计算机组织与体系结构

第十一讲

计算机科学与技术学院

舒燕君

Recap

- 浮点四则运算
  - ✓ 浮点加减法（溢出判断）
  - ✓ 浮点乘除法（阶码运算、尾数运算、规格化）
- 算逻运算部件
  - ✓ 四位**ALU**（**74181**）
- 多级时序系统（**x86**）
  - ✓ 指令周期（取指、间指、执行、中断）
  - ✓ 微操作命令分析
  - ✓ 控制单元的功能（外特性、控制信号）
  - ✓多级时序系统 （指令周期、工作周期、时钟周期、机器周期）

第5章　CPU设计与实现

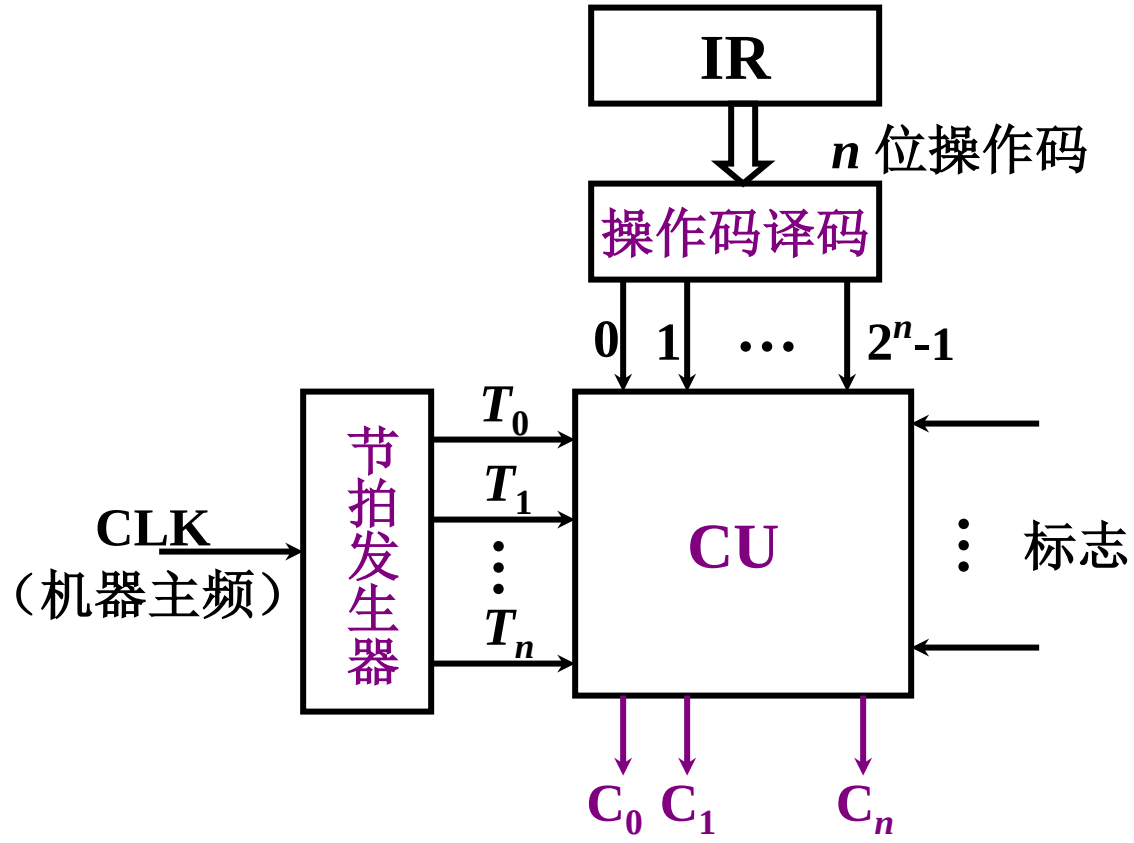
5.1 CPU 的结构

5.2 运算方法与ALU

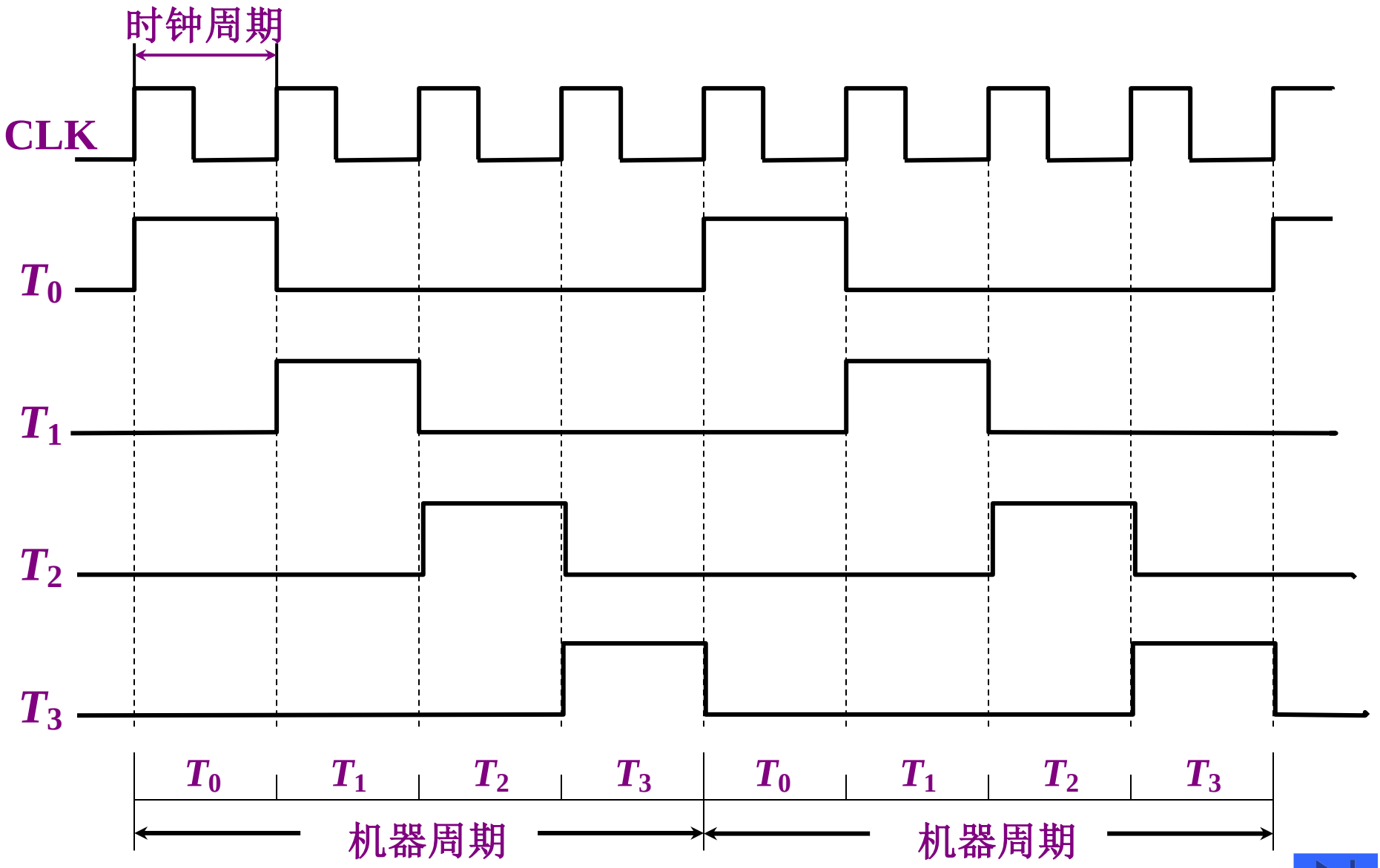
5.3 多级时序系统（X86）

5.4 MIPS CPU的简单实现

控制单元



节拍信号

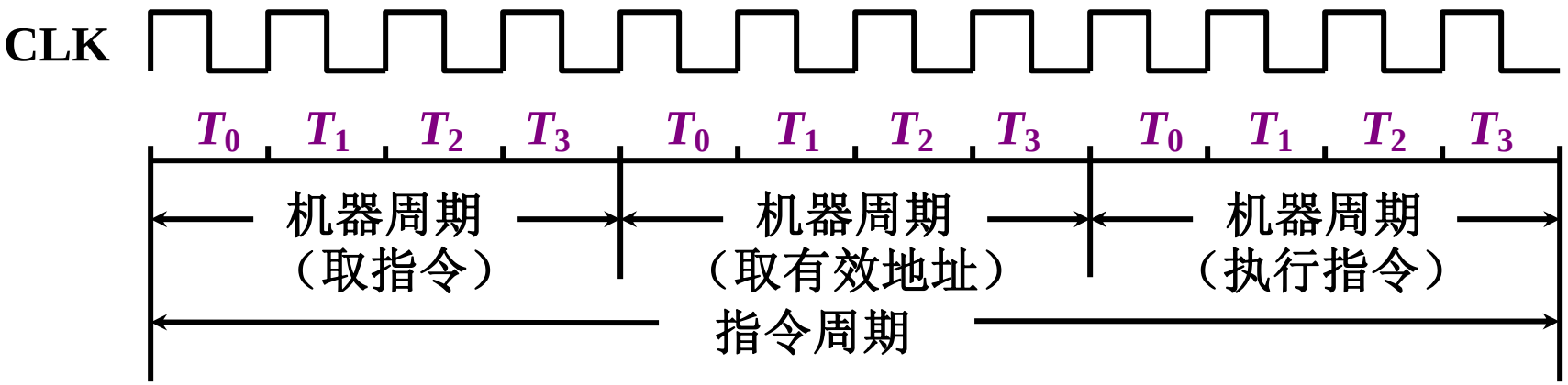


四、控制方式

