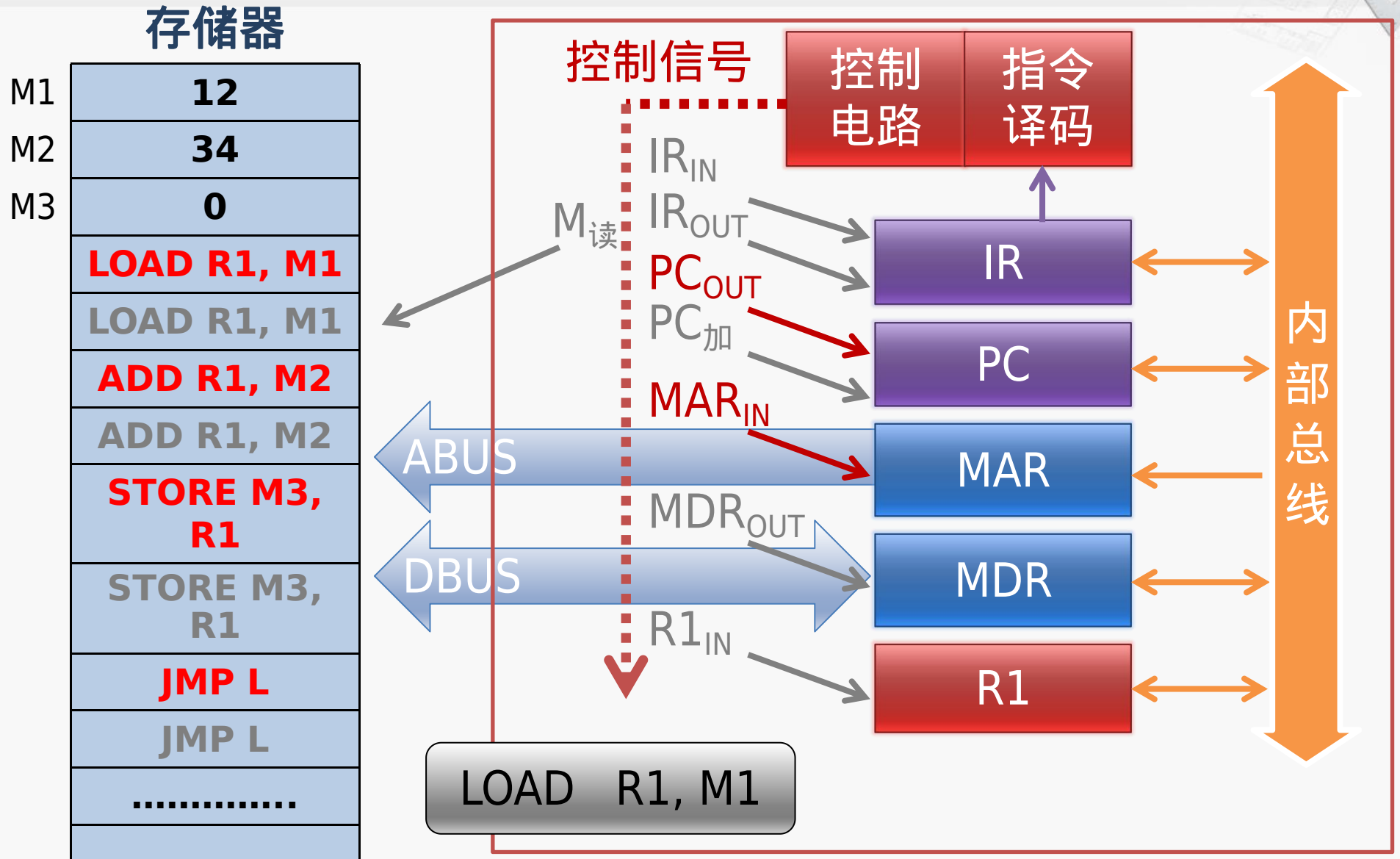
I. 控制器发“PC_{OUT}”和“MAR_{IN}”信号，使PC→MAR

II. 控制器发“$M_{读}$”信号，使存储器的内容（指令）→MDR

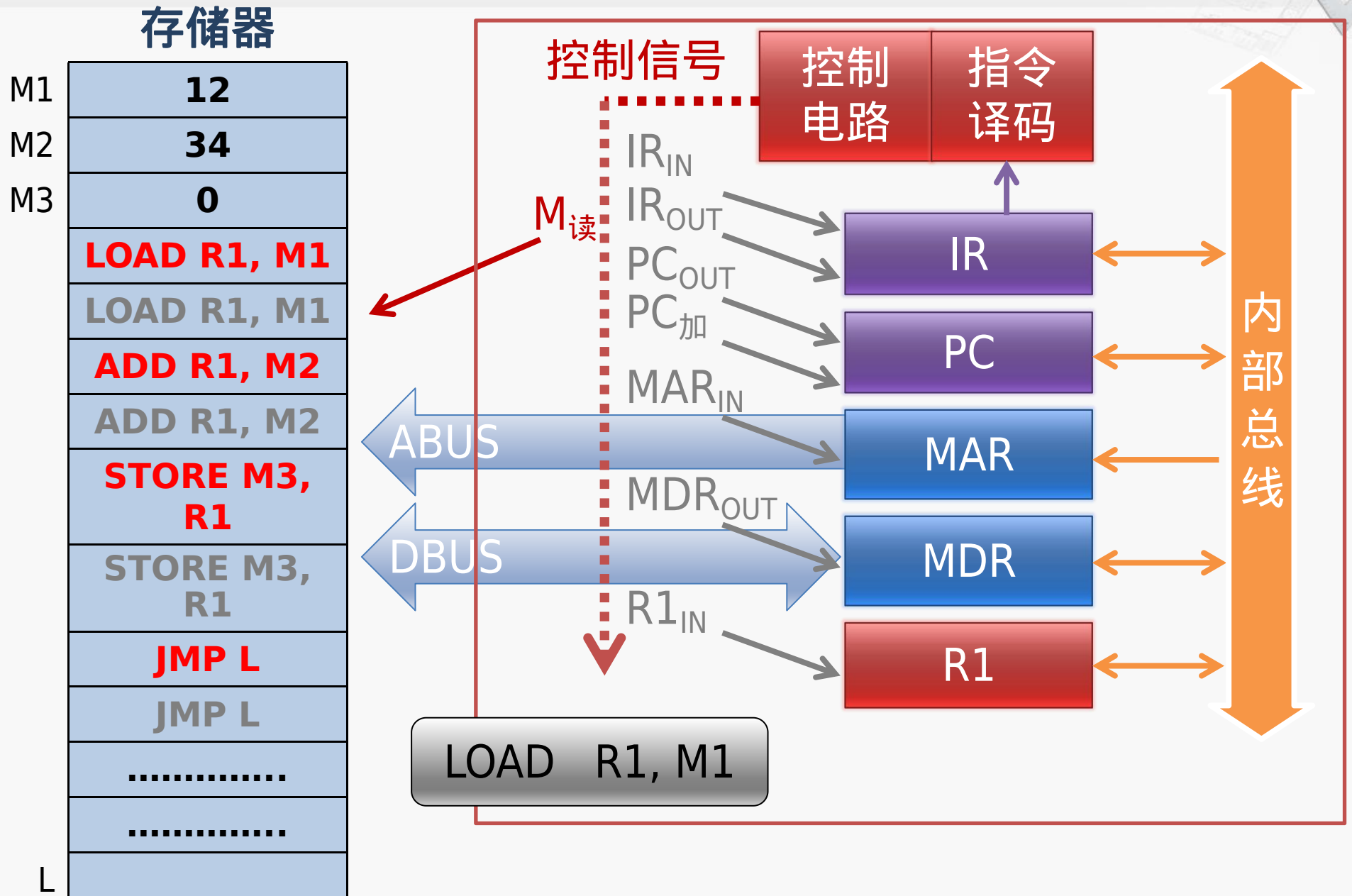
III. 控制器发“$PC_{加}$”信号，使PC+n→PC，其中n为该指令占用的地址数

IV. 控制器发“MDR_{OUT}”和“IR_{IN}”信号，使MDR→IR

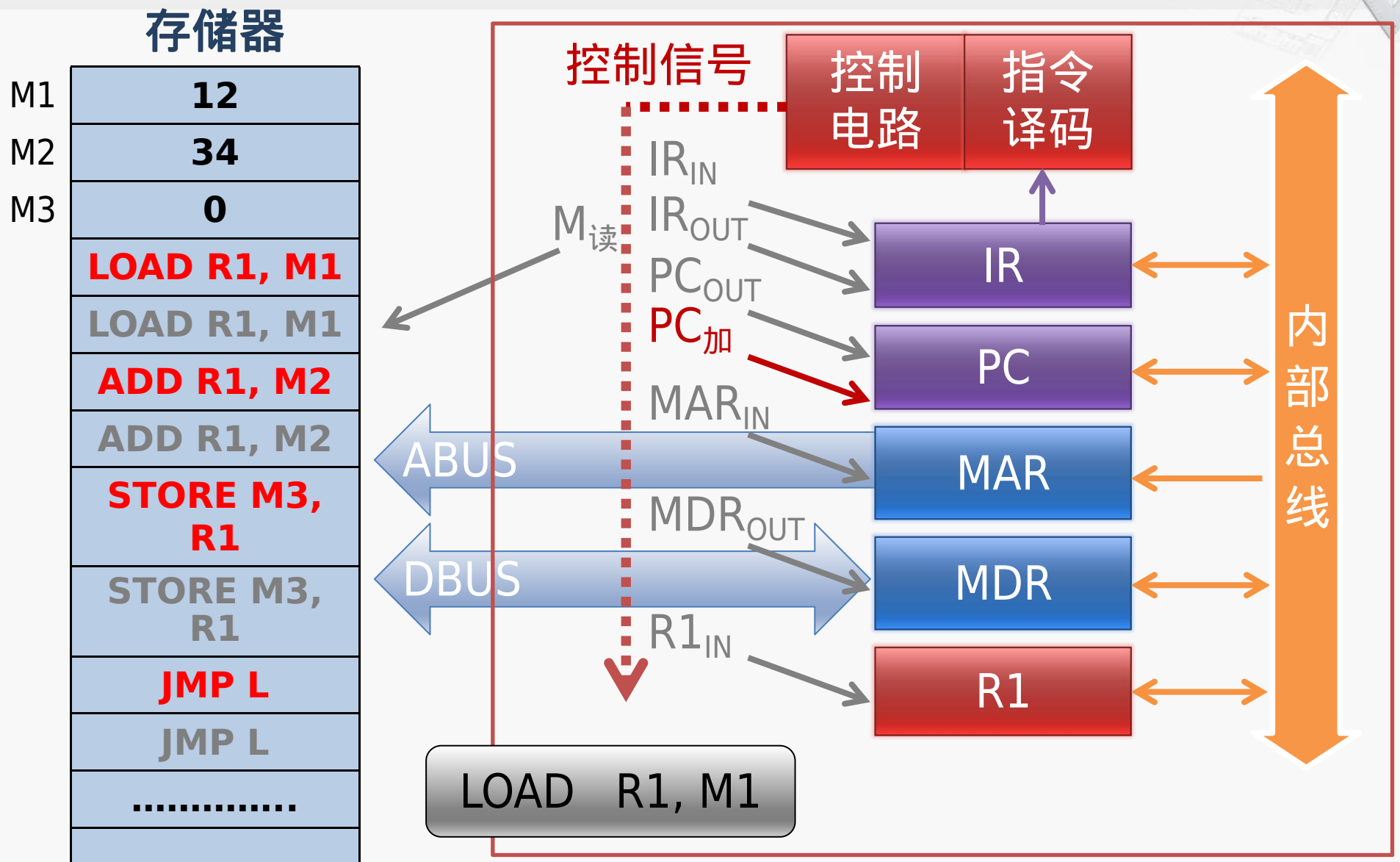
I. 控制器发“PC_{OUT}”和“MAR_{IN}”信号



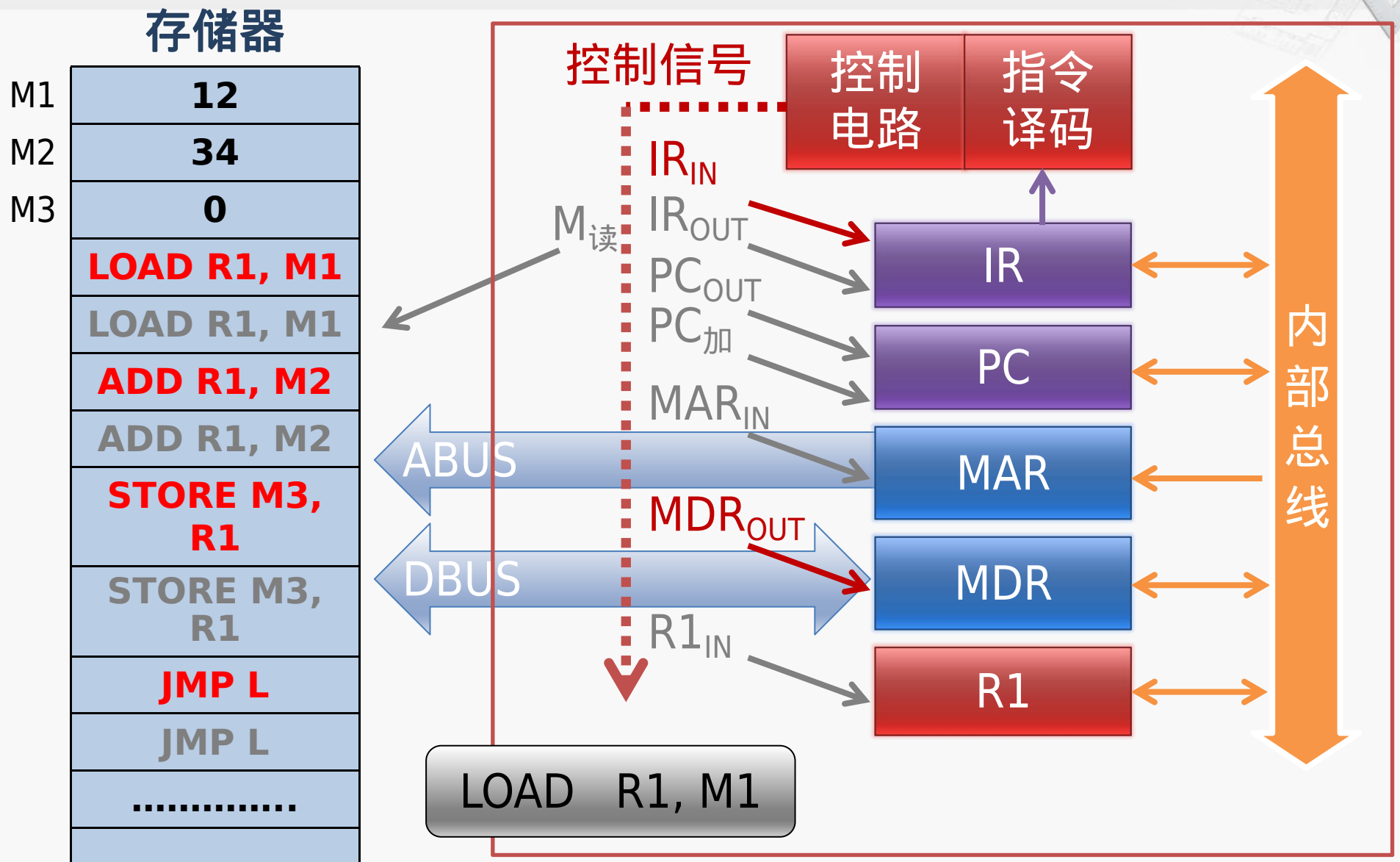
II. 控制器发“$M_{读}$”信号


存储器
M1
12
M2
34
M3
0
LOAD R1, M1
LOAD R1, M1
ADD R1, M2
ADD R1, M2
STORE M3, R1
STORE M3, R1
JMP L
JMP L
.....
.....
L
控制信号
控制电路
指令译码
IR_{IN}
IR_{OUT}
$M_{读}$
PC_{OUT}
$PC_{加}$
MAR_{IN}
ABUS
MDR_{OUT}
DBUS
$R1_{IN}$
IR
PC
MAR
MDR
R1
内部总线
LOAD R1, M1

