
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS
北京邮电大学

第5章 中央处理器2


BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS
北京邮电大学

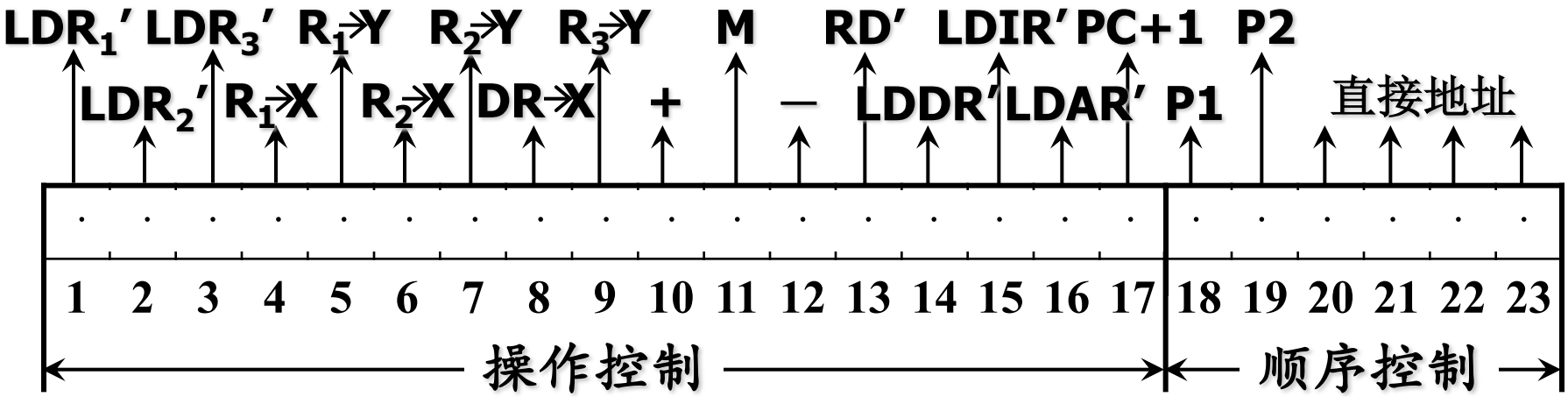
5.4.2 微程序设计技术

1. 微命令编码

- 微指令操作控制字段的微命令表示方法可分为3类
  - 直接表示法
  - 编码表示法
  - 混合表示法
- 在微指令中还可附设一个常数字段
  - 可作为操作数送入ALU运算
  - 也可作为计数器初值用来控制微程序循环次数