产生不同微操作命令序列所用的时序控制方式

1. 同步控制方式

任一微操作均由 统一基准时标 的时序信号控制



(1) 采用 定长 的机器周期

以 最长 的 微操作序列 和 最繁 的微操作作为 标准

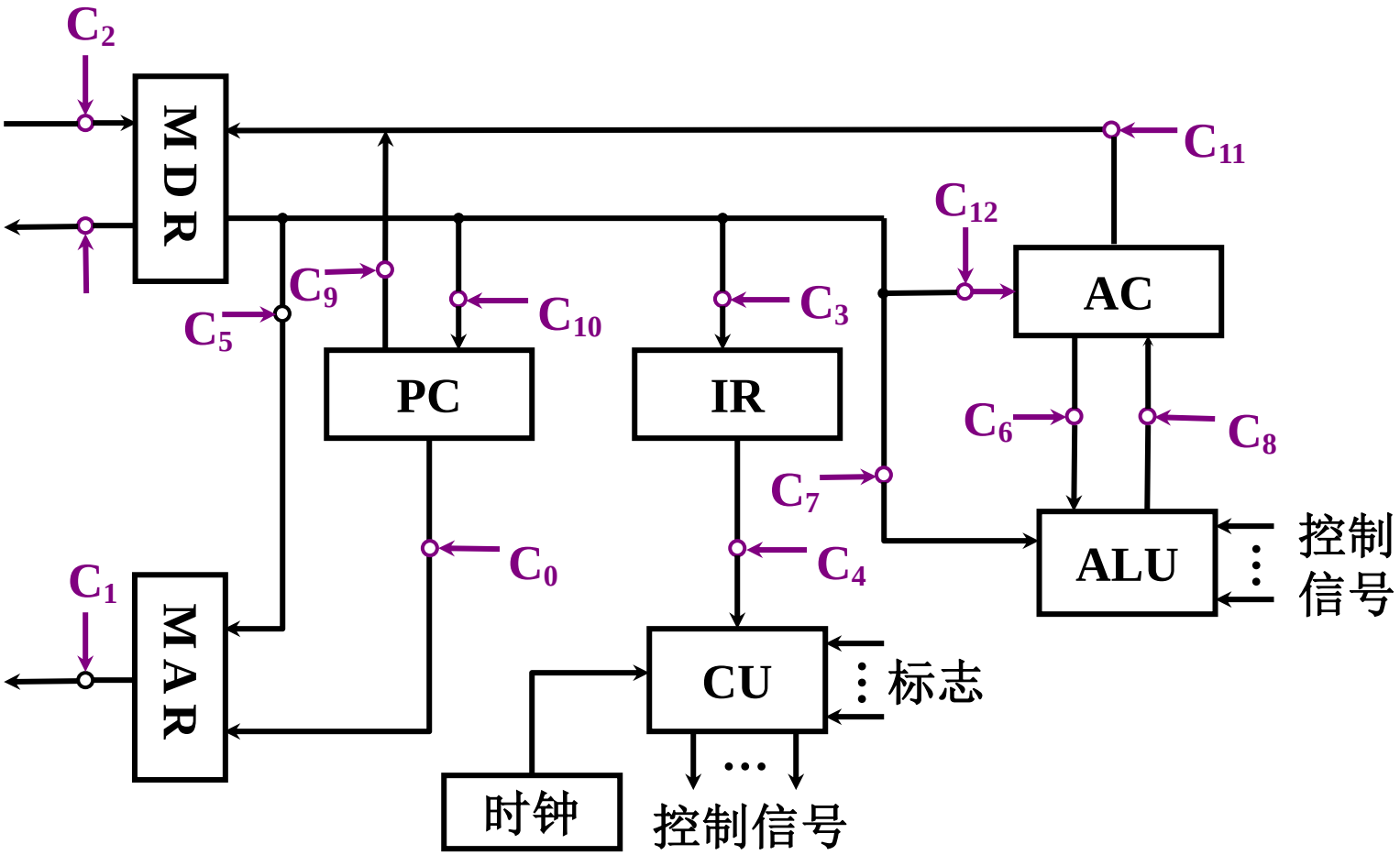
机器周期内 节拍数相同

微操作的节拍安排

采用 同步控制方式

一个 机器周期 内有 3个节拍（时钟周期）

CPU 内部结构采用非总线方式



取指周期 微操作的 节拍安排

T_0 PC ⟶ MAR

1 ⟶ R

T_1 M (MAR) ⟶ MDR

(PC) + 1 ⟶ PC

T_2 MDR ⟶ IR

OP (IR) ⟶ ID

间址周期 微操作的 节拍安排

T_0 Ad (IR) ⟶ MAR

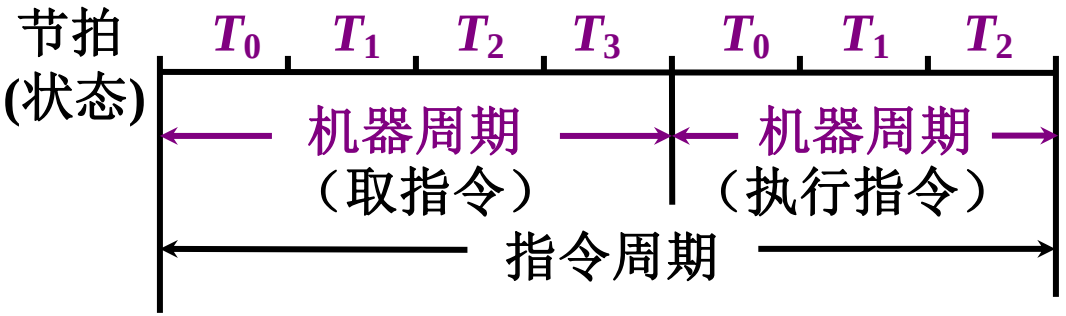
1 ⟶ R

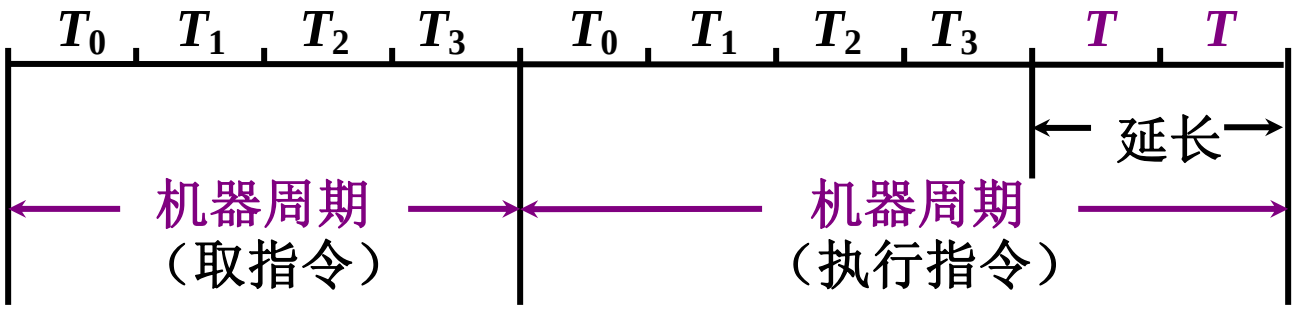
T_1 M (MAR) ⟶ MDR

T_2 MDR ⟶ Ad (IR)