III. 控制器发“$PC_{加}$”信号



IV. 控制器发“MDR_{OUT}”和“IR_{IN}”信号



指令“LOAD R1, M1”的执行过程（续）

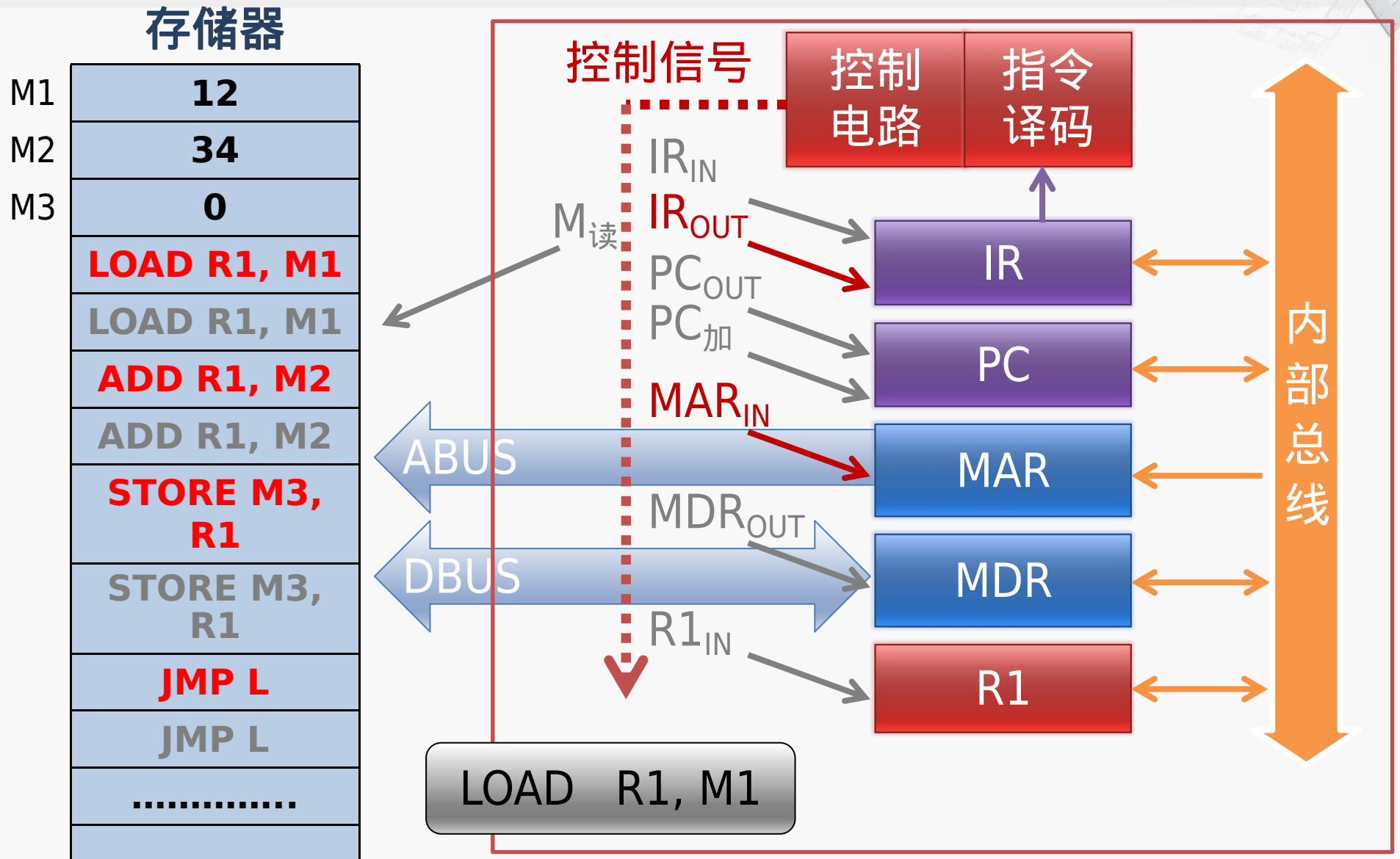
② 执行指令“LOAD　R1, M1”

I. 控制器发“IR_{OUT}”和“MAR_{IN}”信号，使IR中指令的地址段（即M1）→MAR

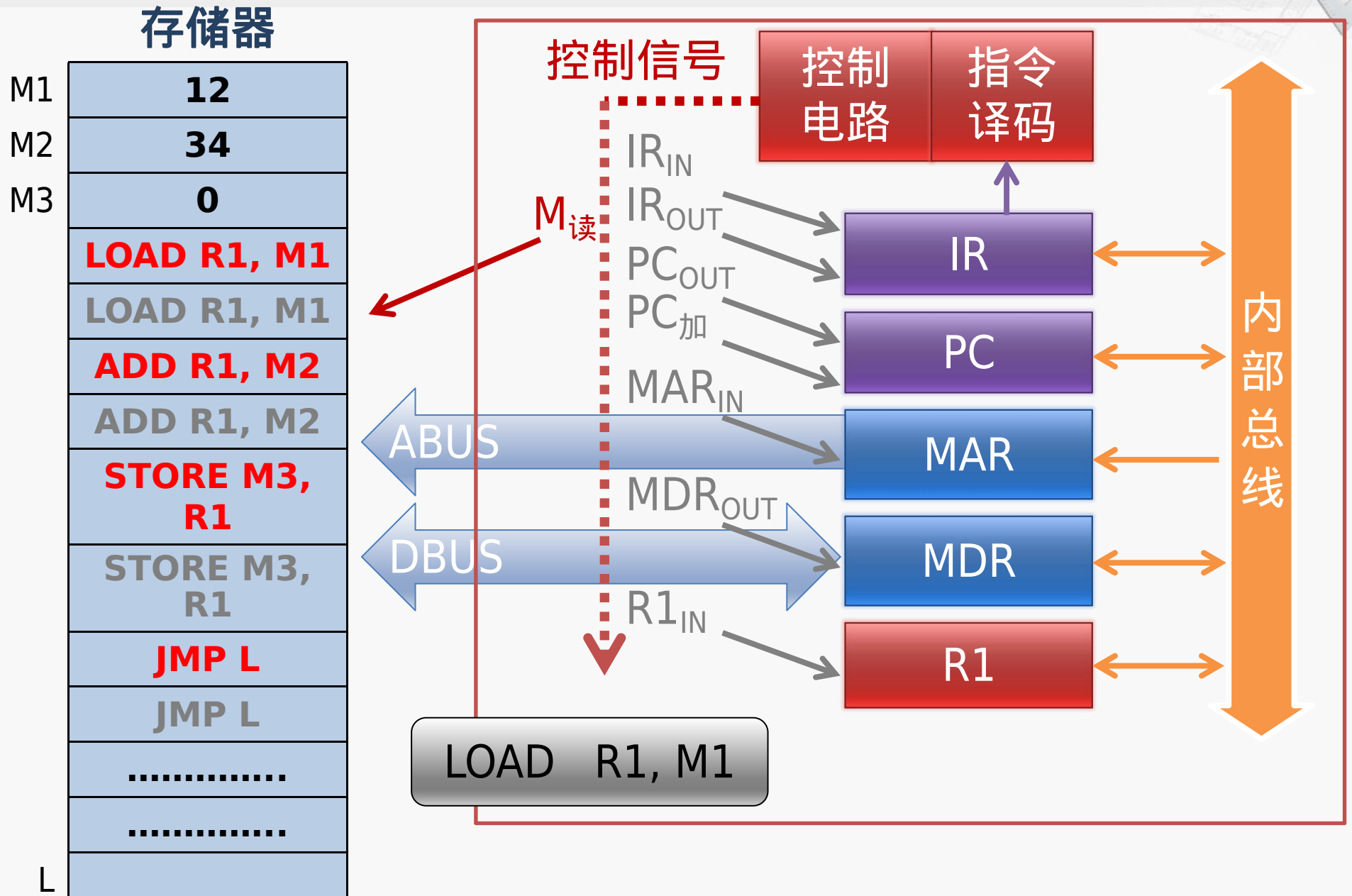
II. 控制器发“$M_{读}$”信号，使存储器M1的内容（数据）→MDR

III. 控制器发“MDR_{OUT}”和“$R1_{IN}$”信号，使MDR→R1

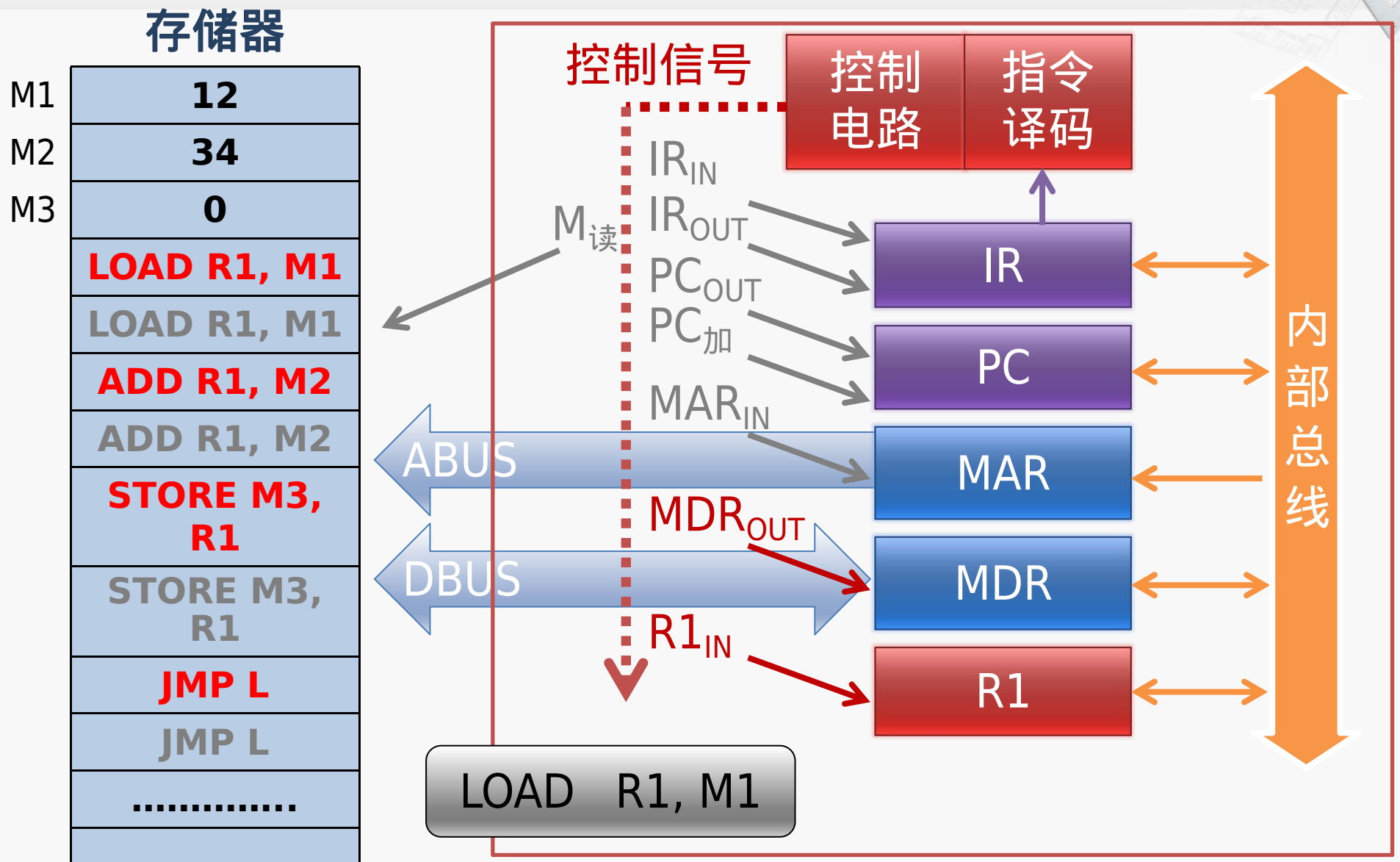
I. 控制器发“IR_{OUT}”和“MAR_{IN}”信号



II. 控制器发“$M_{读}$”信号


存储器
M1
12
M2
34
M3
0
LOAD R1, M1
LOAD R1, M1
ADD R1, M2
ADD R1, M2
STORE M3, R1
STORE M3, R1
JMP L
JMP L
.....
.....
L
控制信号
控制电路
指令译码
IR_{IN}
IR_{OUT}
$M_{读}$
PC_{OUT}
$PC_{加}$
MAR_{IN}
ABUS
MDR_{OUT}
DBUS
$R1_{IN}$
IR
PC
MAR
MDR
R1
内部总线
LOAD R1, M1

III. 控制器发“MDR_{OUT}”和“$R1_{IN}$”信号



二、取指令“ADD R1，M2”并执行

① 取指令“ADD R1，M2”并形成下一条指令的地址

- 操作过程与前一条指令中的第①步相同

二、取指令“ADD R1，M2”并执行

② 执行指令“ADD R1,M2”