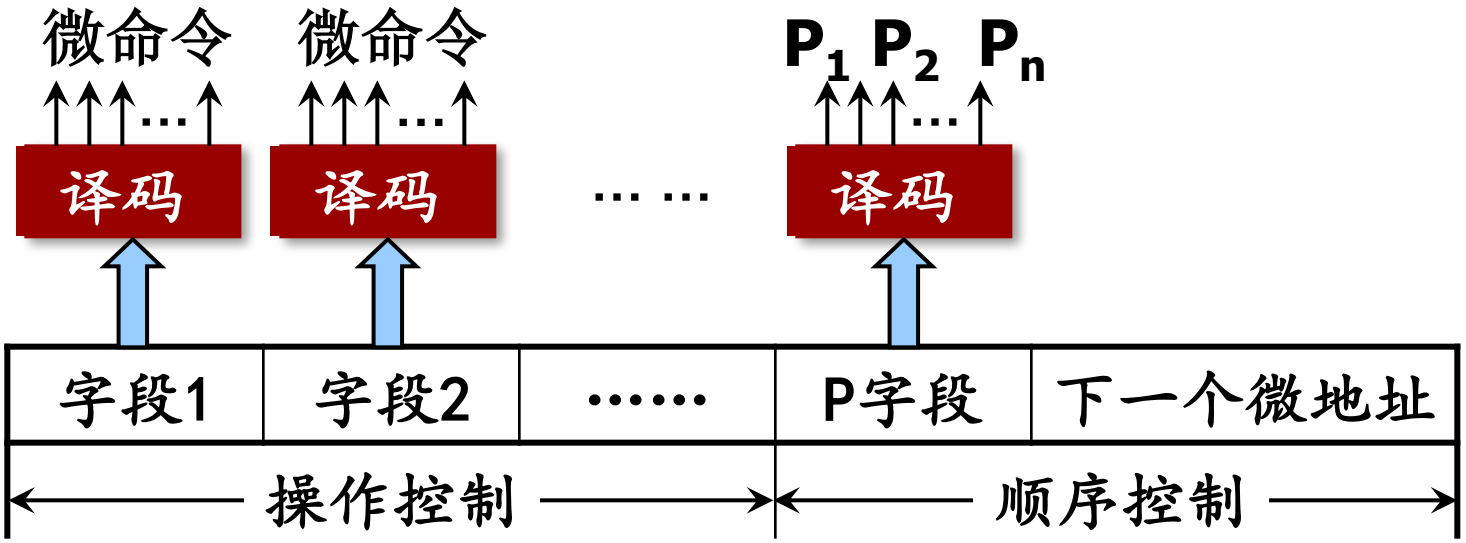
直接表示法

- 其特点是操作控制字段中的每一位代表一个微命令
- 优点：简单直观，其输出直接用于控制
- 缺点：微指令字较长，因而使控制存储器容量较大



编码表示法（1）

- 把一组相斥性的微命令信号组成一个小组(即一个字段)，然后通过小组(字段）译码器对每一个微命令信号进行译码，译码输出作为操作控制信号

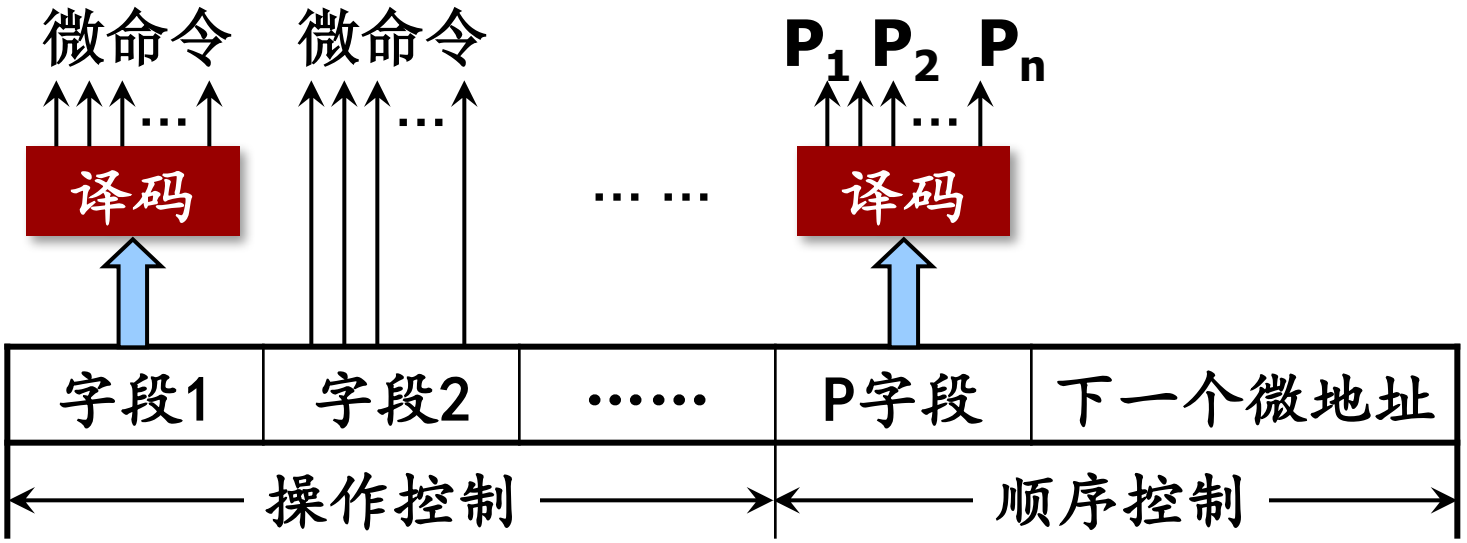


编码表示法（2）

- 用较少的二进制位数表示较多的微命令信号，n位二进位译码后可表示2^n-1个微命令
- 与直接控制法相比，编码表示法使微指令字的长度大大缩短。但由于增加译码电路，微程序的执行速度稍稍减慢
- 编码表示法使用较普遍

混合表示法

- 把直接表示法与字段编码法混合使用，以便能综合考虑指令字长、灵活性、执行微程序速度等方面的要求



2. 微地址的形成方法

- 微指令执行的顺序控制问题，实际上是如何确定下一条微指令的地址问题
- 产生后继微地址的方法
  - 计数器方式
  - 多路转移方式
  - 由指令的操作码转换得到，典型方式为查表方式，即

    在ROM中存放微地址映射表，用指令的操作码作为地址进行查表，查到的表项为该指令对应的微程序入口地址

计数器方式

- 与程序计数器PC类似
  - 在顺序执行微指令时，后继微地址由当前微地址加上一个增量来产生
  - 在非顺序执行微指令时，必须通过转移方式，转去执行指定微地址的微指令
- 微地址寄存器通常可视为计数器，且顺序执行的微指令序列必须安排在控制存储器的连续单元中
- 特点
  - 微指令的顺序控制字段较短，微地址产生机构简单
  - 但是多路并行转移功能较弱，速度较慢，灵活性较差

多路转移方式

- 一条微指令具有多个转移分支的能力称为多路转移
- 当微程序不产生分支时，后继微地址直接由微指令的顺序控制字段给出
- 当微程序出现分支时，有若干“候选”微地址可供选择：即按顺序控制字段P的“判别测试”和“状态条件”标志来选择其中一个微地址
- “状态条件”有n位标志，可实现微程序2^n路转移，且涉及微地址寄存器的n位
- 特点
  - 能以较短的顺序控制字段配合，实现多路并行转移，灵活性好，速度较快
  - 转移地址逻辑需要用组合逻辑方法设计