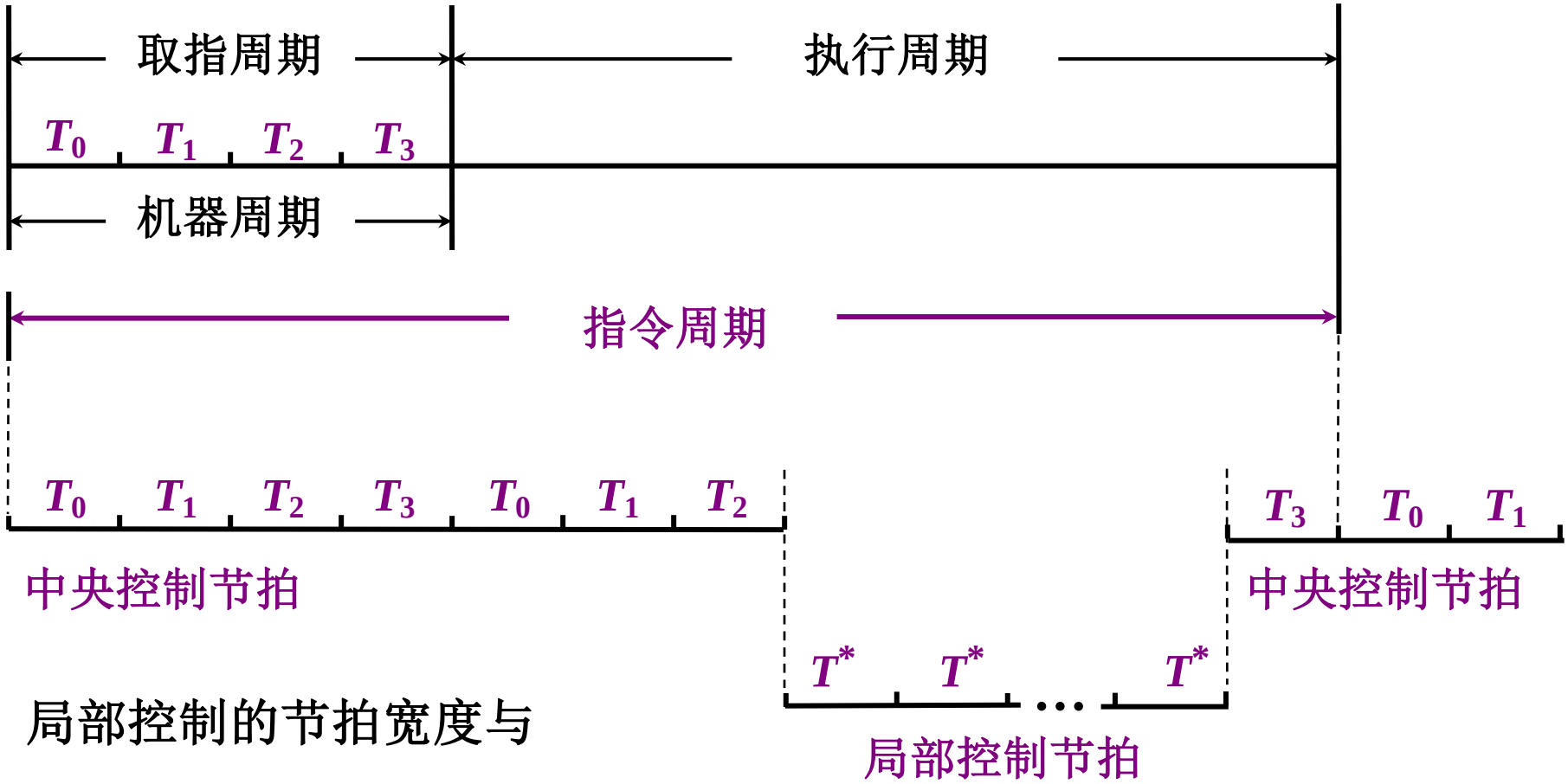
(2) 采用不定长的机器周期

机器周期内 节拍数不等





(3) 采用中央控制和局部控制相结合的方法



局部控制的节拍宽度与

中央控制的节拍宽度一致

2. 异步控制方式

无基准时标信号

无固定的周期节拍和严格的时钟同步

采用 应答方式

3. 联合控制方式

同步与异步相结合

4. 人工控制方式

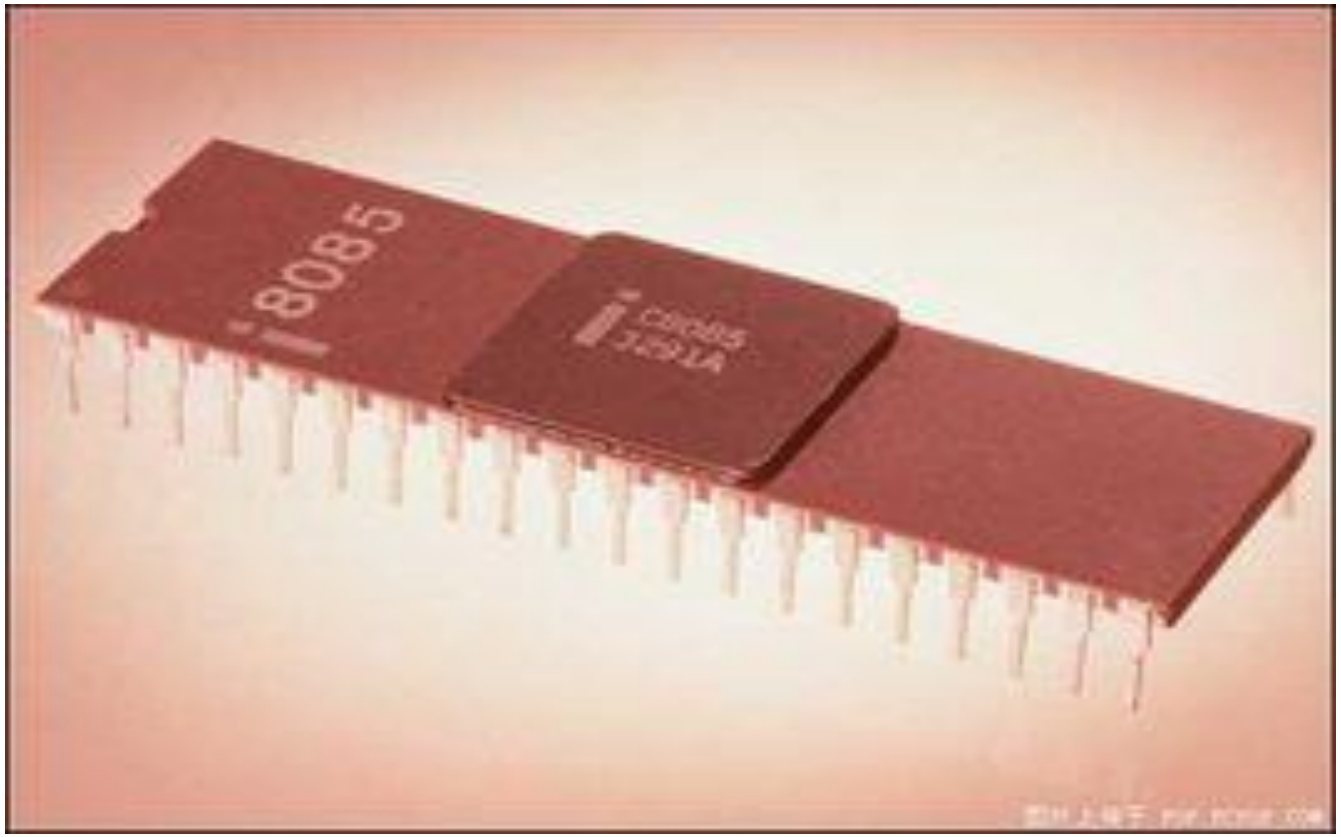
(1) Reset

(2) 连续 和 单条 指令执行转换开关

(3) 符合停机开关

五、多级时序系统实例分析

1. 8085 的组成



2. 8085 的外部引脚

(1) 地址和数据信号

A_{15}~A_8　AD_7~AD_0

SID　　SOD

(2) 定时和控制信号

入　X_1　X_2

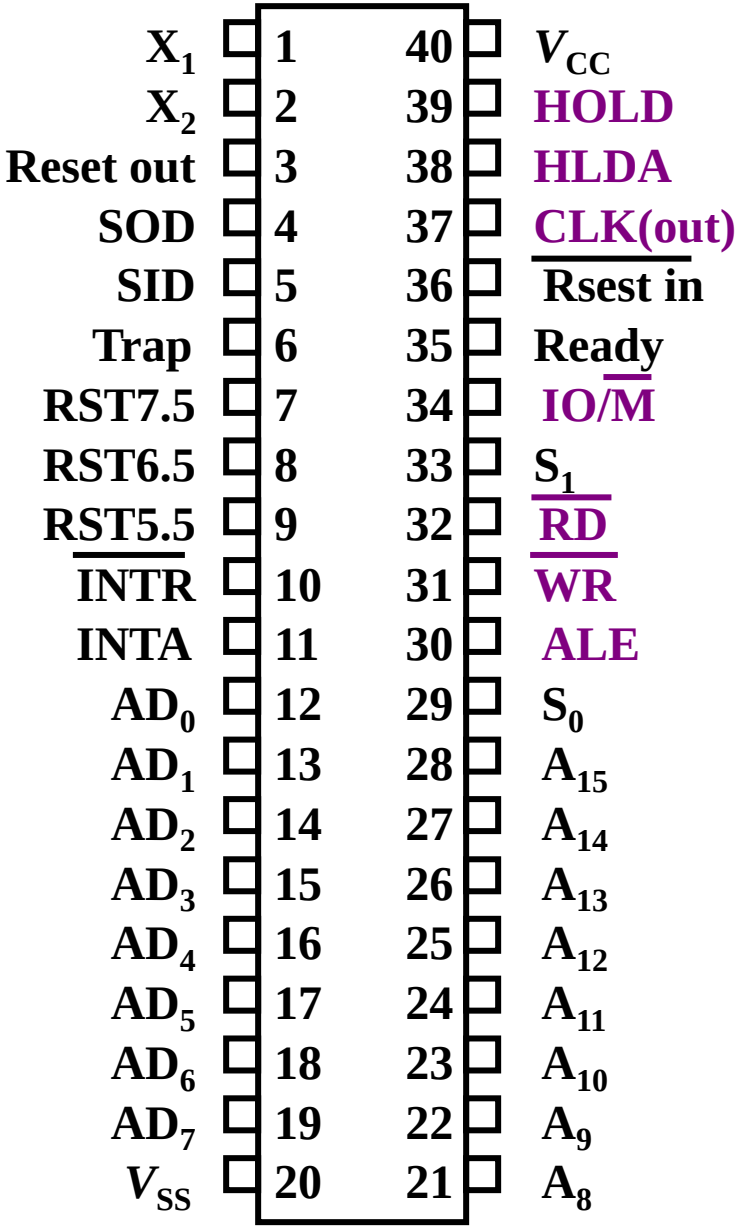
出　CLK　ALE　S_0　S_1

　　$\overline{IO/M}$　$\overline{RD}$　$\overline{WR}$

(3) 存储器和 I/O 初始化

入　HOLD　Ready

出　HLDA



(4) 与中断有关的信号

入　$\overline{\text{INTR}}$

出　INTA

Trap 重新启动中断

(5) CPU 初始化

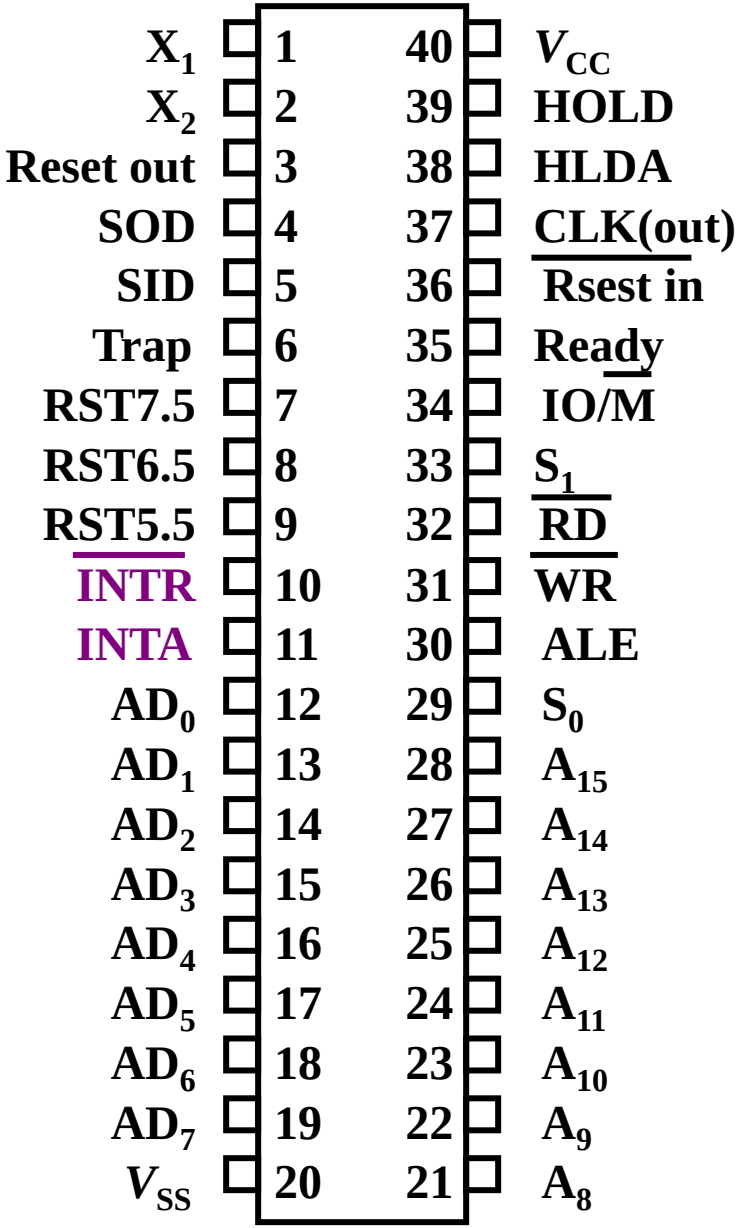
入　$\overline{\text{Reset in}}$

出　Reset out

(6) 电源和地

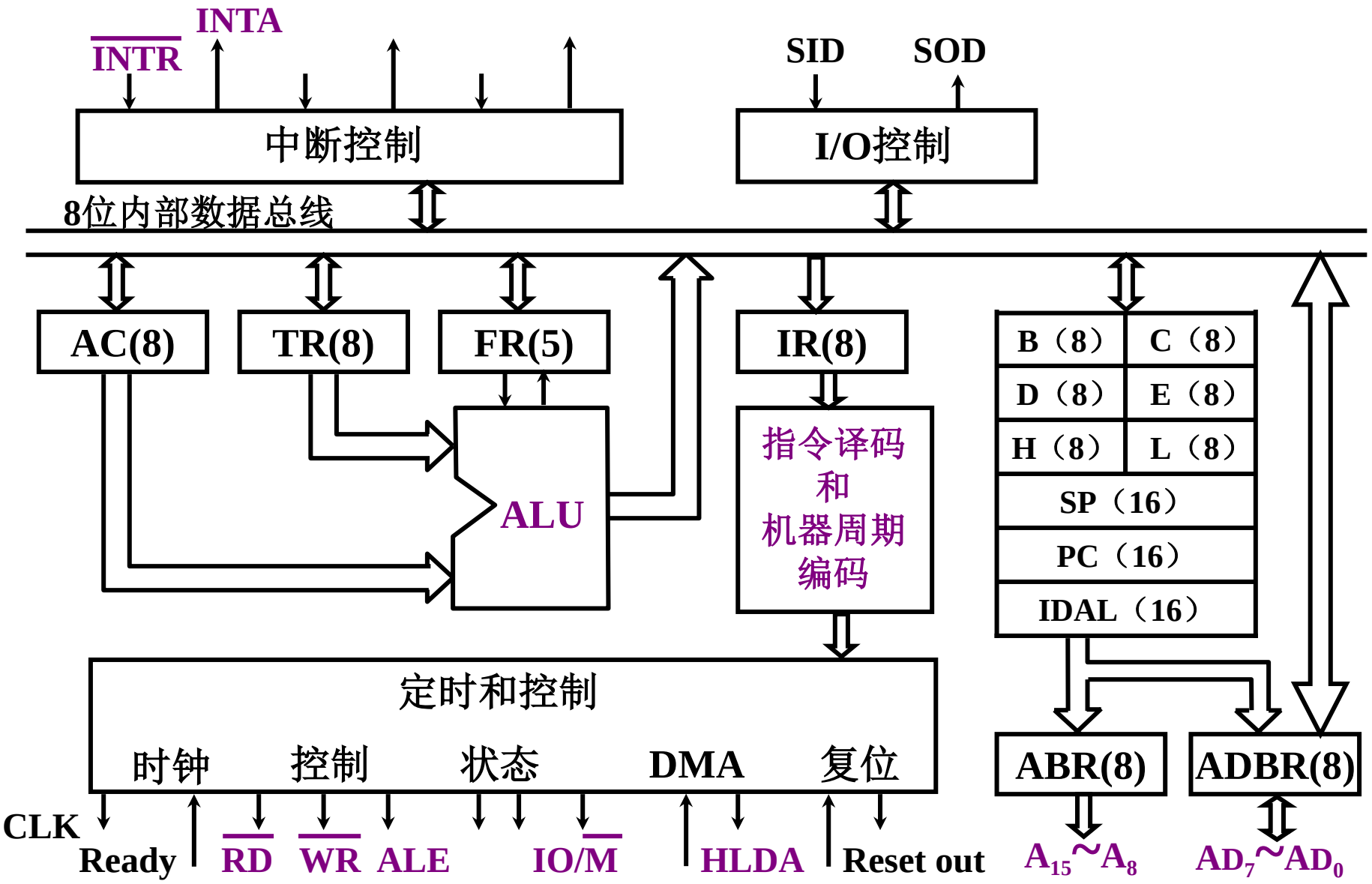
V_{CC}　+5 V

V_{SS}　地

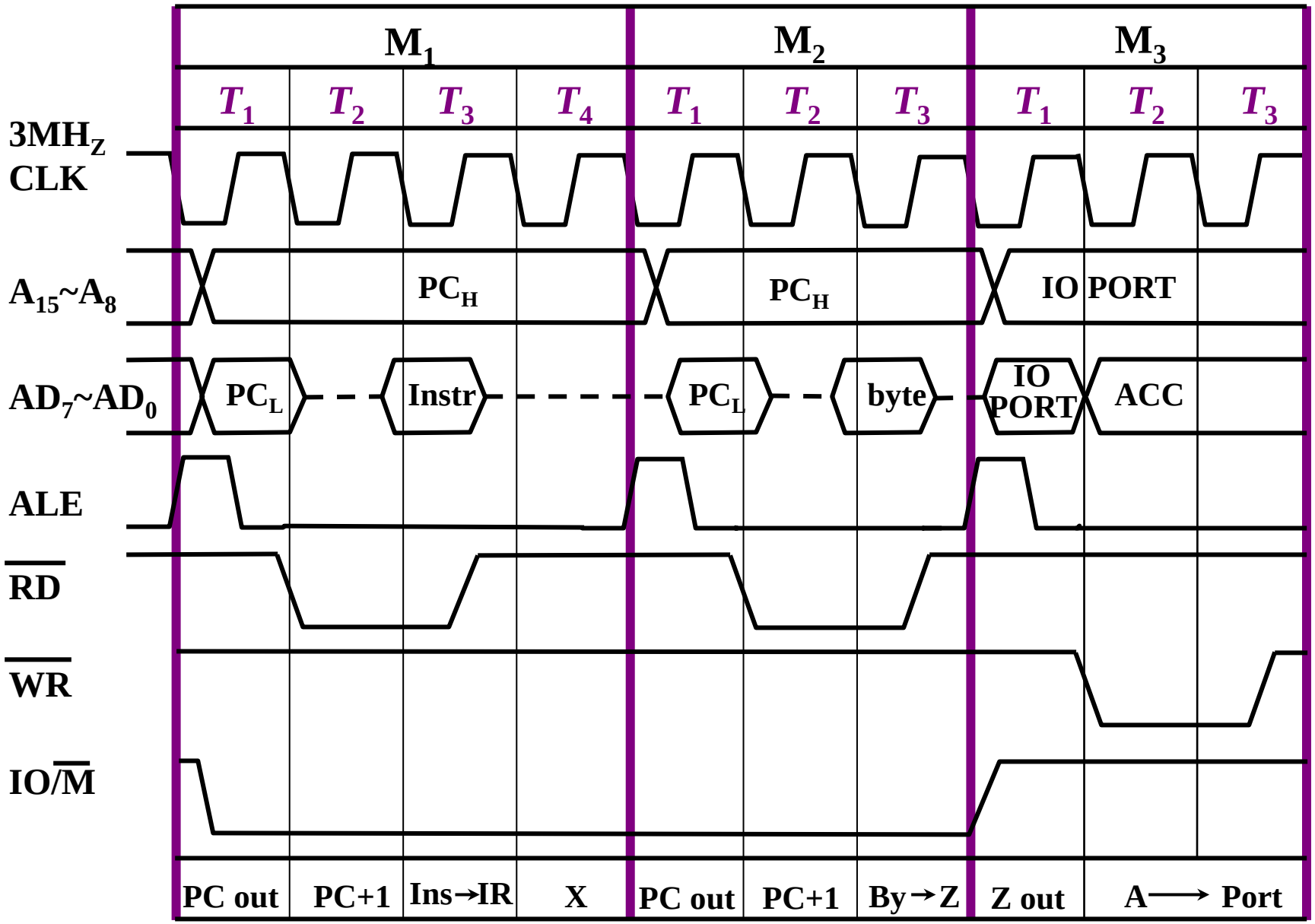


五、多级时序系统实例分析

3. 8085 的组成



4. 机器周期和节拍（状态）与控制信号的关系



小结

以一条输出指令（I/O 写）为例

机器周期 M_1　取指令操作码

机器周期 M_2　取设备地址

机器周期 M_3　执行 ACC 的内容写入设备

每个 控制 信号在 指定机器周期 的

指定节拍 *T* 时刻 发出

第5章　CPU设计与实现

5.1 CPU 的结构

5.2 运算方法与ALU

5.3 多级时序系统（X86）

5.4 MIPS CPU的简单实现

5.4 MIPS的一种简单实现

- 处理器的指令字长为32位，包含
  - 32个32位通用寄存器R0~R31（R0值为0）；
  - 1个32位的指令寄存器IR；
  - 1个32位的程序计数器PC

- 取指令时，可以直接从指令存储器中提取32位的指令信息。取数据时，与数据存储器进行32位的数据交换。处理器的地址总线宽度是32位，数据总线宽度也是32位，无论是取指还是数据访问，都假设能在一个时钟周期内完成。

- 只讨论整数指令的实现（包括：Load和Store，等于0转移，整数ALU指令等。）

处理器功能及指令系统定义

- 存数指令 SW 30(R2), R1
- 取数指令 LW R1, 30(R2)

- 逻辑与指令 AND R1, R2, R3
- 逻辑或指令 OR R1, R2, R3

- 立即数加法指令 ADDI R4, R5, #6
- 立即数减法指令 SUBI R4, R5, #6