I. 控制器发“IR_{OUT}”和“MAR_{IN}”信号，使IR中指令的地址段（即M2）→MAR

II. 控制器发“$M_{读}$”信号，使存储器M2的内容（数据）→MDR，通过DBUS

III. 控制器发“MDR_{OUT}”和“Y_{IN}”信号，使MDR→Y（即ALU的A端）

IV. 控制器发“$R1_{OUT}$”，使R1→内部总线（即ALU的B端）

V. 控制器发“add”信号，使A+B→Z

VI. 控制器发“Z_{OUT}”和“$R1_{IN}$”信号，使Z→R1

三、取指令“STORE M3,R1”并执行

① 取指令“STORE M3, R1”并形成下一条指令的地址

- 操作过程与前一条指令中的第①步相同

② 执行指令“STORE M3, R1”

I. 控制器发“IR_{OUT}”和“MAR_{IN}”信号，使IR中指令的地址段（即M3）→MAR

II. 控制器发“$R1_{OUT}$”和“MDR_{IN}”信号，使R1→MDR

III. 控制器发“$M_{写}$”信号，使MDR→存储器M3

四、取指令JMP L并执行

① 取指令JMP L并形成下一条指令的地址

- 操作过程与前一条指令中的第①步相同

② 执行指令JMP L

I. 控制器发“IR_{OUT}”和“PC_{IN}”信号，使IR中指令的地址（即L）→PC

主要内容

通过学习本课程
了解计算机的发展历程，理解计算机的组成原理，掌握计算机的设计方法

控制器的分类

1. 硬布线控制器
2. 微程序控制器

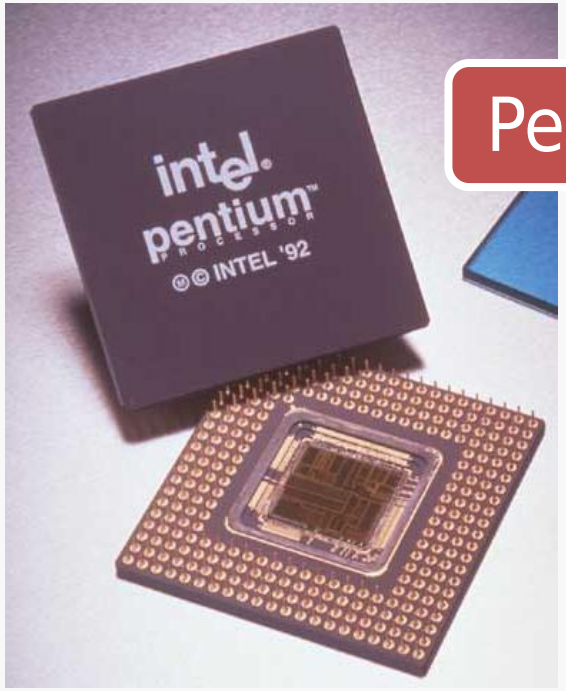
采用硬布线控制器的处理器实例

MIPS

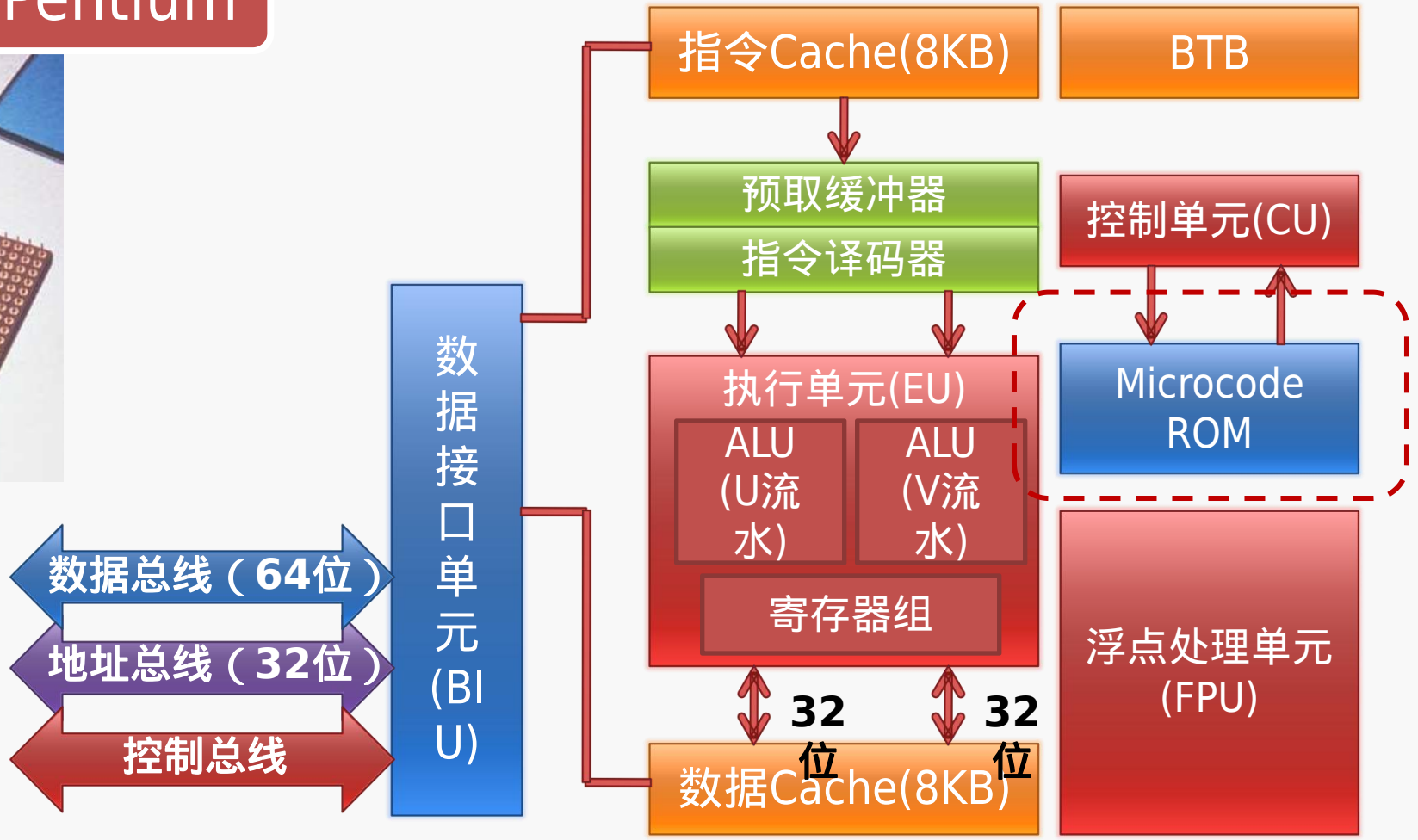
PowerPC

ARM

采用硬布线+微程序控制器的处理器实例



Pentium



主要内容

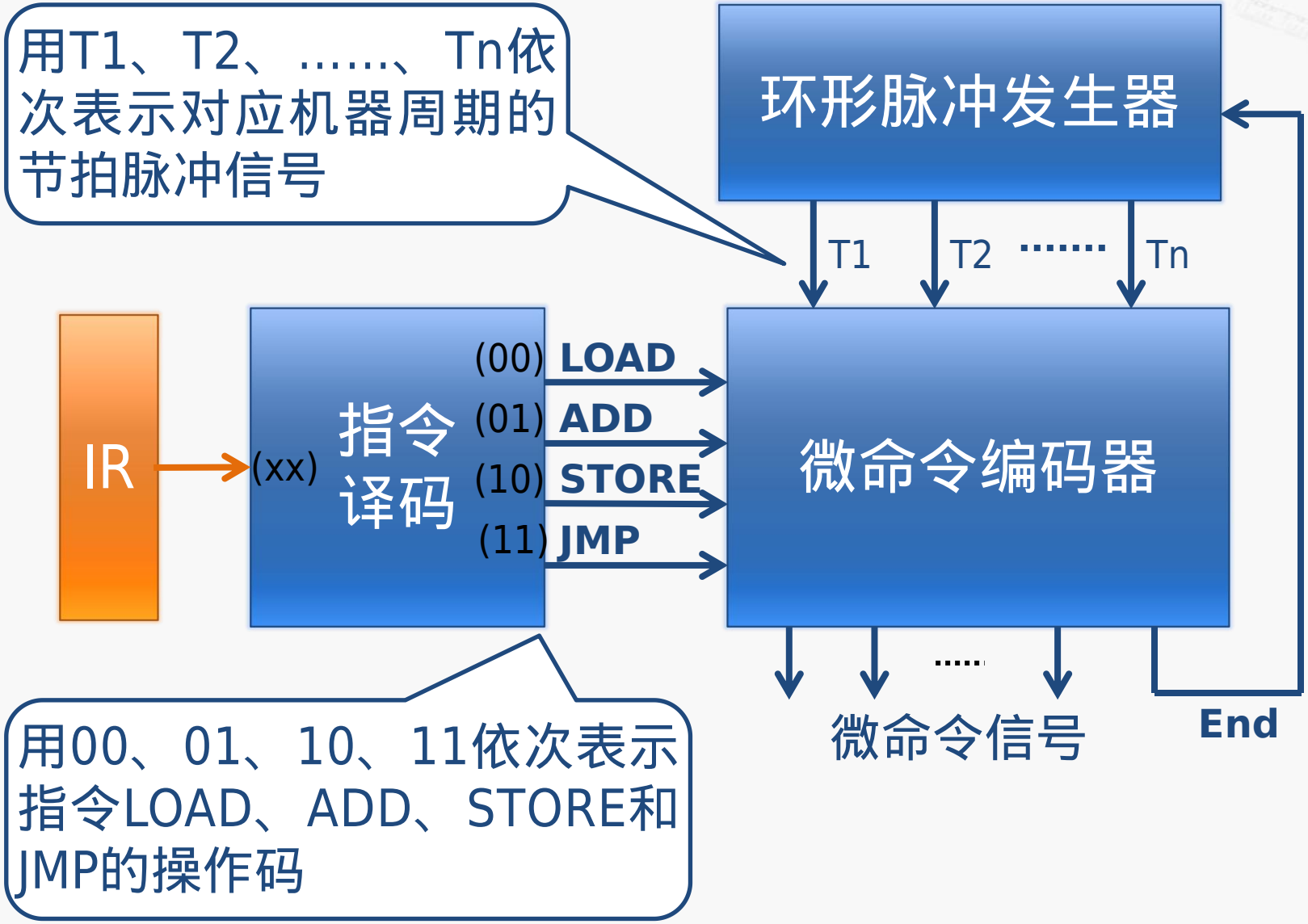
通过学习本课程
了解计算机的发展历程，理解计算机的组成原理，掌握计算机的设计方法

硬布线控制器的基本结构

- 硬布线控制器，也称硬连线控制器或组合逻辑控制器

- 硬布线控制器的主要部件
  - **环形脉冲发生器**：循环地产生节拍脉冲信号
  - **指令译码器**：确定IR中存放的是哪一条指令
  - **微命令编码器**：在不同节拍脉冲信号的同步下产生相应的微命令信号（即控制信号）

硬布线控制器的原理图


用T1、T2、……、Tn依次表示对应机器周期的节拍脉冲信号
环形脉冲发生器
T1
T2
……
Tn
IR
(xx)
指令译码
(00) LOAD
(01) ADD
(10) STORE
(11) JMP
微命令编码器
……
微命令信号
End
用00、01、10、11依次表示指令LOAD、ADD、STORE和JMP的操作码

微命令（控制信号）

- 在取指令和执行指令时，都需要控制器能针对不同的指令在不同的机器周期内发出所需要的各种控制信号

- 例如，在取指令时控制器发出的控制信号
  - 在第一个周期内，发出PC_{OUT}、MAR_{IN}、$M_{读}$、$PC_{加}$共4个控制信号
  - 在第二个周期内，发出MDR_{OUT}和IR_{IN}共2个控制信号

不同机器周期的控制信号

- 不同指令在不同机器周期内应发出的控制信号
  - （End代表指令执行结束的控制信号）

指令名	T1	T2	T3	T4	T5	T6	T7
LOAD (00)	PC_{OUT} MAR_{IN} $M_{读}$ $PC_{加}$	MDR_{OUT} IR_{IN}	IR_{OUT} MAR_{IN} $M_{读}$	MDR_{OUT} $R1_{IN}$	End		
ADD (01)	同上	同上	IR_{OUT} MAR_{IN} $M_{读}$	MDR_{OUT} Y_{IN}	$R1_{OUT}$ add	Z_{OUT} $R1_{IN}$	End
STORE (10)	同上	同上	IR_{OUT} MAR_{IN}	$R1_{OUT}$ MDR_{IN} $M_{写}$	End		
JMP (11)	同上	同上	IR_{OUT} PC_{IN}	End			

用逻辑表达式表示控制信号

- 用一个逻辑表达式来表示某个控制信号应在什么时间发出、对哪些指令发出
  - 用“·”表示“与”，用“+”表示“或”
  - 用Ti表示第i个机器周期的节拍脉冲信号
  - 设所有的信号都是高电平有效

控制信号的逻辑表达式示例

- 以控制信号MAR_{IN}为例
  - 所有的四条指令在T1周期需要
  - LOAD、ADD、STORE三条指令在T3周期需要
  - 其他指令和周期则不需要

- 描述上述关系的逻辑表达式：

$$MAR_{IN} = T1\cdot(LOAD+ADD+STORE+JMP) + T3\cdot(LOAD+ADD+STORE)$$

$$= T1+T3\cdot(LOAD+ADD+STORE)$$

注：设$Ti\cdot(LOAD+ADD+STORE+JMP)=Ti$

各个控制信号的逻辑表达式

PC_{OUT}=T1

$PC_{加}$=T1

IR_{IN}=T2

IR_{OUT}=T3

Y_{IN}=T4·ADD

MDR_{IN}=T4·STORE

add=T5·ADD

Z_{OUT}=T6·ADD

MAR_{IN}=T1+T3·(LOAD+ADD+STORE)

$M_{读}$=T1+T3·(LOAD+ADD)

MDR_{OUT}=T2+T4·(LOAD+ADD)