- 条件转移（零则转）指令 BEQZ R3, x
- 无条件转移指令 J x

给定的指令系统

J x	PC ← PC+x
BEQZ R3, x	If(R3=0) then PC ← PC+x
AND R1, R2, R3	Regs[R1] ←Regs[R2] and Regs[R3]
OR R1, R2, R3	Regs[R1] ←Regs[R2] or Regs[R3]
ADDI R4, R5, #6	Regs[R4] ←Regs[R5] + 6
SUBI R4, R5, #6	Regs[R4] ←Regs[R5] - 6
LW R1, 30(R2)	Regs[R1] ← Mem[30+Regs[R2]]
SW 30(R2), R1	Mem[30+Regs[R2]]← Regs[R1]

操作码

操作码占据了6位，最多可支持64种指令的设计。目前的指令系统仅包含了8种操作，下表定义这8种操作的操作码

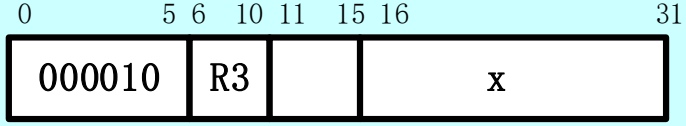
指令名称	助记符	二进制操作码
无条件跳转	J	000000
条件跳转	BEQZ	000010
逻辑与操作	AND	000100
逻辑或操作	OR	000100
立即数减法	SUBI	001000
立即数加法	ADDI	001010
存数操作	SW	001100
取数操作	LW	001110

每条指令的格式描述

J name

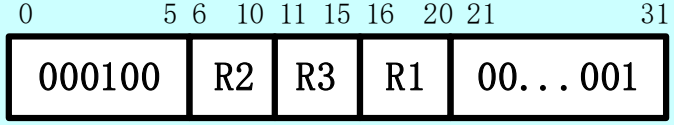
PC←PC+x

BEQZ R3,X



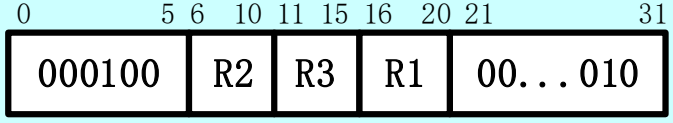
if R3=0 then PC←PC+x

AND R1,R2,R3



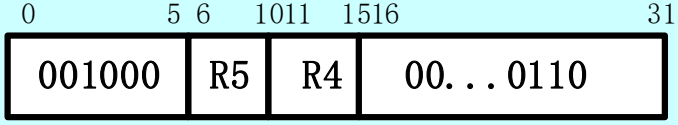
R1←R2 and R3

OR R1,R2,R3



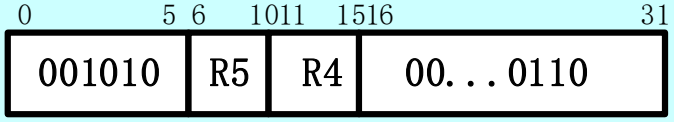
R1←R2 or R3

SUBI R4,R5,#6



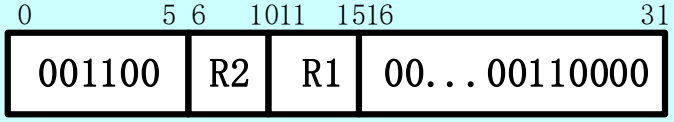
R4←R5-6

ADDI R4,R5,#6



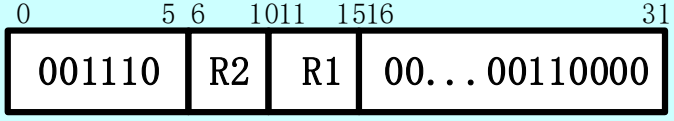
R4←R5+6

SW 30(R2),R1



Mem[R2+30]←R1

LW R1,30(R2)



R1←Mem[R2+30]

指令格式举例

- 38410030

  00111000010000010000000000110000

  LW R1,30(R2)

- 10430801

  00010000010000110000100000000001

  AND R1,R2,R3

5.4 MIPS的一种简单实现

实现MIPS指令的一种简单数据通路

- 将指令执行划分为5个阶段
  - 取指令周期
  - 指令译码/读寄存器周期
  - 执行/有效地址计算周期
  - 存储器访问/分支完成周期
  - 写回周期

5.4 MIPS的一种简单实现

1.取指令周期(IF)

操作为：

根据PC值从存储器中取出指令，并将指令送入指令寄存器IR；PC值增加4，指向顺序的下一条指令，并将下一条指令的地址放入临时寄存器NPC中。

IR←IMem[PC]

NPC←PC+4

图示

5.4 MIPS的一种简单实现

2.指令译码/读寄存器周期(ID)

操作为：

进行指令译码，读IR寄存器（指令寄存器），按照相应寄存器号读寄存器文件，并将读出结果放入两个临时寄存器A和B中。同时对IR寄存器中内容的低16位进行符号扩展，然后将符号扩展之后的32位立即值保存在临时寄存器Imm中。

$$A \leftarrow Regs[IR_{6..10}]$$

$$B \leftarrow Regs[IR_{11..15}]$$

$$Imm \leftarrow ((IR_{16})^{16} \#\# IR_{16..31})$$

MIPS的固定字段译码技术:图示

5.4 MIPS的一种简单实现

3.执行/有效地址计算周期(EX)

操作为：

存储器访问：	**ALUoutput←A+Imm**
寄存器-寄存器ALU：	**ALUoutput←A func B**
寄存器-立即值ALU:	**ALUoutput←A op Imm**
分支操作：	**ALUoutput←NPC+Imm**
	Cond←(A op 0)

问题：为什么执行和有效地址计算可以合并？

5.4 MIPS的一种简单实现

4.访存/分支操作完成周期(MEM)

操作为：

访存操作：

Load： **LMD←DMem[ALUoutput]**

Store： **DMem[ALUoutput]←B**

分支操作： **if (Cond) PC←ALUoutput**

else PC←NPC

5.4 MIPS的一种简单实现

5.写回周期(WB)

操作为：

- 寄存器-寄存器型ALU指令：

  $Reg[IR_{16..20}] \leftarrow ALUoutput$

- 寄存器-立即值型ALU指令：

  $Reg[IR_{11..15}] \leftarrow ALUoutput$

- Load指令：

  $Reg[IR_{11..15}] \leftarrow LMD$

5.4 MIPS的一种简单实现

6.性能分析

在该数据通路上，

分支指令需要4个时钟周期

其它指令需要5个时钟周期

假设分支指令占总指令数的12%，问CPI=?

$$CPI=4\times12\%+5\times(1-12\%)=4.88$$

结论：就性能和硬件开销而言，上述实现不是一种优化实现！

5.4 MIPS的一种简单实现

7.改进方法

(1)在Mem周期完成ALU指令

假设ALU指令数占指令总数的44%，则在时钟周期时间不变的同时，CPI可以降低至4.44

(2)如要进一步降低CPI，可能需要延长时钟周期时间，使每个时钟周期中能够完成更多的工作

(3)采用单周期实现，可以将CPI降低为1，但时钟周期时间却会增加为原来的5倍

一般不采用这种方法，为什么？

- 流水技术

本章小结

1. **CPU的主要结构包括：CU、ALU、寄存器、中断系统。**

2. **数据在计算机中的运算方法和硬件构成。**

3. **x86架构的多级时序系统。**

4. **MIPS架构下CPU的一种简单实现方式。**

第五章作业

唐朔飞教材

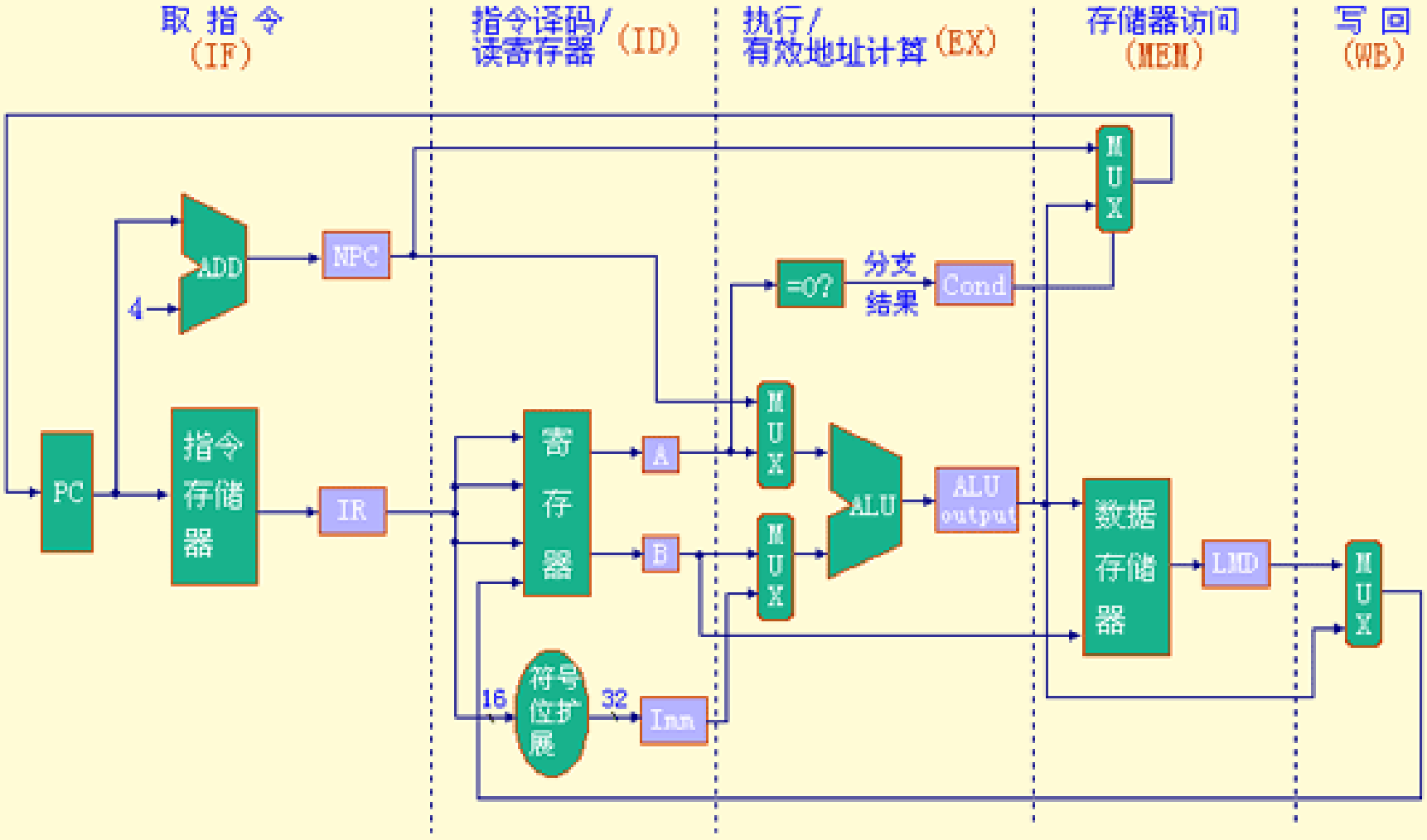
1. P290： T4、T9、T19（1、3）、T20（1、2，原码一位乘）、T27（1、3）。