PC_{IN}=T3·JMP

$R1_{IN}$=T4·LOAD+T6·ADD

$R1_{OUT}$=T4·STORE+T5·ADD

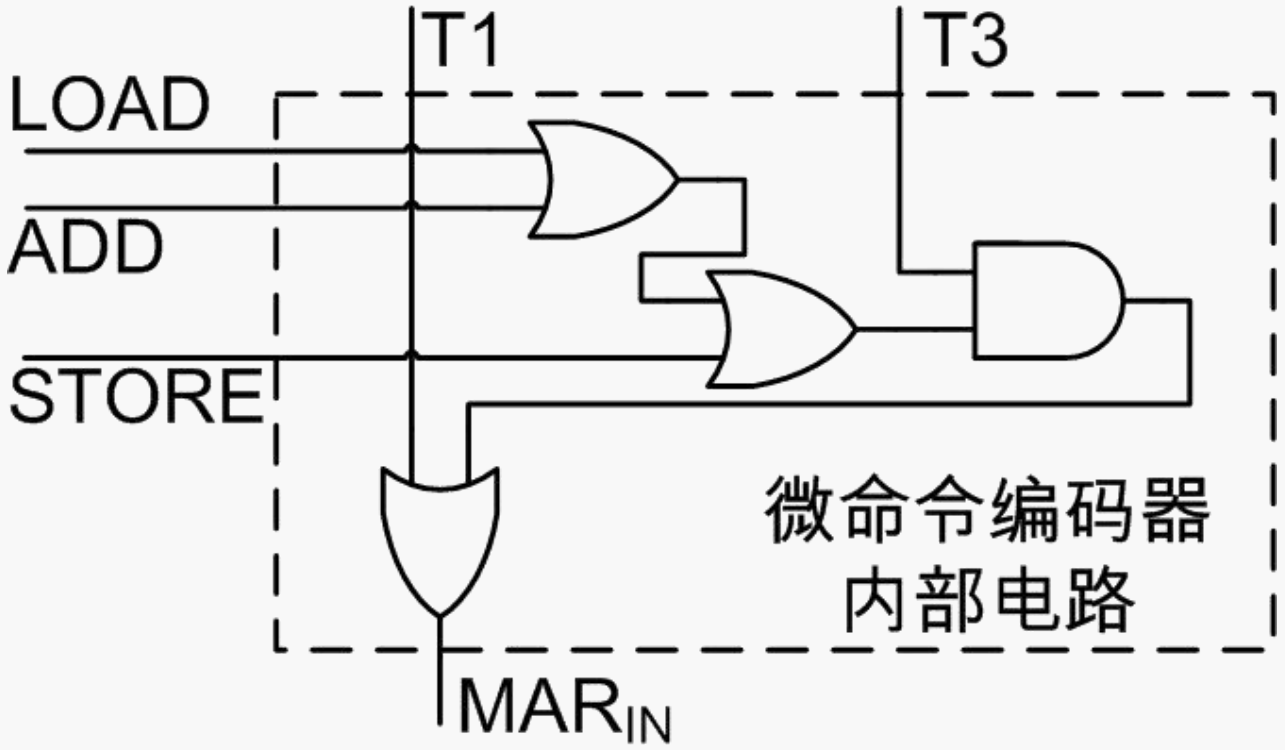
$M_{写}$=T4·STORE

End=T5·(LOAD+STORE)+T7·ADD+T4·JMP

逻辑表达式和微命令编码器内部电路

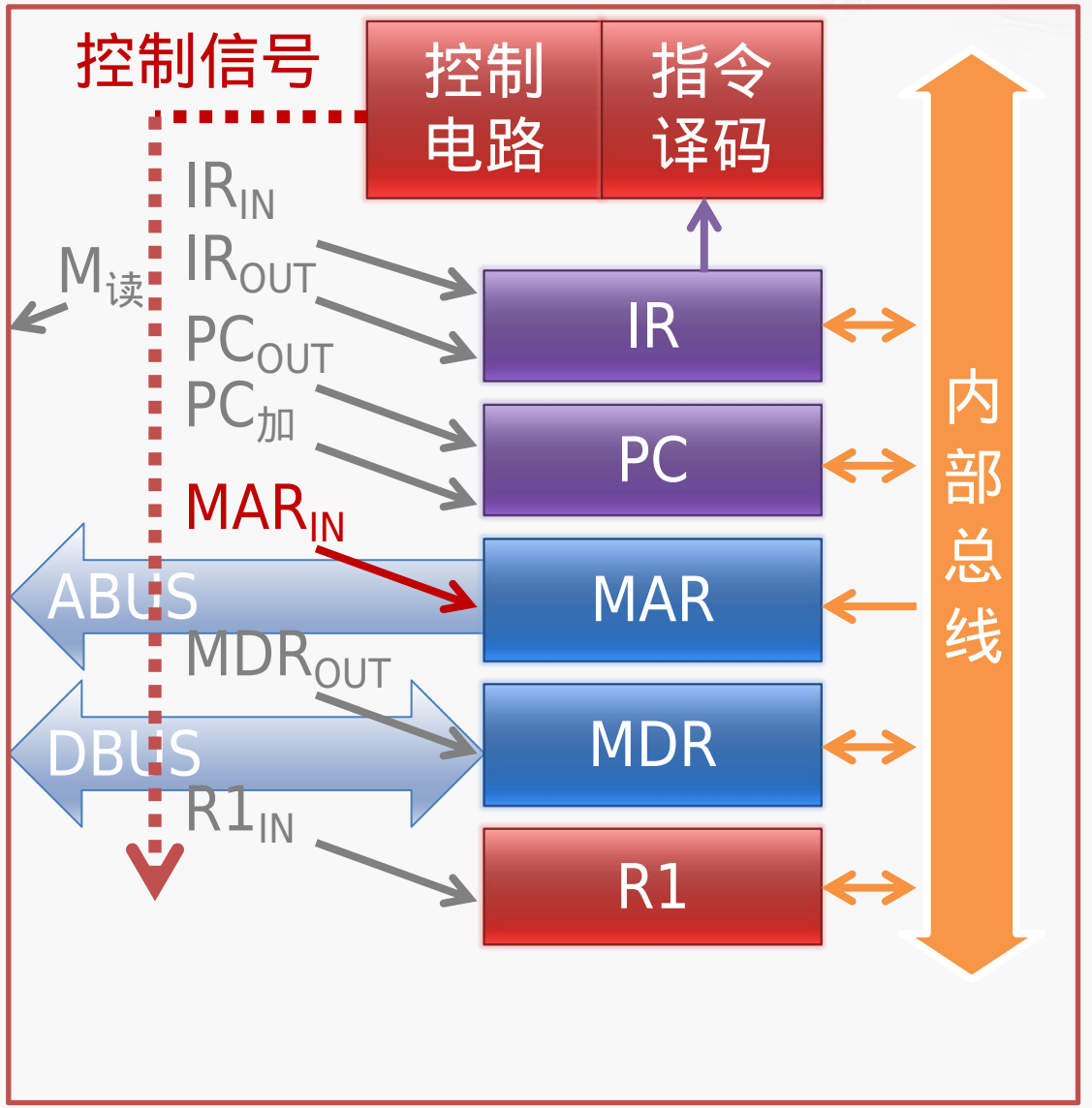
$$MAR_{IN} = T1+T3\cdot(LOAD+ADD+STORE)$$

- 逻辑表达式左端：微命令编码器输出的控制信号
- 逻辑表达式右端：微命令编码器的内部电路



控制器发“MAR_{IN}”信号

- LOAD指令的T1/T3周期
- ADD指令的T1/T3周期
- STORE指令的T1/T3周期
- JMP指令的T1周期



硬布线控制器的优缺点（小结）

- 硬布线控制器关键特征：直接由组合逻辑电路产生微操作控制信号

- 优点：指令执行速度很快

- 缺点：控制逻辑的电路复杂，设计和验证难度大；扩充和修改也很困难

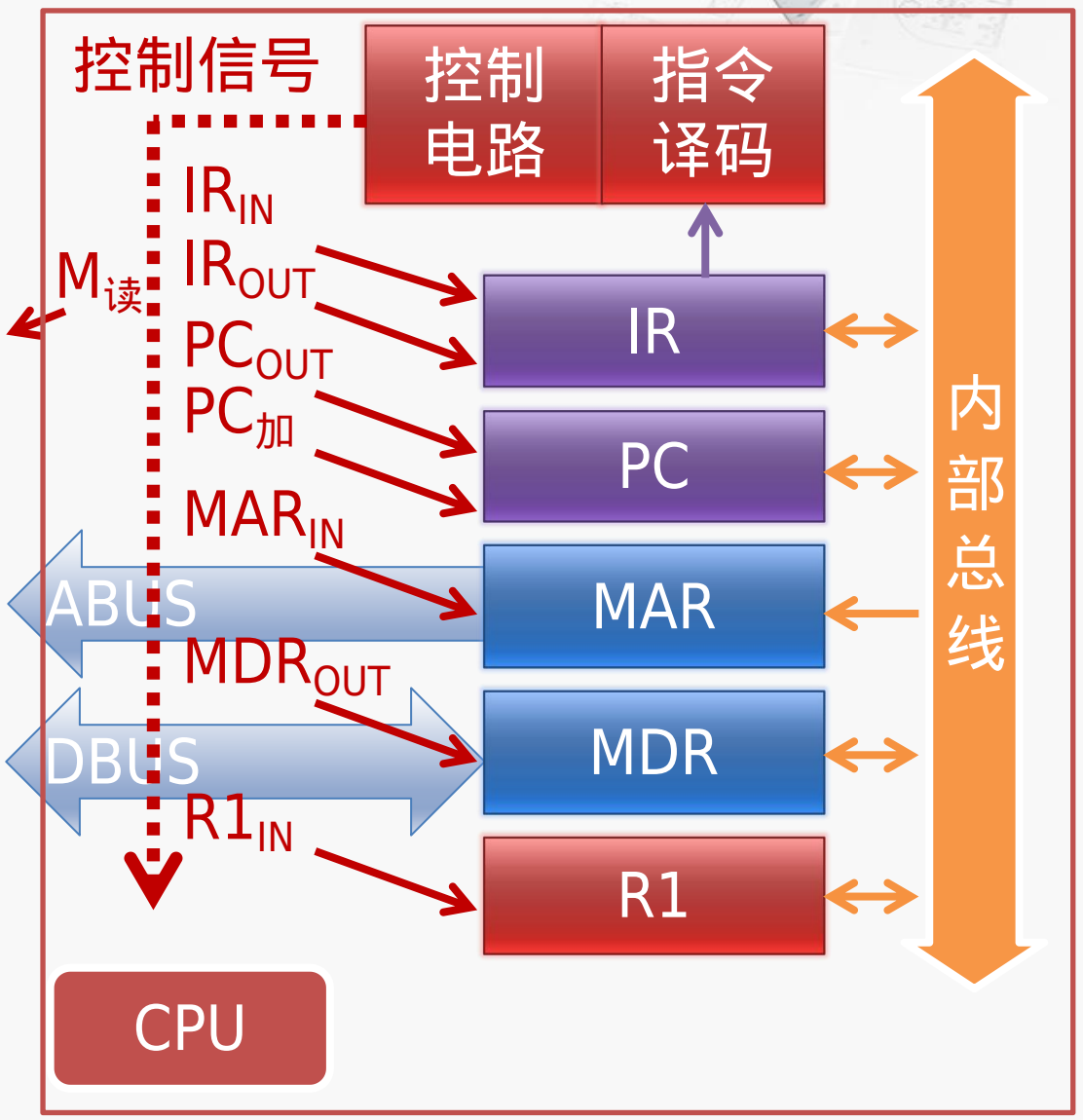
主要内容

通过学习本课程
了解计算机的发展历程，理解计算机的组成原理，掌握计算机的设计方法

微程序控制器的基本思想

第一点

- 计算机每一条机器指令所代表的操作可以被分解为一系列更基本的“微操作”
- 计算机中控制实现这些微操作的命令信号，称为“微命令”



微程序控制器的基本思想

第二点

- 将计算机控制器所要产生的微命令，以代码（微码）形式编成“微指令”
- 在制造CPU时，将微命令存储在CPU内部的一个只读存储器（ROM）
- 在CPU执行程序时，从ROM中取出微指令，根据它所包含的微命令，控制相关部件的操作

微指令（即控制信号序列）

- 本模型机中，控制器发出的控制信号共16个

- 对某一机器周期，微指令形式的信号序列的含义：
  - 需要发出的控制信号用1表示
  - 不需要发出的控制信号用0表示

不同机器周期的控制信号

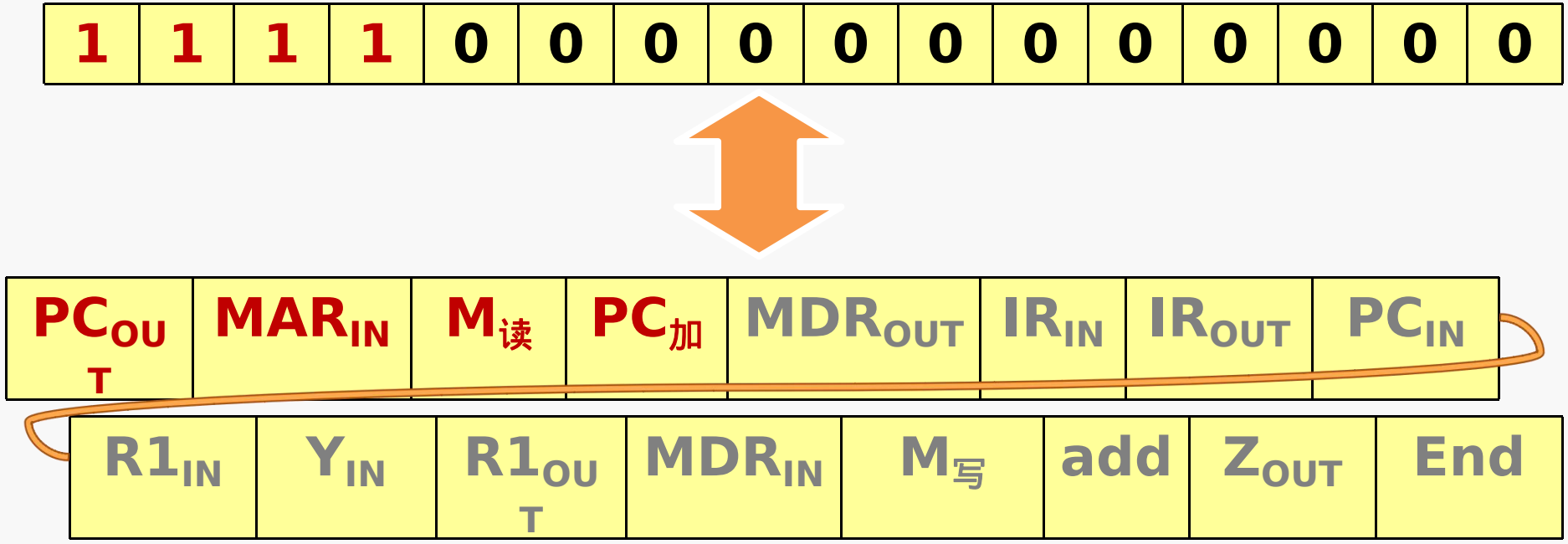
- 每条指令的T1周期，都要发出PC_{OUT}、MAR_{IN}、$M_{读}$和$PC_{加}$共四个信号