2. P370：T1、T2。

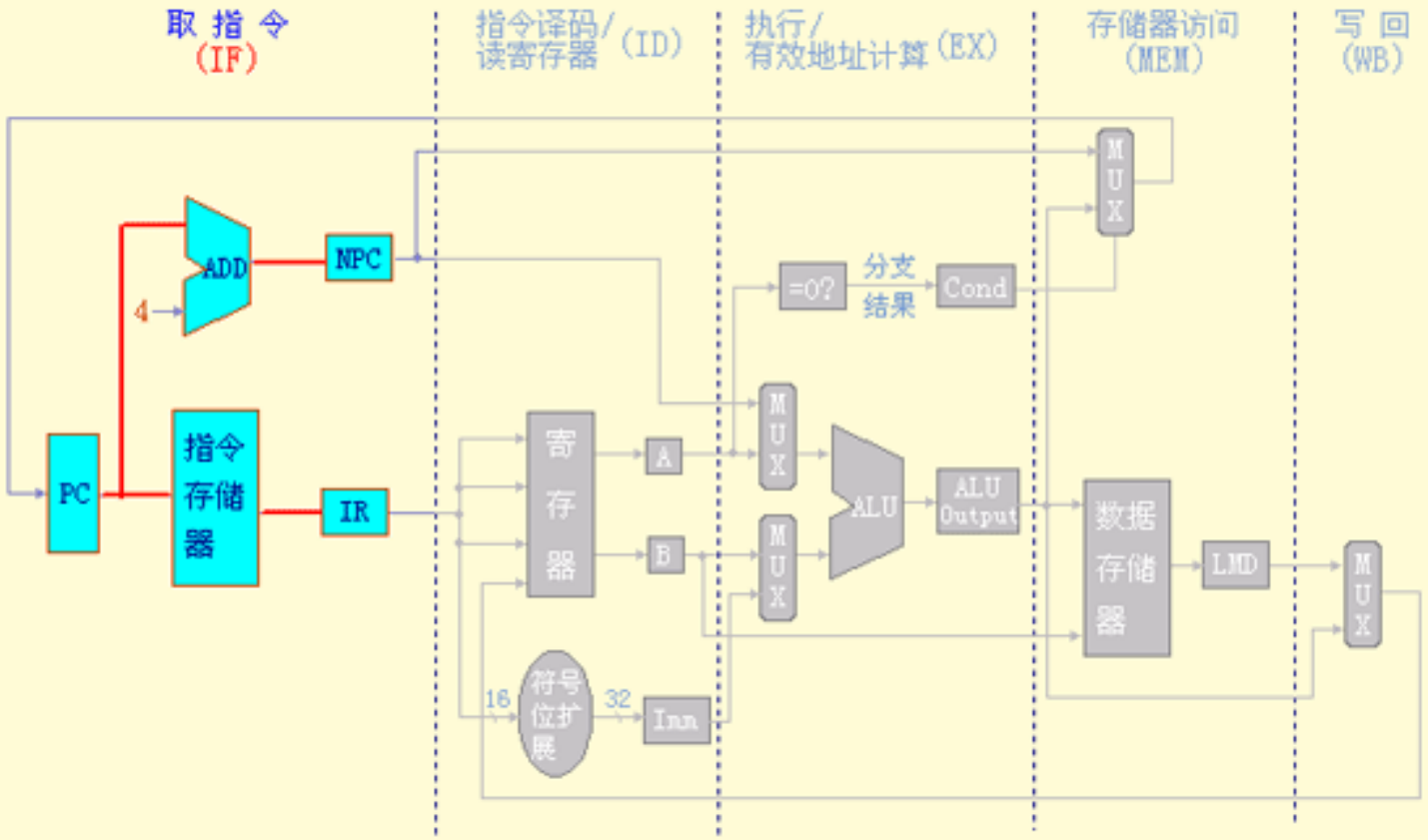
3. P393：T3、T5。

第四、五章作业**10**月**27**日交到综合楼**514**

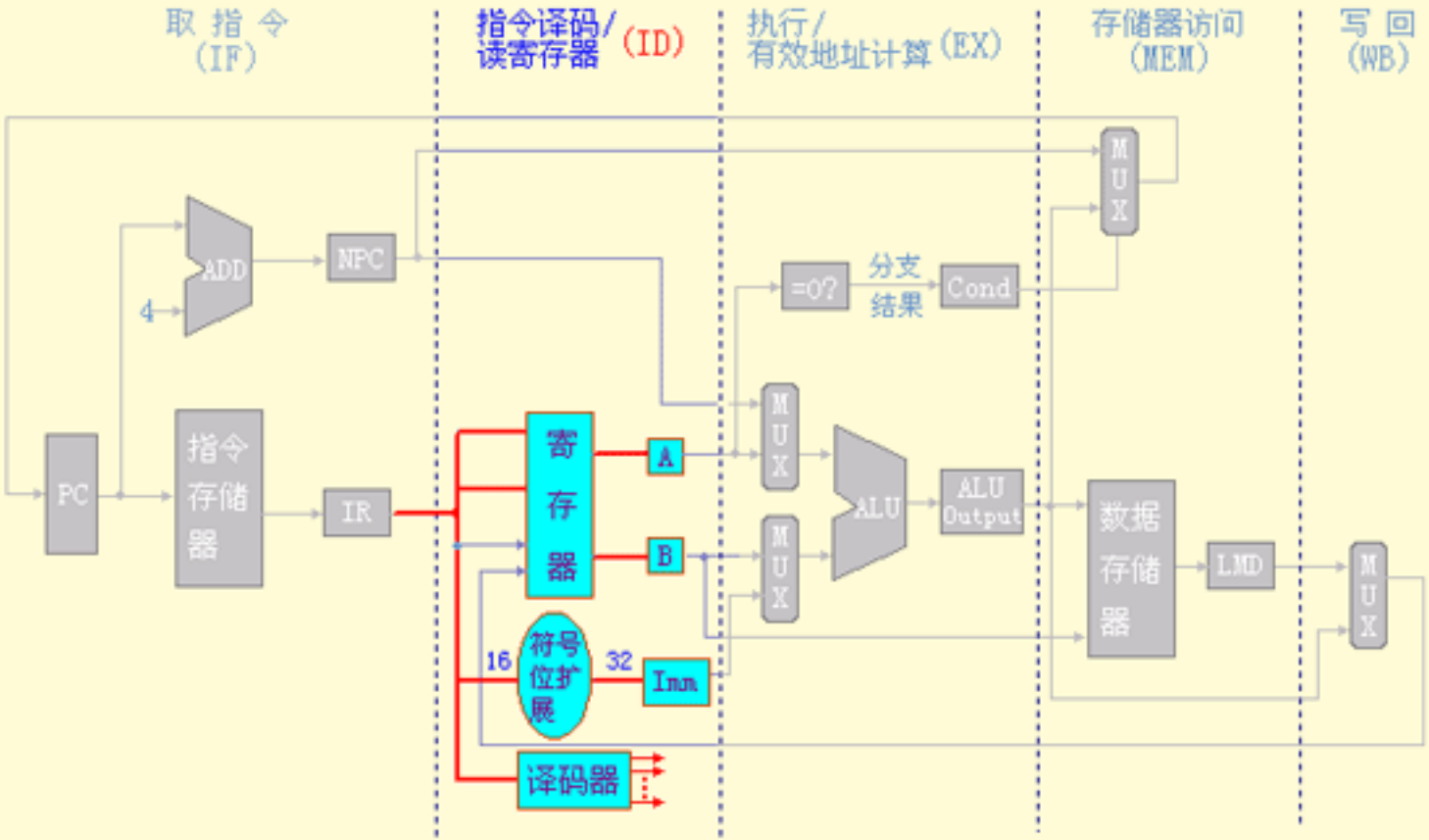
实现DLX指令的一种简单数据通路



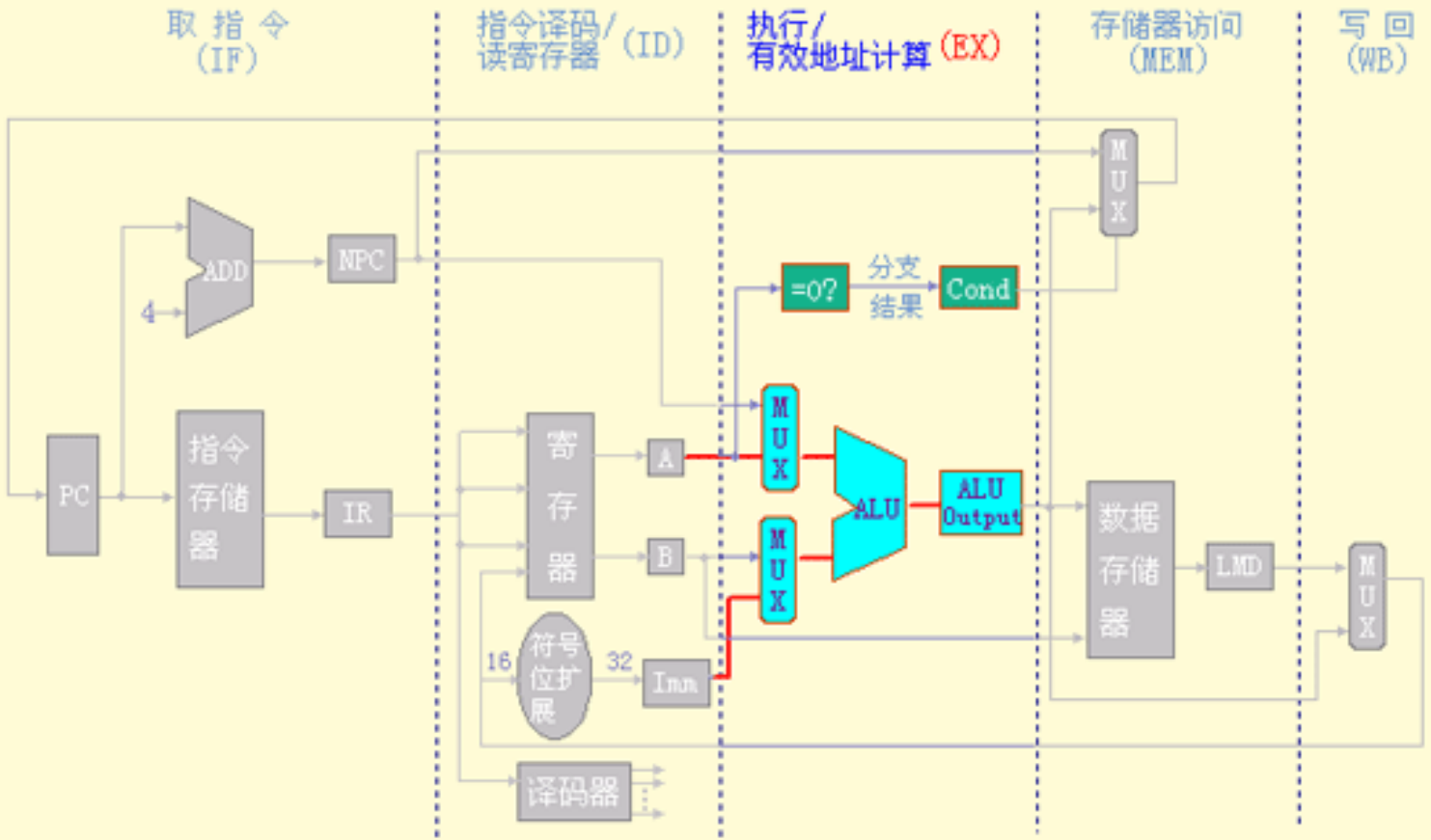
取指令周期的操作


取 指 令
(IF)
指令译码/
读寄存器 (ID)
执行/
有效地址计算 (EX)
存储器访问
(MEM)
写 回
(WB)
ADD
4
NPC
PC
指令
存储
器
IR
寄
存
器
A
B
16
符号
位扩
展
32
Imm
=0?
分支
结果
Cond
MUX
MUX
MUX
ALU
ALU
Output
数据
存储
器
LMD
MUX