指令名	T1	T2	T3	T4	T5	T6	T7
LOAD (00)	**PC_{OUT} MAR_{IN} $M_{读}$ $PC_{加}$**	MDR_{OUT} IR_{IN}	IR_{OUT} MAR_{IN} $M_{读}$	MDR_{OUT} $R1_{IN}$	End		
ADD (01)	同上	同上	IR_{OUT} MAR_{IN} $M_{读}$	MDR_{OUT} Y_{IN}	$R1_{OUT}$ add	Z_{OUT} $R1_{IN}$	End
STORE (10)	同上	同上	IR_{OUT} MAR_{IN}	$R1_{OUT}$ MDR_{IN} $M_{写}$	End		
JMP (11)	同上	同上	IR_{OUT} PC_{IN}	End			

微指令示例1

- 每条指令的T1周期，都要发出PC_{OUT}、MAR_{IN}、$M_{读}$和$PC_{加}$共四个信号

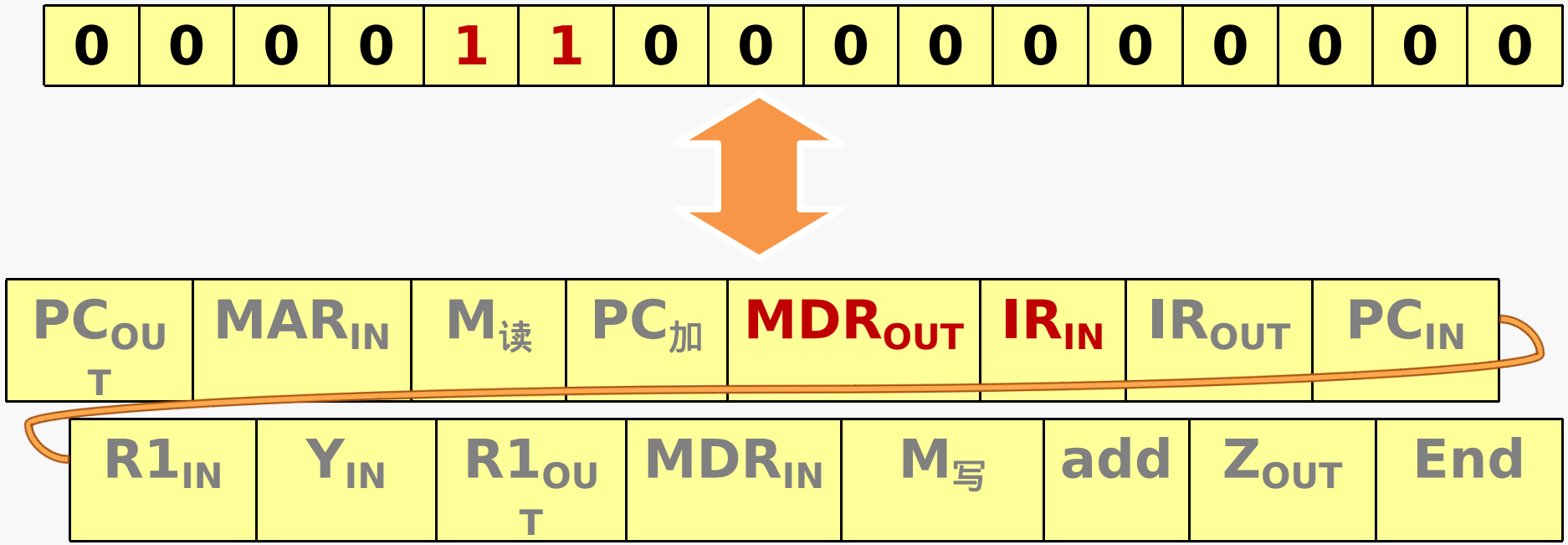
- 这条微指令形式的信号序列如下：



微指令示例2

- 每条指令的T2周期，都要发MDR_{OUT}、IR_{IN}两个信号

- 这条微指令形式的信号序列如下：



不同机器周期的控制信号

- ADD指令的T3周期，要发出**IR_{OUT}、MAR_{IN}、**和**$M_{读}$**共三个信号

指令名	T1	T2	T3	T4	T5	T6	T7
LOAD (00)	PC_{OUT} MAR_{IN} $M_{读}$ $PC_{加}$	MDR_{OUT} IR_{IN}	IR_{OUT} MAR_{IN} $M_{读}$	MDR_{OUT} $R1_{IN}$	End		
ADD (01)	同上	同上	**IR_{OUT}** **MAR_{IN}** **$M_{读}$**	MDR_{OUT} Y_{IN}	$R1_{OUT}$ add	Z_{OUT} $R1_{IN}$	End
STORE (10)	同上	同上	IR_{OUT} MAR_{IN}	$R1_{OUT}$ MDR_{IN} $M_{写}$	End		
JMP (11)	同上	同上	IR_{OUT} PC_{IN}	End			

微指令示例3

- ADD指令的T3周期，要发出IR_{OUT}、MAR_{IN}、和$M_{读}$共三个信号

- 这条微指令形式的信号序列如下：

0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0

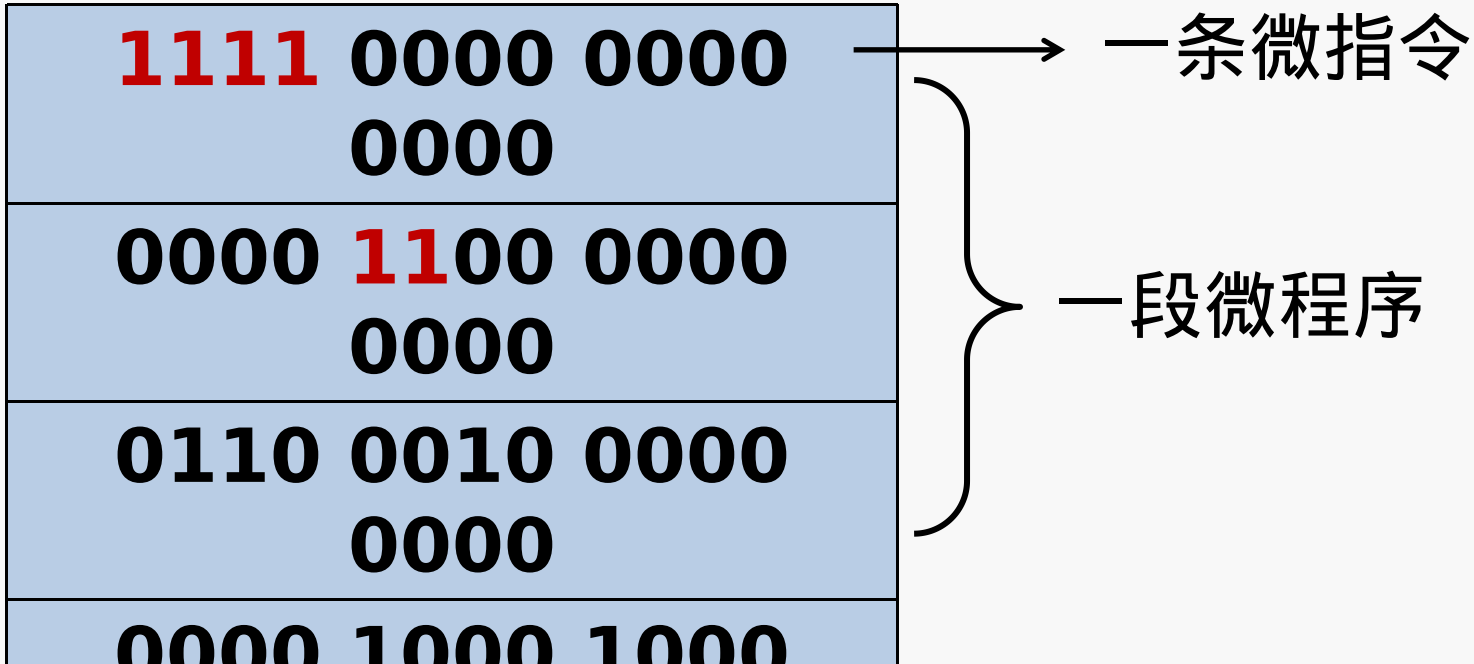
PC_{OUT}	MAR_{IN}	$M_{读}$	$PC_{加}$	MDR_{OUT}	IR_{IN}	IR_{OUT}	PC_{IN}

$R1_{IN}$	Y_{IN}	$R1_{OUT}$	MDR_{IN}	$M_{写}$	add	Z_{OUT}	End

微程序控制器的基本思想

第三点

- 若干条微指令组成一小段“微程序”，解释执行一条机器指令
- 针对整个指令系统的需要，编制出一套完整的微程序



微程序控制器的起源

- 微程序控制的概念
  - 最早由剑桥大学的威尔克斯（M·V·Wilkes）于1951年提出来，并在1953年的一篇文章中发表了一个微程序控制的实例
  - 由于缺少简单快速的微程序存储手段，当时这一项发明并未得到广泛采用

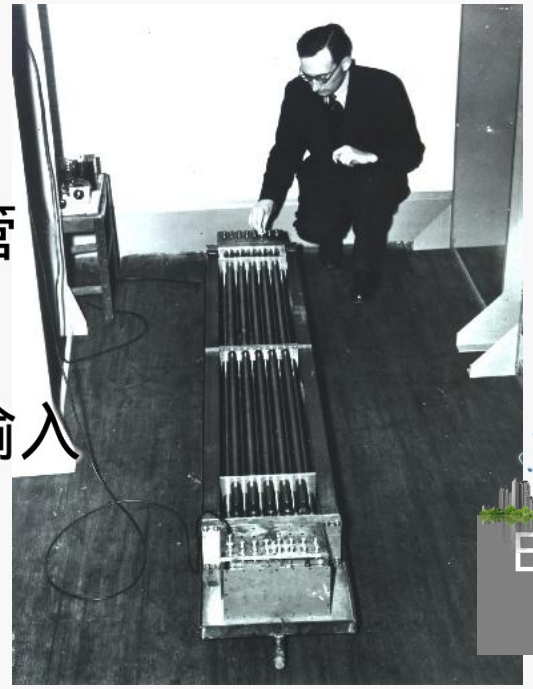
- 微程序的发展
  - 1964年，首次在当时最有影响的IBM360系列计算机中实际应用，成功地解决了不同的计算机之间指令系统兼容性问题

回顾：关于威尔克斯

- 1949年5月6日正式运行
- 英国剑桥大学数学实验室研制
- 主设计师：莫里斯·威尔克斯
- 以EDVAC为蓝本设计和建造

五个组成部分

- 运算器和控制器：电子管
- 存储器：水银延迟线
- 输入设备：从穿孔纸带输入
- 输出设备：电传打印机

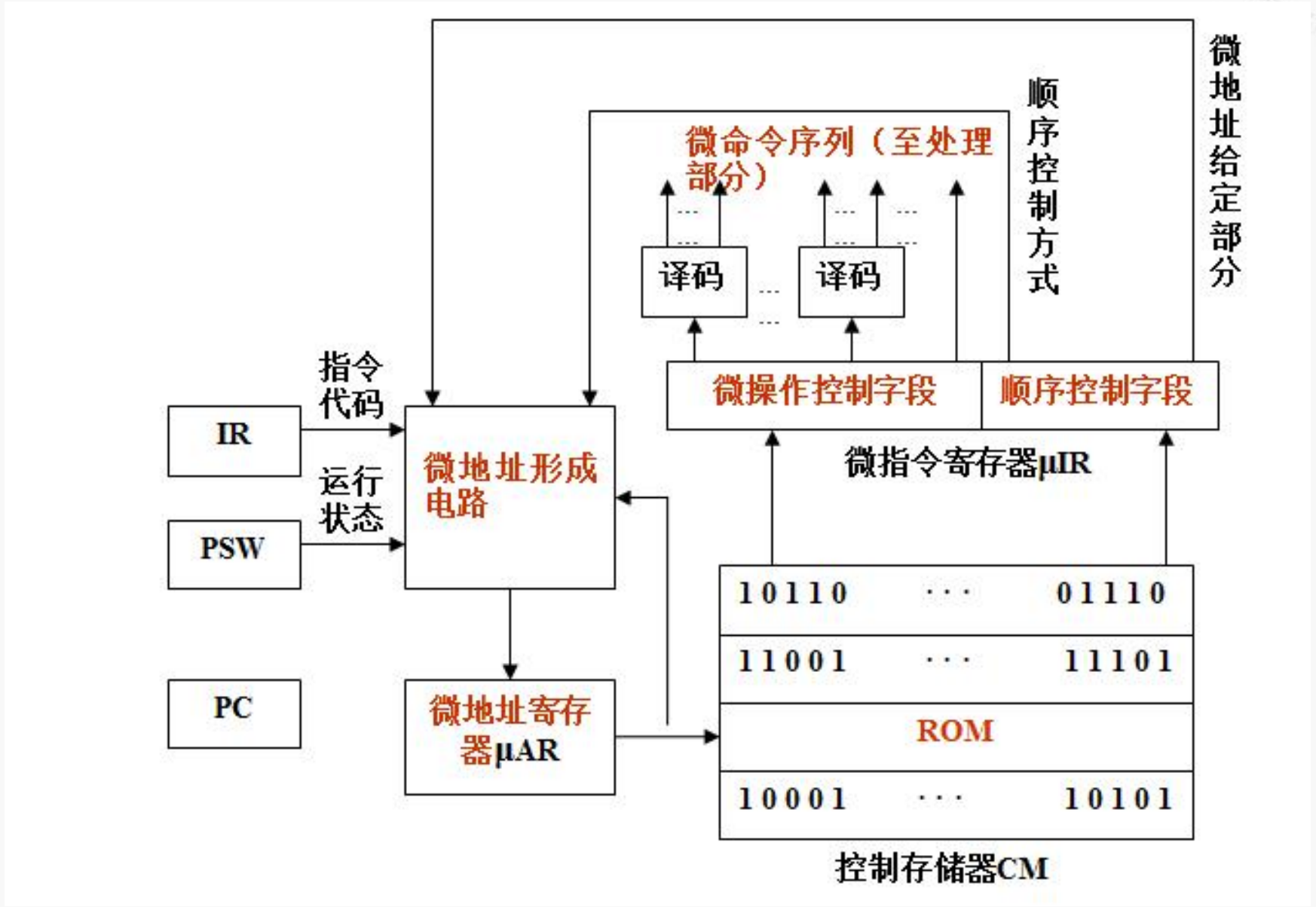


EDSAC：

第一台实用的存储程序式计算机

Electronic Delay Storage Automatic Calculator
电子延迟存储自动计算器

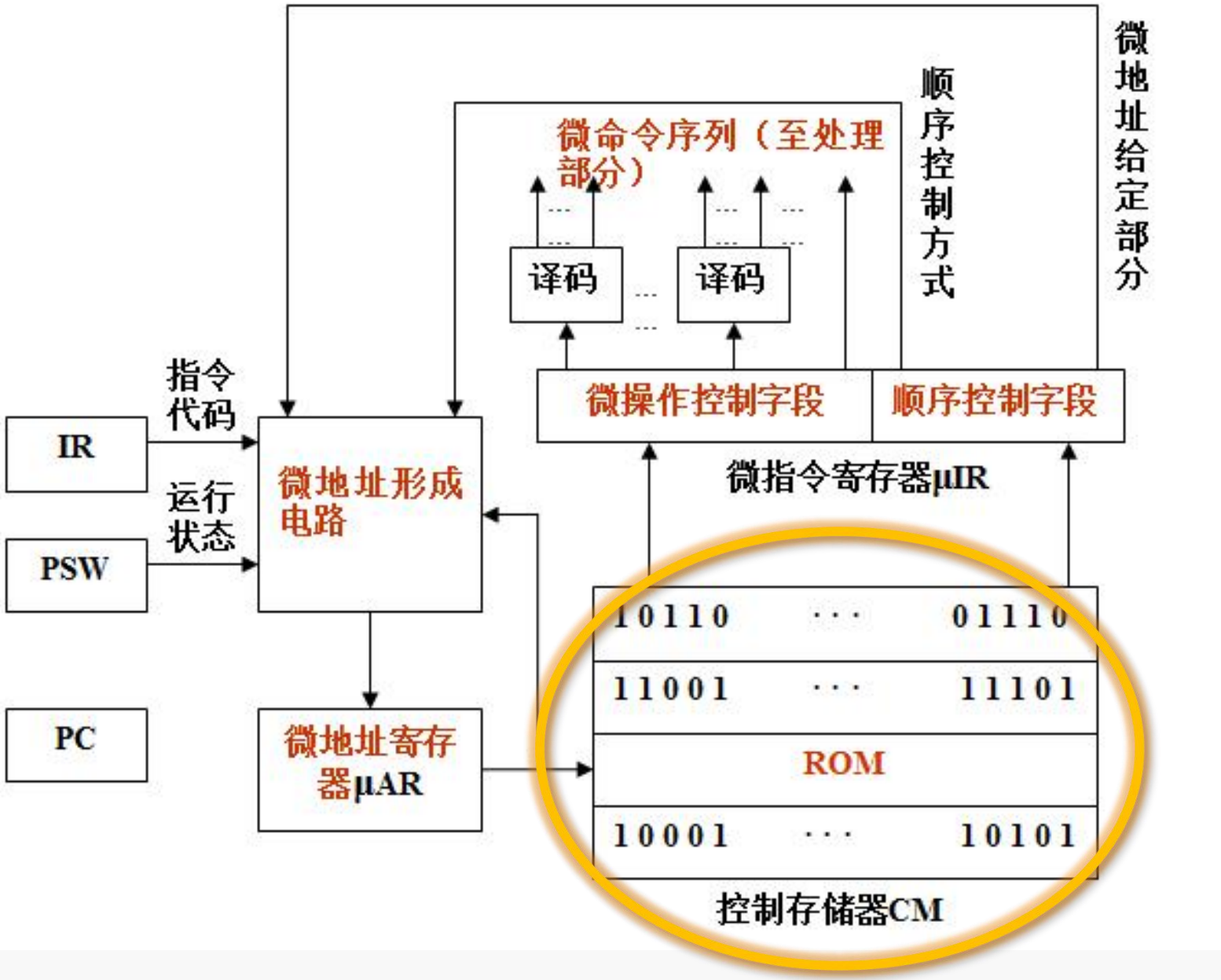
微程序控制器的基本结构



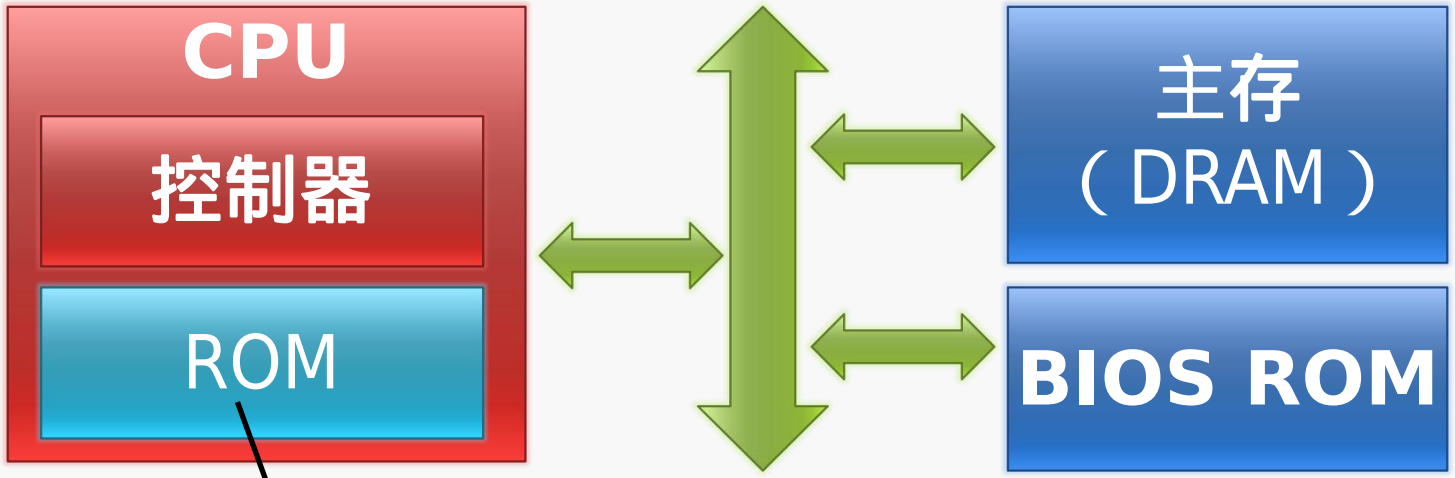
微程序控制器主要部件的功能

控制存储器CM

- 用来存放微程序，每个存储单元存放一个微指令代码
- 通常采用只读存储器（ROM）构成，在制造CPU时写入微程序代码



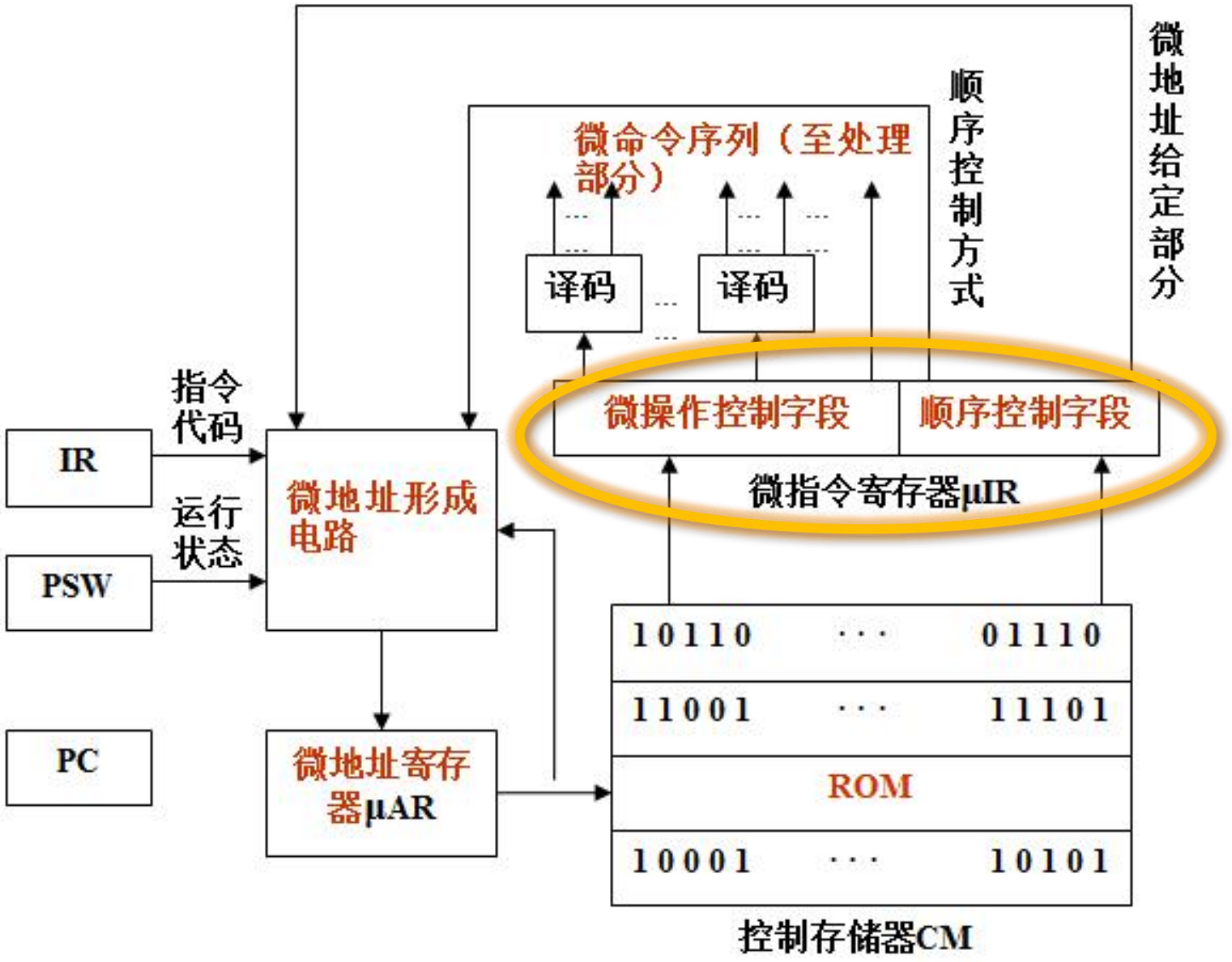
微程序控制器中的控制存储器



地址	内容	
0000	**1111 0000 0000 0000**	一条微指令
0001	**0000 1100 0000 0000**	一段微程序
0010	**0110 0010 0000 0000**	

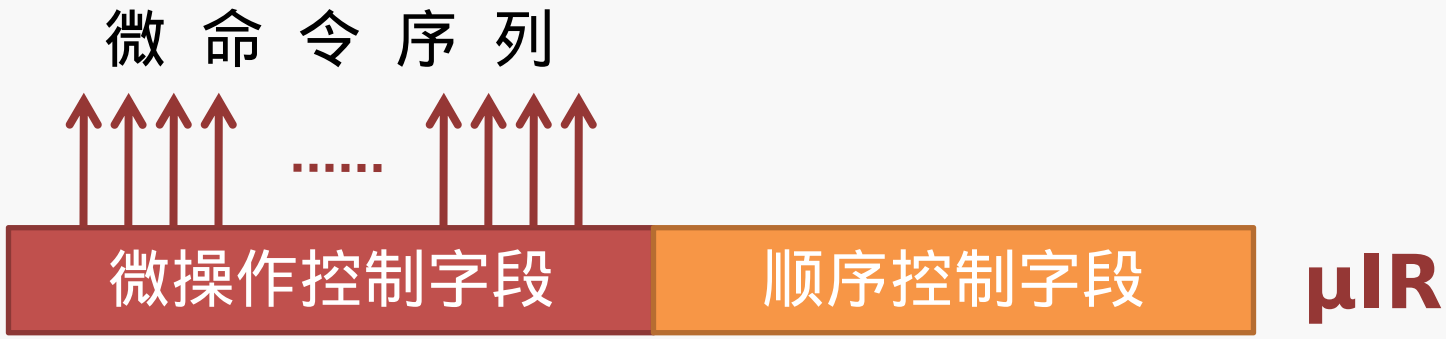
微程序控制器主要部件的功能

- 微指令寄存器μIR
  - 用来存放从CM中读出的微指令
  - 包括“微操作控制字段”和“顺序控制字段”两个部分
- 微指令的编码形式
  - 直接表示法
  - 编码表示法
  - 混合表示法