指令译码/读寄存器周期的操作

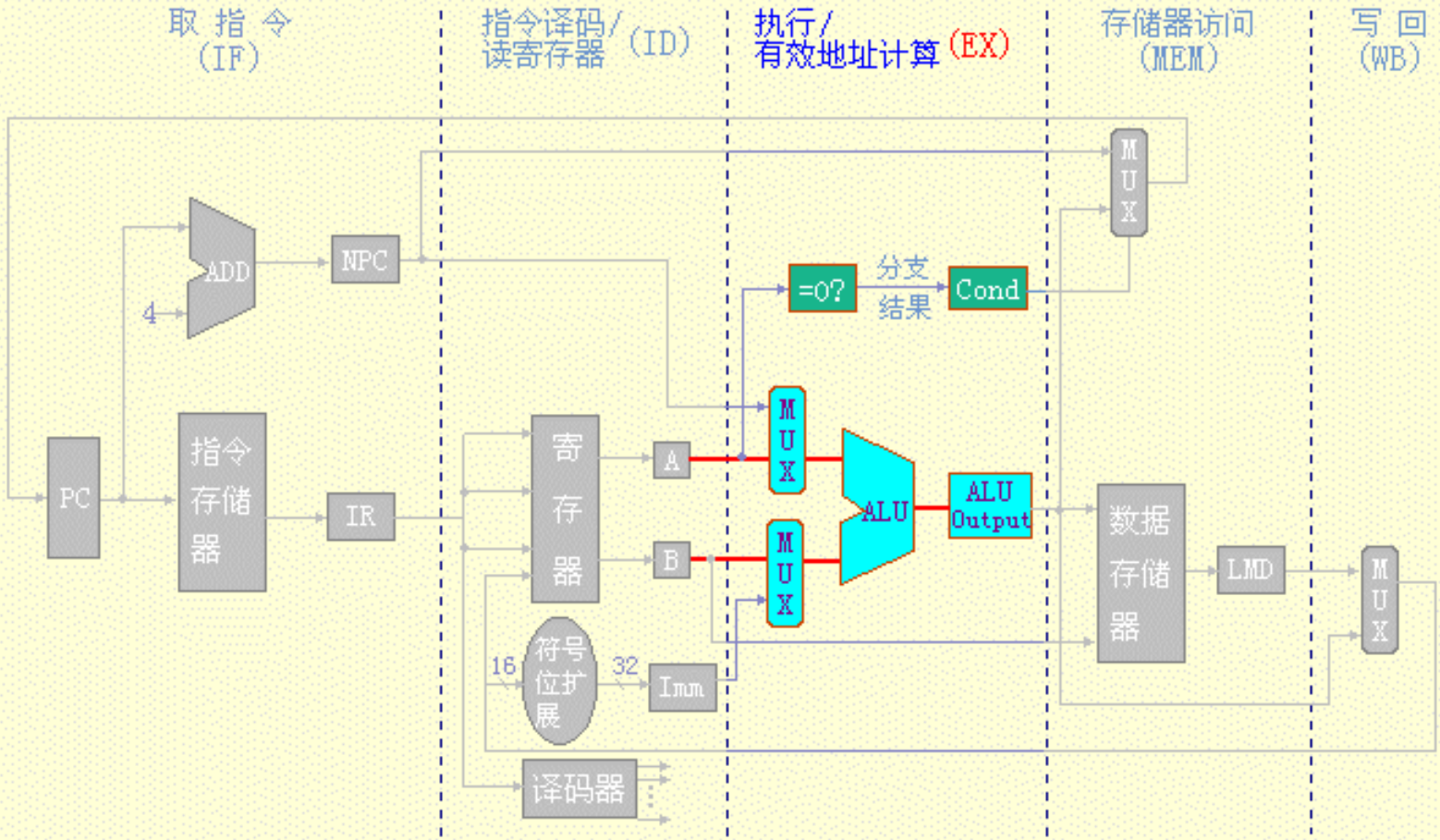


执行／有效地址计算周期的操作
（存储器访问指令）

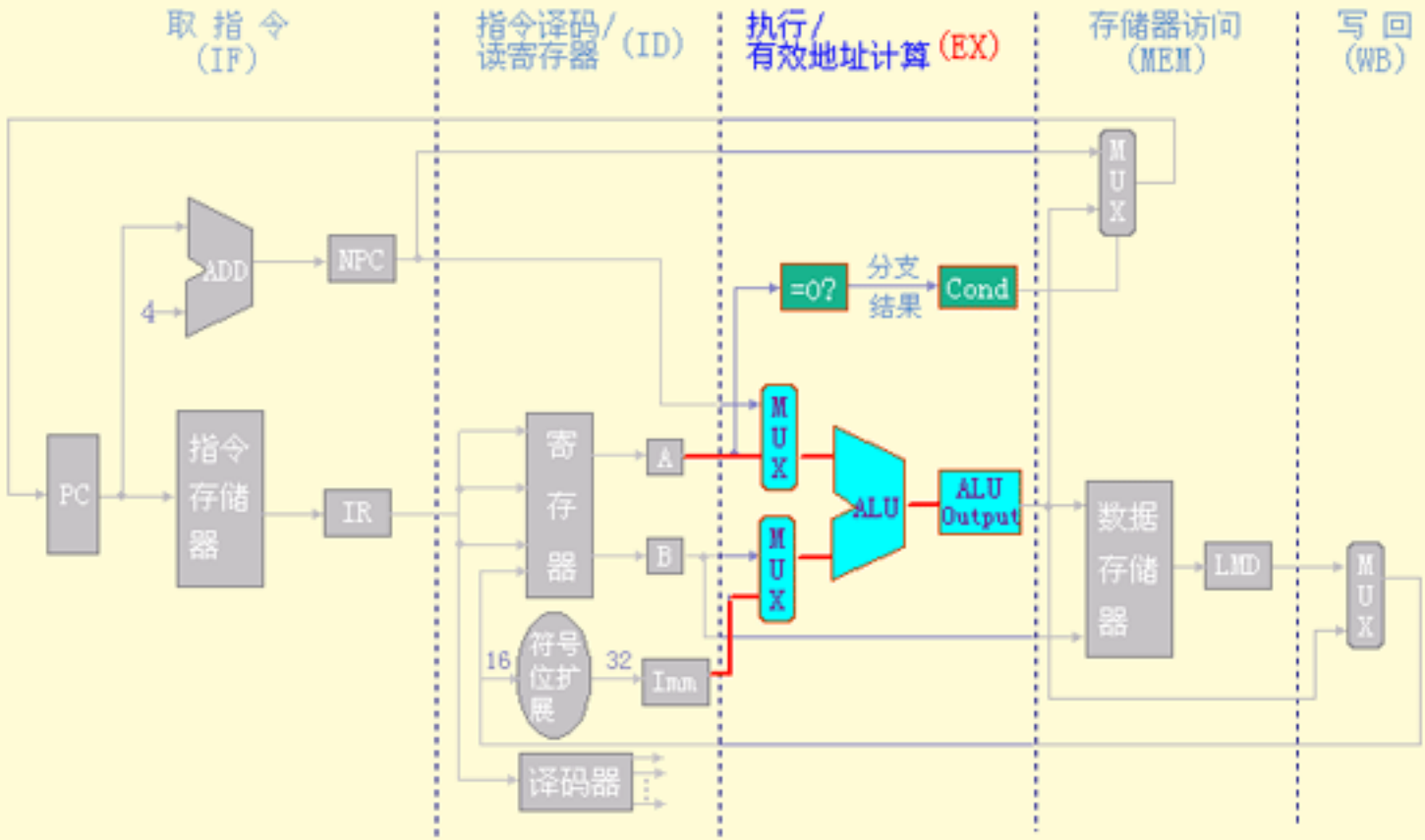


执行／有效地址计算周期的操作

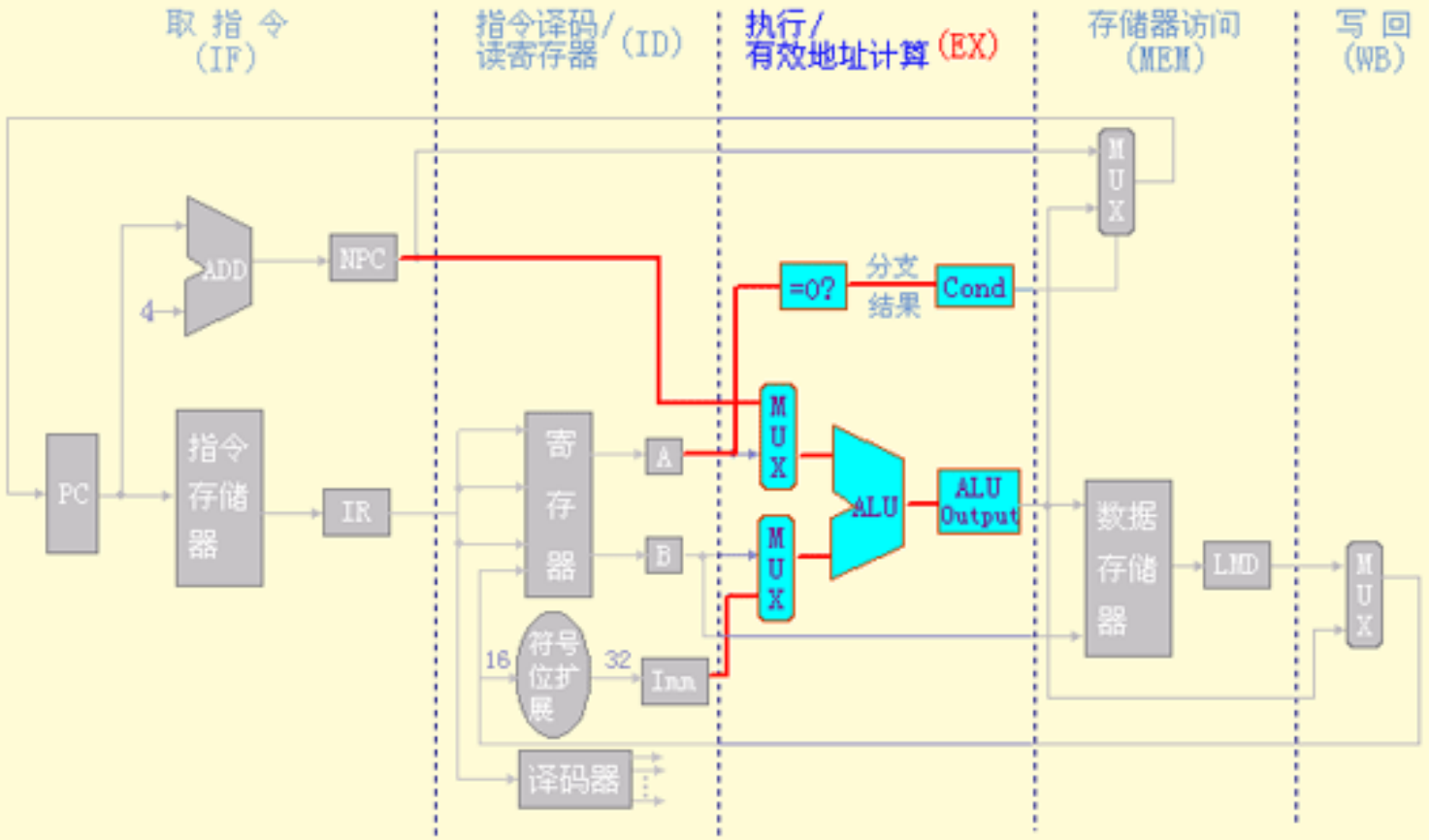
（寄存器-寄存器ALU操作指令）



执行／有效地址计算周期的操作
（寄存器-立即值ALU操作指令）

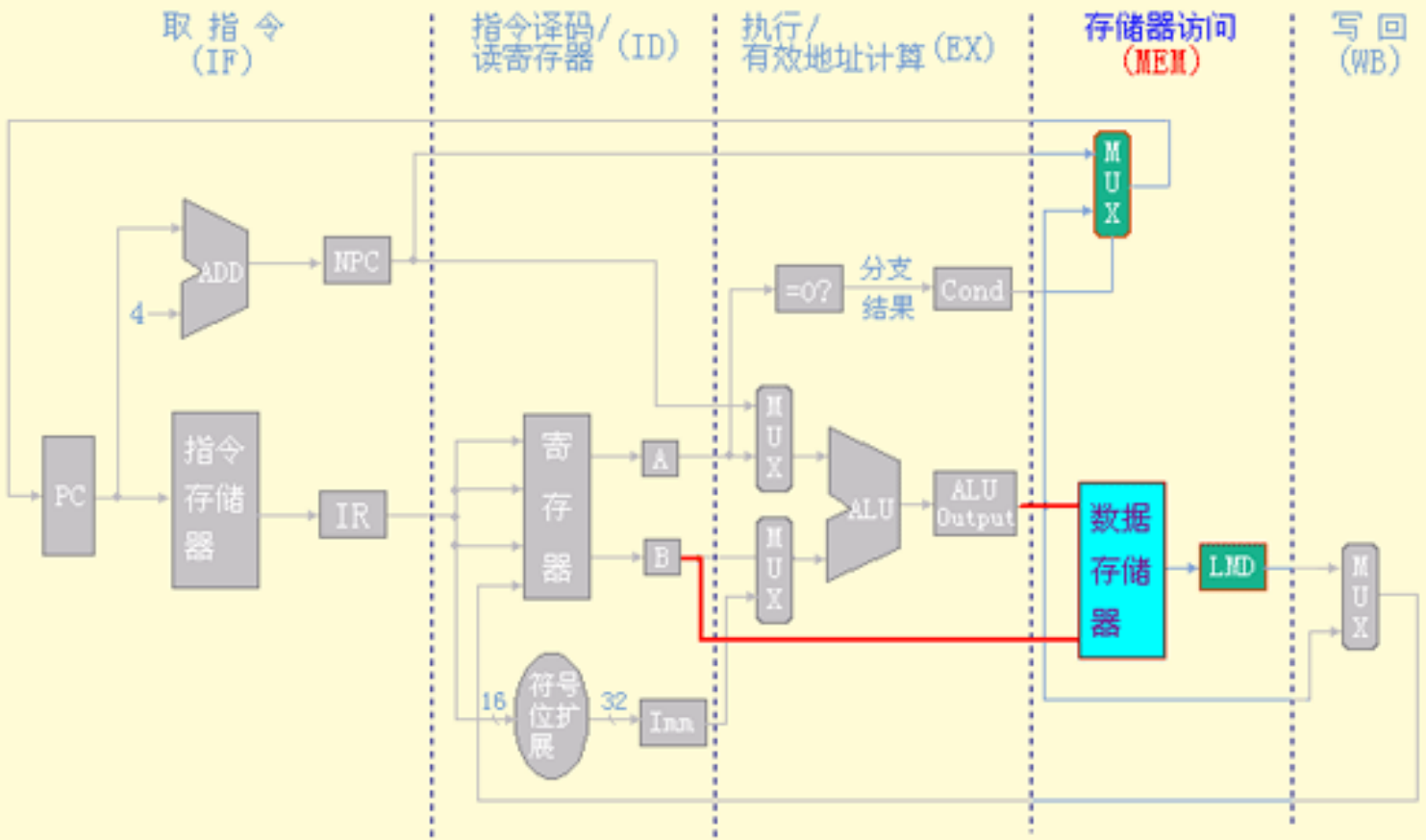