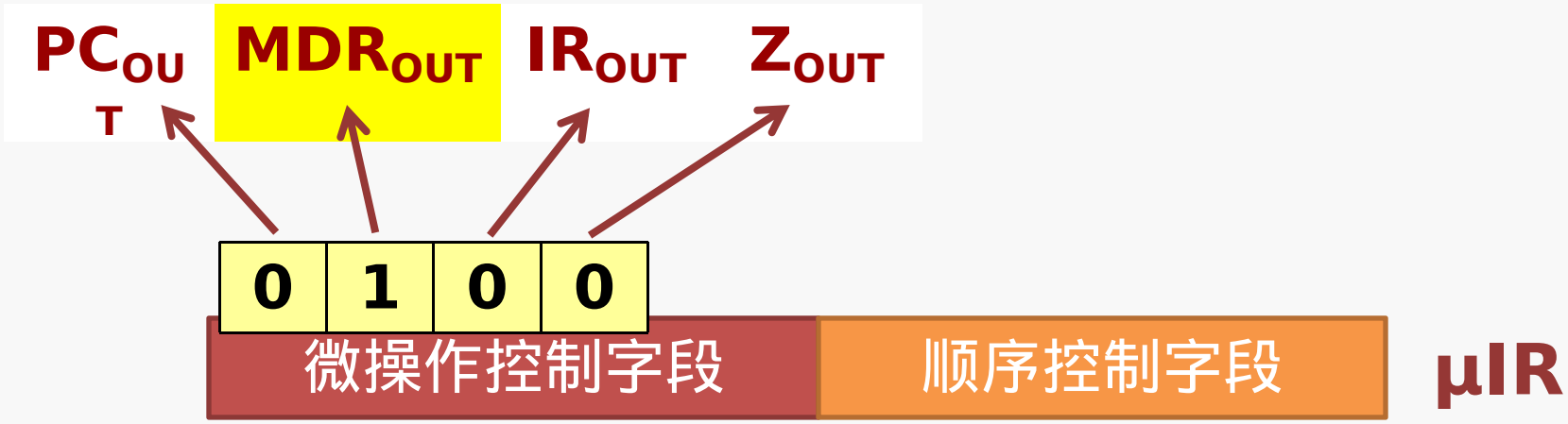
微指令的编码方式 之一

- 直接表示法
  - 将每个控制信号都作为微指令中的一位来编码的方法称为微指令的直接表示法，也称直接控制法



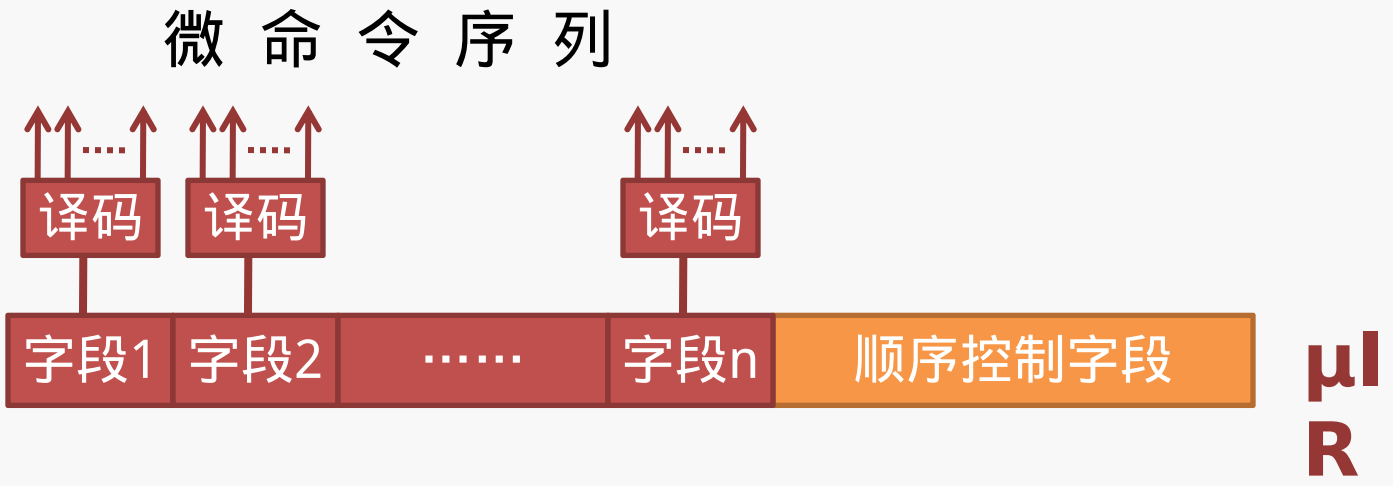
微指令的编码方式 之一（续）

- 优点：简单直观，其输出直接用于控制
- 缺点：编码效率低，因为设置的多个控制位中，对于某一条微指令而言，只有少数几位是有效的



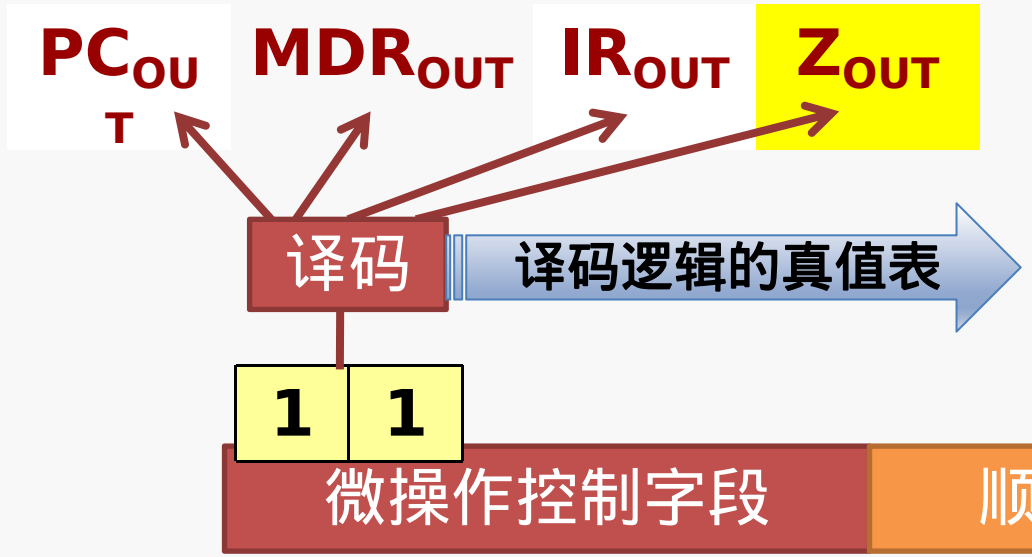
微指令的编码方式 之二

- 编码表示法
  - 将微指令代码进行分组编码，将不能同时出现的相斥信号分在一组中，然后将其编码成较短的代码



微指令的编码方式 之二（续）

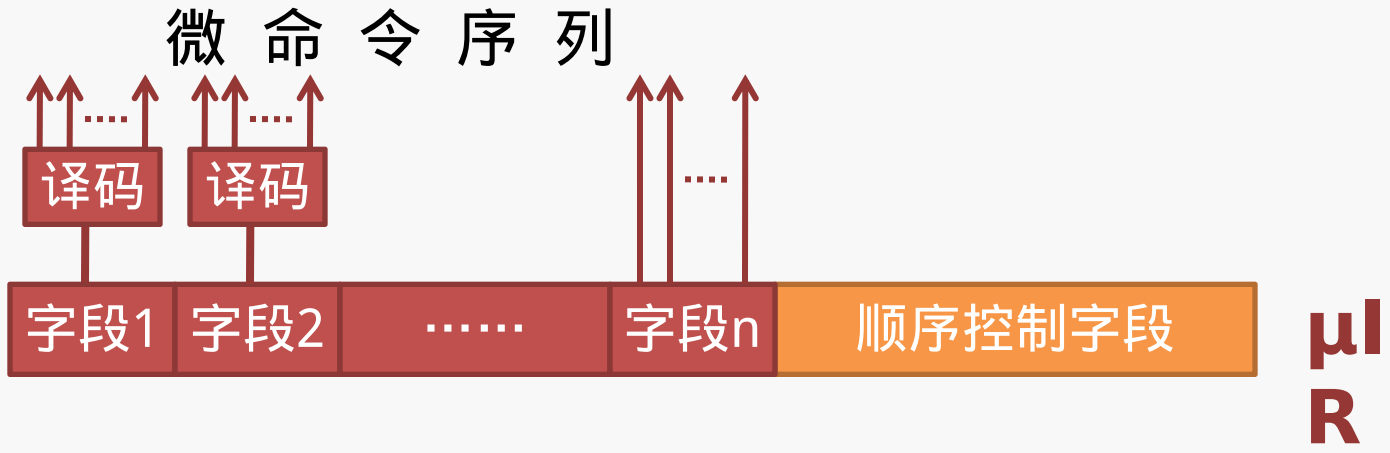
- **优点：减少控制存储器所需存储的代码数量**
- **缺点：**编码的微指令代码需经译码后才能成为控制信号，**增加译码器的硬件开销，增加控制信号的延迟**，从而影响微程序执行速度



输入	输出	说明
00	1000	PC_{OUT}有效
01	0100	MDR_{OUT}有效
10	0010	IR_{OUT}有效
11	0001	Z_{OUT}有效

微指令的编码方式 之三

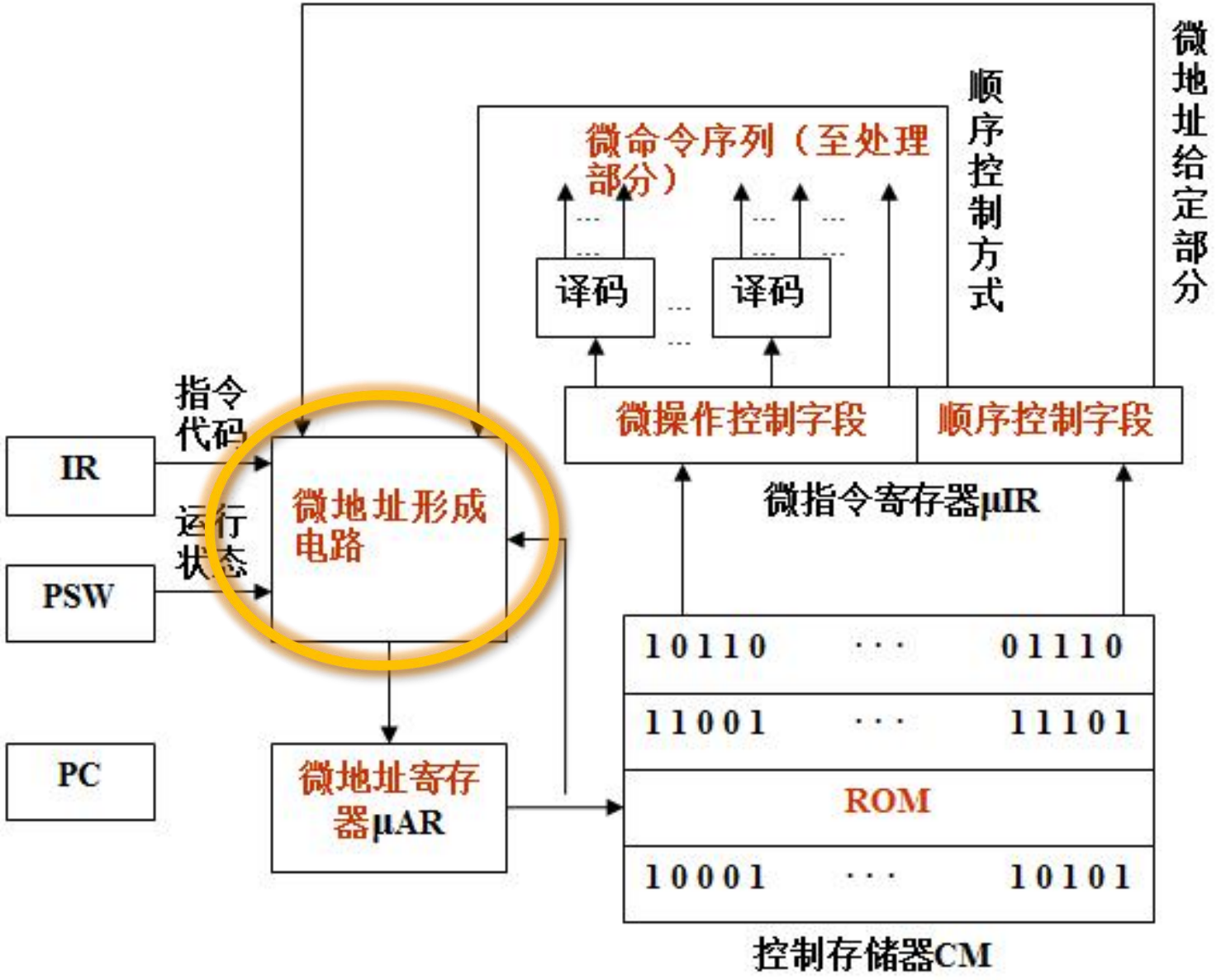
- 混合表示法
  - 直接表示法与字段编码表示法相结合
  - 采用“直接表示法”：对速度要求高，或者与其它控制信号都相斥的控制信号
  - 采用“编码表示法”：其它控制信号



微程序控制器主要部件的功能

微地址形成电路

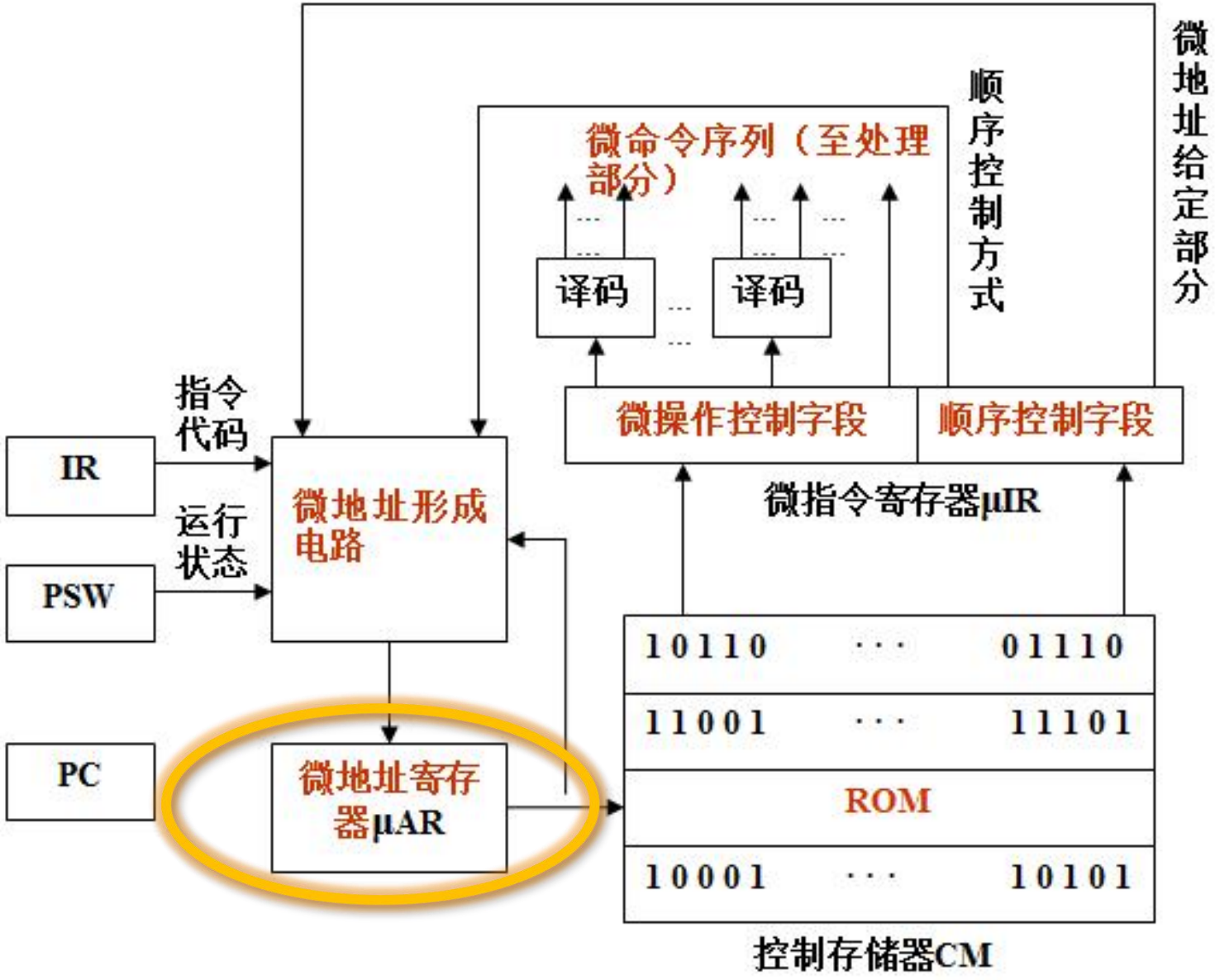
- 形成下一条微指令的微地址
- 形成依据包括：
  - 微地址给定部分
  - 现行微指令中的顺序控制方式（如是否发生转移）
  - 机器指令的有关代码（如操作码）
  - 机器运行状态等



微程序控制器主要部件的功能

微地址寄存器μAR

- 保存微指令对应的微地址，指向相应的CM单元
- 当读出一条微指令后，微地址形成电路将后继微地址存入μAR中，从而指向下一个CM单元



机器指令的读取与执行

1. 在开始执行机器指令时，先从控制存储器中读取“取机器指令用的微指令”，它所包含的微命令使CPU访问主存储器，读取机器指令，送入指令寄存器IR，然后修改程序计数器PC的内容

（注：在微程序中有一条或几条微指令，其微命令是实现“取指”的操作，称为**“取机器指令用的微指令”**，属于微程序的公用部分）

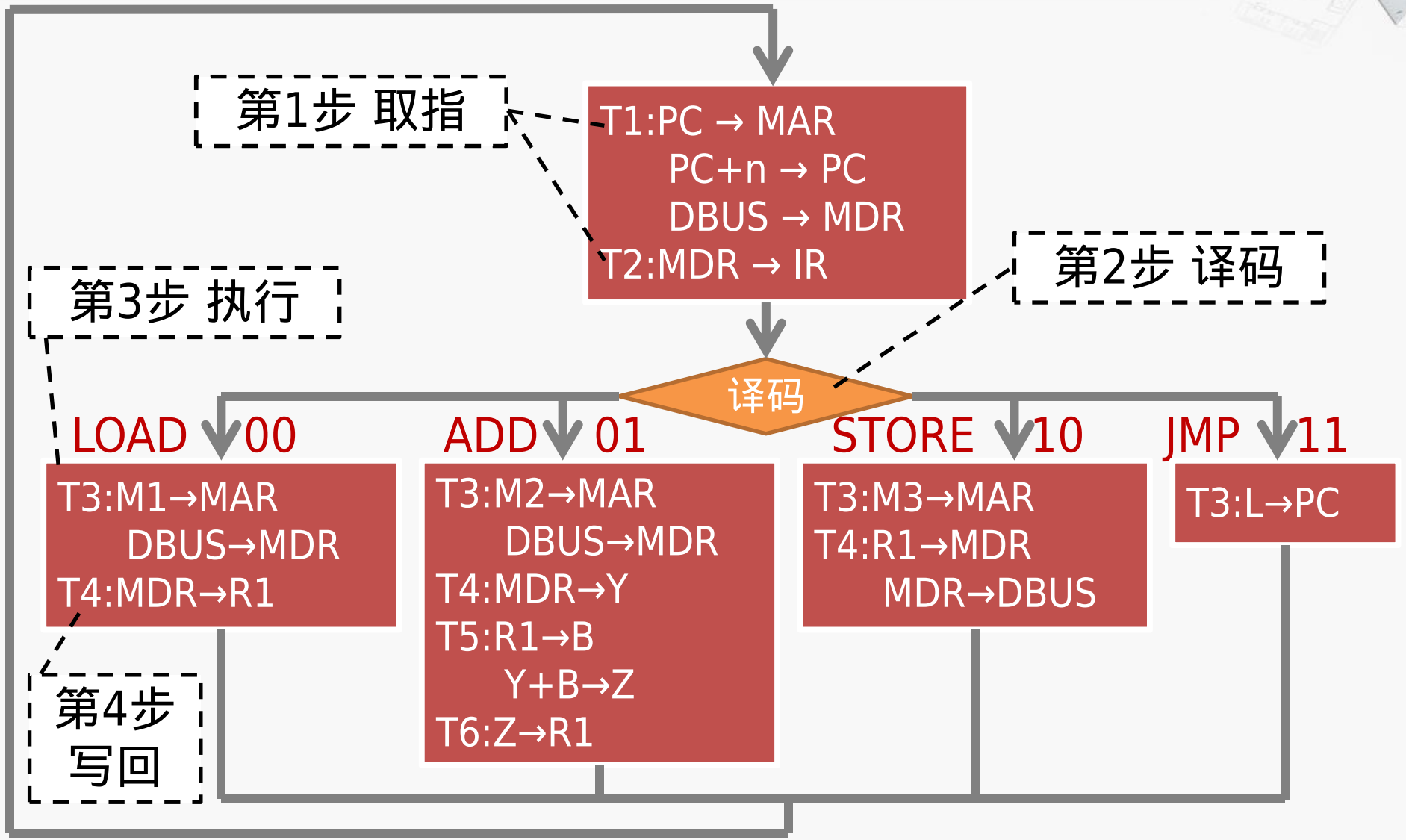
机器指令的读取与执行

2. 根据机器指令中的操作码，通过微地址形成电路，找到与该机器指令所对应的微程序入口地址
3. 逐条取出对应的微指令，每一条微指令提供一个微命令序列，控制相关部件的操作。执行完一条微指令后，根据微地址形成方法产生后继微地址，读取下一条微指令
4. 执行完对应于一条机器指令的一段微程序后，返回到“取机器指令用的微指令”，开始读取与执行又一条机器指令

一个简单的微程序实例

- 以某假想计算机指令系统的4条指令为例，可归纳出指令周期的流程图
  - LOAD R1, M1
    - 传送类指令，将M1中的内容装入R1
  - STORE M3, R1
    - 传送类指令，将R1的内容存入M3中
  - ADD R1, M2
    - 运算类指令，将R1的内容与M2中的内容相加存入R1
  - JMP L
    - 转移类指令，无条件转向L处

指令周期的流程图


第1步 取指
T1:PC → MAR
PC+n → PC
DBUS → MDR
T2:MDR → IR
第2步 译码
第3步 执行
译码
LOAD 00
ADD 01
STORE 10
JMP 11
T3:M1→MAR
DBUS→MDR
T4:MDR→R1
T3:M2→MAR
DBUS→MDR
T4:MDR→Y
T5:R1→B
Y+B→Z
T6:Z→R1
T3:M3→MAR
T4:R1→MDR
MDR→DBUS
T3:L→PC
第4步
写回

不同机器周期的控制信号

- 不同指令在不同机器周期内应发出的控制信号
  - （End代表指令执行结束的控制信号）

指令名	T1	T2	T3	T4	T5	T6	T7
LOAD (00)	PC_{OUT} MAR_{IN} $M_{读}$ $PC_{加}$	MDR_{OUT} IR_{IN}	IR_{OUT} MAR_{IN} $M_{读}$	MDR_{OUT} $R1_{IN}$	End		
ADD (01)	同上	同上	IR_{OUT} MAR_{IN} $M_{读}$	MDR_{OUT} Y_{IN}	$R1_{OUT}$ add	Z_{OUT} $R1_{IN}$	End
STORE (10)	同上	同上	IR_{OUT} MAR_{IN}	$R1_{OUT}$ MDR_{IN} $M_{写}$	End		
JMP (11)	同上	同上	IR_{OUT} PC_{IN}	End			

微指令形式的控制信号序列

- 本例中，控制器应发出的控制信号共有16个

- 对某一机器周期，微指令形式的信号序列的含义：
  - 需要发出的控制信号用1表示
  - 不需要发出的控制信号用0表示

微指令形式的控制信号序列：示例1

- 每条指令的T1周期，都要发出PC_{OUT}、MAR_{IN}、$M_{读}$和$PC_{加}$共四个信号

- 这条微指令形式的信号序列如下：

1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

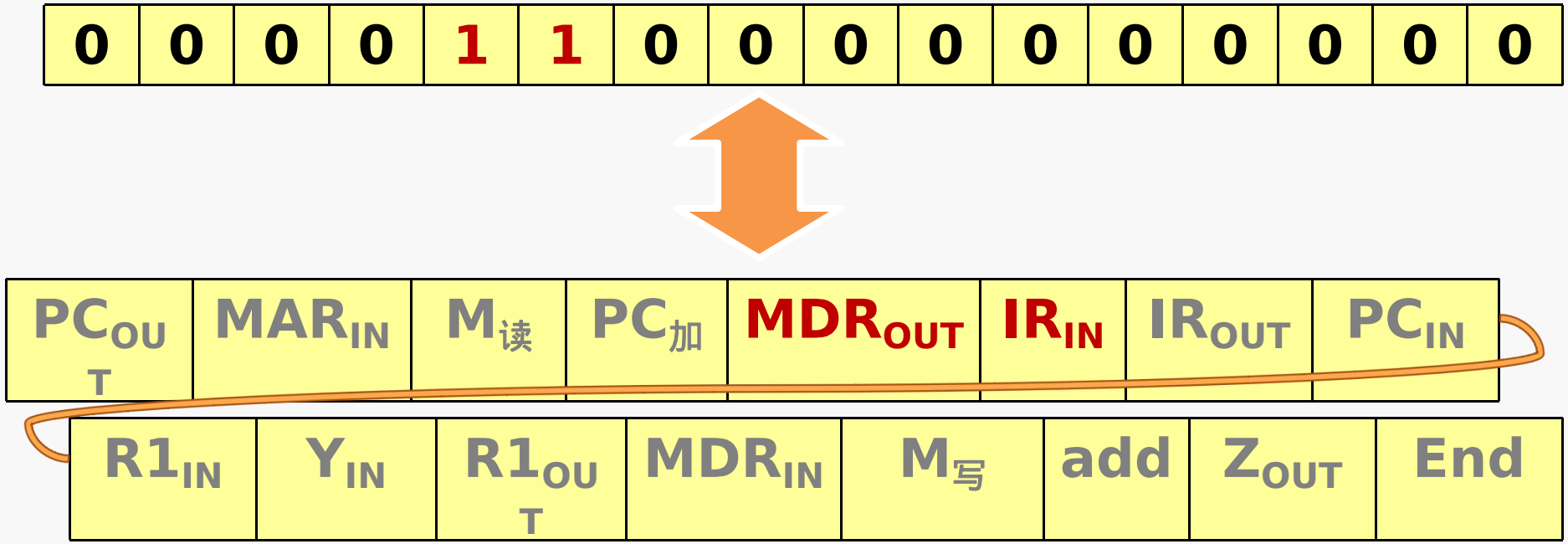
PC_{OUT}	MAR_{IN}	$M_{读}$	$PC_{加}$	MDR_{OUT}	IR_{IN}	IR_{OUT}	PC_{IN}

$R1_{IN}$	Y_{IN}	$R1_{OUT}$	MDR_{IN}	$M_{写}$	add	Z_{OUT}	End

微指令形式的控制信号序列：示例2

- 每条指令的T2周期，都要发MDR_{OUT}、IR_{IN}两个信号

- 这条微指令形式的信号序列如下：



控制信号序列组成微程序

- 按照控制信号的排列次序，参照指令周期的流程图，可以写出对每条指令在每一个机器周期中需要发出的控制信号序列

- 将这些信号序列再加上地址转移部分，即可构成微指令，并组成微程序，如后表所示

一个简单的微程序实例

微地址	指令	机器周期	微操作字段	下址字段
0000	取指令	T1	1111 0000 0000 0000	0 0001
0001		T2	0000 1100 0000 0000	1 xx10
0010	LOAD(00)	T3	0110 0010 0000 0000	0 0011
0011		T4	0000 1000 1000 0000	0 1001
0110	ADD(01)	T3	0110 0010 0000 0000	0 0100
0100		T4	0000 1000 0100 0000	0 0101
0101		T5	0000 0000 0010 0100	0 0111
0111		T6	0000 0000 1000 0010	0 1001
1010	STORE(10)	T3	0100 0010 0000 0000	0 1000
1000		T4	0000 0000 0011 1000	0 1001
1110	JMP(11)	T3	0000 0011 0000 0000	0 1001
1001	End	Tn	0000 0000 0000 0001	0 0000

示例：ADD指令的执行过程（1）

- 控制器首先取出0000地址处的第一条微指令执行，发出了4个微命令

微地址	微操作字段	下址字段
0000	1111 0000 0000 0000	0 0001
0001	0000 1100 0000 0000	1 xx10

- 该微指令的“下址字段”为0 0001，表示不发生分支，下一条微指令的地址为0001

示例：ADD指令的执行过程（2）

- 控制器在0001地址处取出第二条微指令执行，发出了两个微命令

微地址	微操作字段	下址字段
0000	1111 0000 0000 0000	0 0001
0001	0000 1100 0000 0000	1 xx10

- 至此，取ADD指令的操作已经完成，译码后知该指令的操作码为01

示例：ADD指令的执行过程（3）

- 第二条微指令的“下址字段”为1 xx10，最高位的1表示要发生分支，xx为该指令的操作码。因此，下一条微指令在0110地址处

微地址	微操作字段	下址字段
0000	1111 0000 0000 0000	0 0001
0001	0000 1100 0000 0000	1 xx10
0010	LOAD(00)	
0110	ADD(01)	
1010	STORE(10)	
1110	JMP(11)	

示例：ADD指令的执行过程（4）

- 继续执行，直至在1001地址处取出最后一条微指令执行，该微指令发出指令结束的微命令End，并转向0000地址处开始取下一条指令，并执行该指令所对应的微程序

微地址	微操作字段	下址字段
0110	0110 0010 0000 0000	0 0100
0100	0000 1000 0100 0000	0 0101
0101	0000 0000 0010	0 0111
1001	0000 0000 0000 0001	0 0000
	0000 0000 1000	0 1001

微程序实例的工作过程

Store 指令
修改、增加功能，如何修改指令
硬布线如何产生对应的控制信号、电路

微地址	指令	机器周期	微操作字段	下址字段
0000	取指令	T1	1111 0000 0000 0000	0 0001
0001		T2	0000 1100 0000 0000	1 xx10
0010	LOAD(00)	T3	0110 0010 0000 0000	0 0011
0011		T4	0000 1000 1000 0000	0 1001
0110	ADD(01)	T3	0110 0010 0000 0000	0 0100
0100		T4	0000 1000 0100 0000	0 0101
0101		T5	0000 0000 0010 0100	0 0111
0111		T6	0000 0000 1000 0010	0 1001
1010	STORE(10)	T3	0100 0010 0000 0000	0 1000
1000		T4	0000 0000 0011 1000	0 1001
1110	JMP(11)	T3	0000 0011 0000 0000	0 1001
1001	End	Tn	0000 0000 0000 0001	0 0000

微程序控制器的主要特点

- 与硬连线控制器相比，微程序控制器是一种利用软件方法来设计硬件的技术，可实现复杂指令的操作控制

- **缺点：速度较慢**
  - 每执行一条指令都要启动控制存储器中的一串微指令（即一段微程序），**执行速度相对于“硬布线控制器” 要慢**

微程序控制器的主要特点

优点1：规整性

- 用程序的方法来产生和组织微命令信号（将程序技术引入CPU）
- 用存储逻辑控制代替组合逻辑控制（将存储逻辑引入CPU）

优点2：灵活性

- 可以较方便地增加和修改指令，只要增加或修改一部分微程序即可

本讲到此结束，谢谢

欢迎继续学习本课程

计算机组织与体系结构 Computer Architectures

主讲：陆俊林

北京大学 Peking University