执行／有效地址计算周期的操作
（分支操作指令）

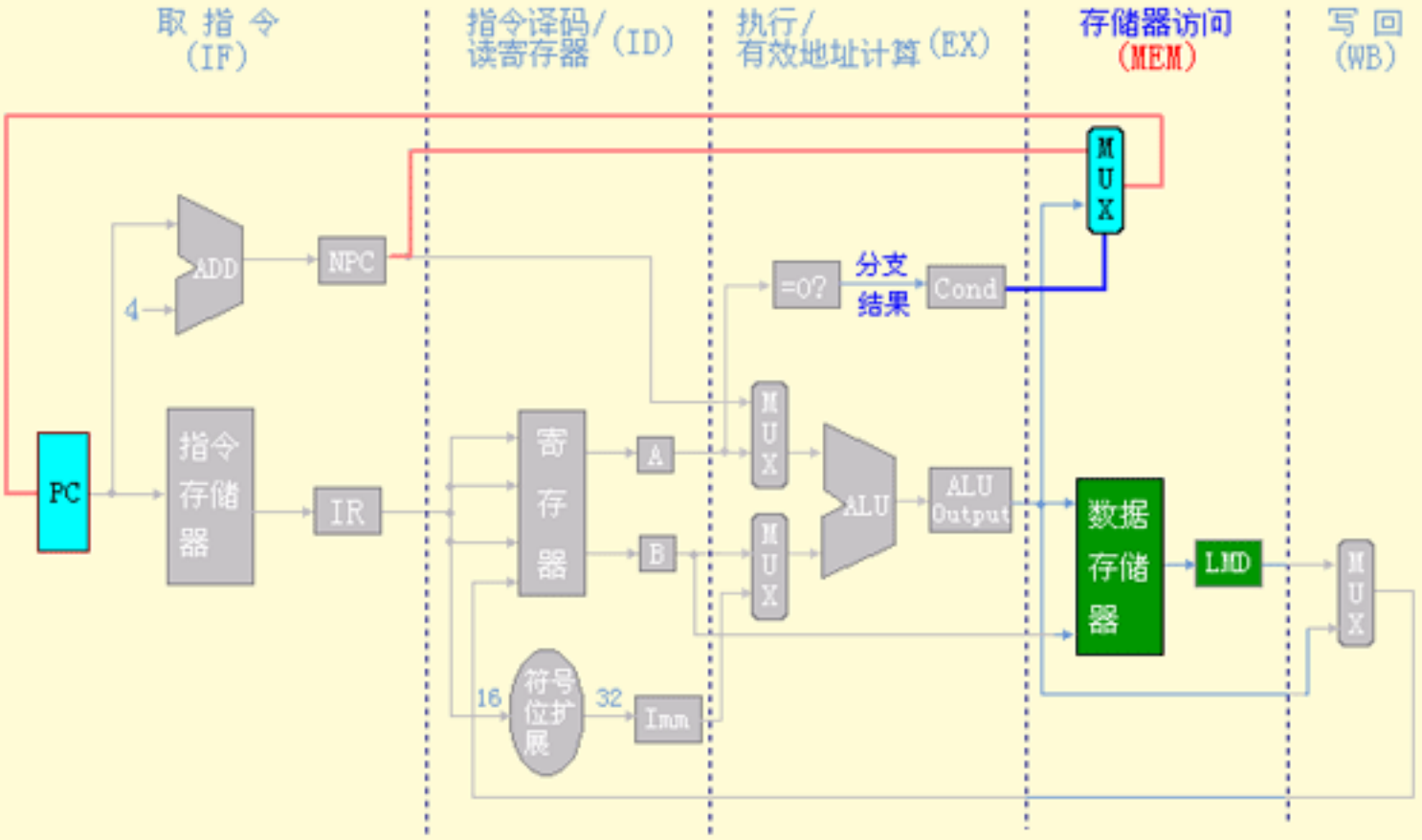


存储器访问/分支完成周期的操作

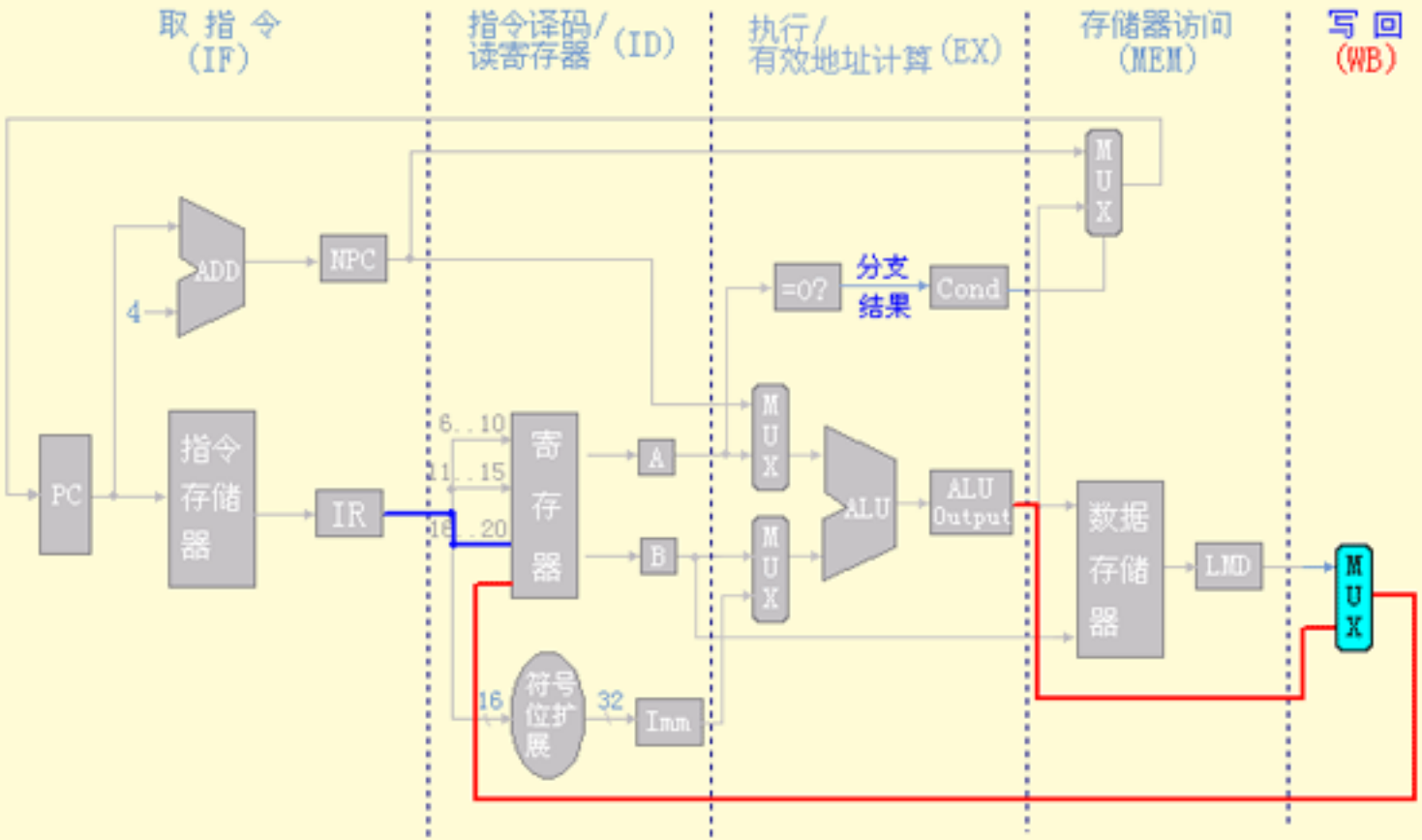
（存储器访问操作指令）



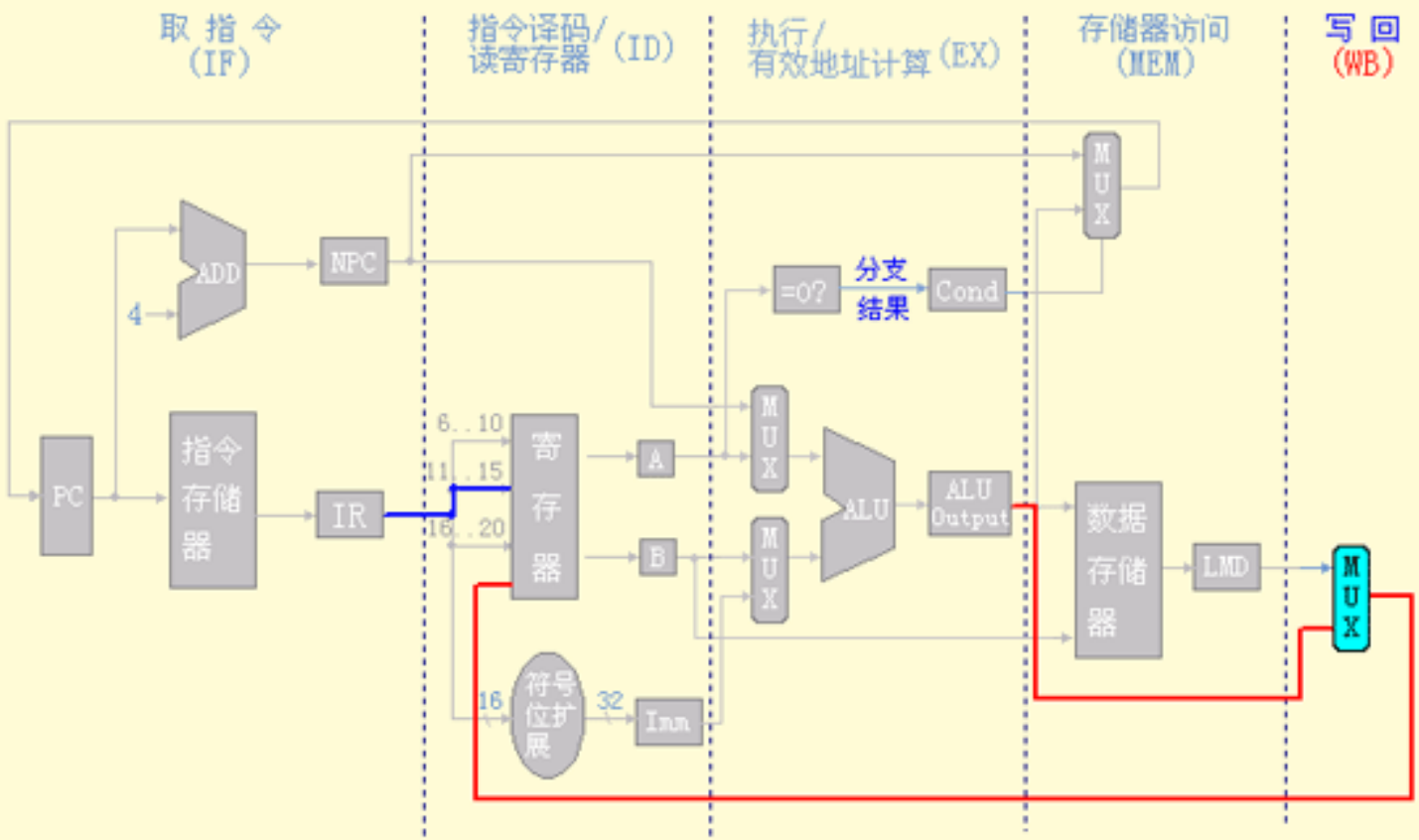
存储器访问/分支完成周期的操作
（分支操作指令）



写回周期的操作
（寄存器-寄存器型ALU指令）



写回周期的操作
（寄存器-立即值型ALU指令）



写回周期的操作

（Load指令）