例2

微地址寄存器有6位(μA_5-μA_0)，当需要修改其内容时，可通过某一位触发器的置“1”端S将其置“1”。现有三种情况：

(1)执行“取指”微指令后，微程序按IR的OP字段(IR_3-IR_0)进行16路分支；

(2)执行条件转移指令微程序时，按进位标志C的状态进行2路分支；

(3)执行某控制台指令微程序时，按IR_4、IR_5的状态进行4路分支。

请按多路转移方法设计微地址转移逻辑。

例3解（1）

解：按所给设计条件，微程序有三种判别测试方式，分别为P_1，P_2，P_3。 由于修改μA_5-μA_0内容具有很大灵活性，现分配如下：

(1)用P_1和IR_3-IR_0修改μA_3-μA_0

(2)用P_2和C修改μA_0

(3)用P_3和IR_5，IR_4修改μA_5，μA_4

例3解（2）

另外还要考虑时间因素T_4（假设CPU周期最后一个节拍脉冲），故转移逻辑表达式如下：

$$\mu A_5=P_3\cdot IR_5\cdot T_4 \qquad \mu A_4=P_3\cdot IR_4\cdot T_4$$

$$\mu A_3=P_1\cdot IR_3\cdot T_4 \qquad \mu A_2=P_1\cdot IR_2\cdot T_4$$

$$\mu A_1=P_1\cdot IR_1\cdot T_4 \qquad \mu A_0=P_1\cdot IR_0\cdot T_4+P_2\cdot C\cdot T_4$$

由于在触发器强置端（低电平有效）修改，所以前5个表达式可用“与非”门实现，最后一个用“与或非”门实现

3. 微指令格式

- 微指令的格式大体分成两类
  - 水平型微指令
  - 垂直型微指令

水平型微指令

- 一次能定义并执行多个并行操作微命令的微指令

控制字段	判别测试字段	下地址字段

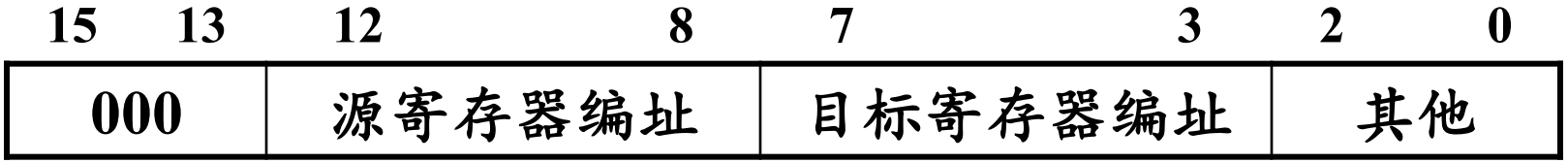
- 按控制字段的编码方法不同可分为:
  - 全水平型（不译码法）微指令
  - 字段译码法水平型微指令
  - 直接和译码相混合的水平型微指令

垂直型微指令

- 微指令中设置微操作码字段，由微操作码规定微指令的功能
- 结构类似于机器指令的结构，它有操作码，在一条微指令中只有1～2个微操作命令，每条微指令的功能简单
- 实现一条机器指令的微程序要比水平型微指令编写的微程序长得多
- 特点：微程序较长，而每一条微指令较短

垂直型微指令举例（1）

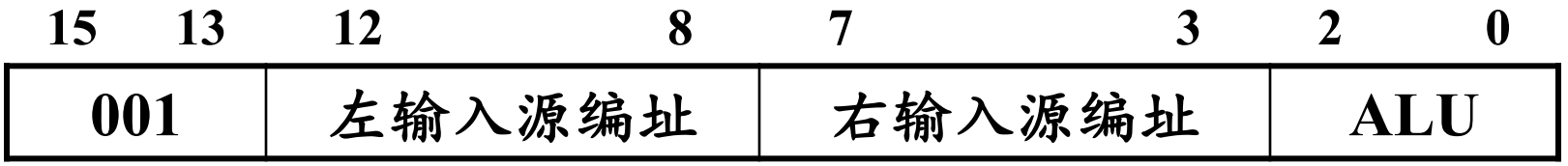
- 寄存器-寄存器传送型微指令



- 功能：把源寄存器数据送目标寄存器
- 13-15位为微操作码，源寄存器和目标寄存器变址各为5位，可指定31个寄存器

垂直型微指令举例（2）

- 运算控制型微指令



- 功能：选择ALU的左、右输入源信息，按ALU字段所指定的运算功能进行处理，并将结果送入暂存器中
- 有8种运算功能，左、右输入源编址可指定31种信息源之一

垂直型微指令举例（3）

- 访问主存微指令

15 13	12 8	7 3	2 1	0
010	寄存器编址	存储器源编址	读写	其他

- 功能：将主存中的一个单元的信息送入寄存器或者将寄存器的数据送往主存
- 存储器编址是按规定的寻址方式进行编址
- 第1、2位指定读操作或写操作（取其之一）

垂直型微指令举例（4）

- 条件转移微指令

- 功能：根据测试对象的状态决定是转移到D所指定的微地址单元，还是顺序执行下一条微指令
- 9位D字段不足以表示一个完整的微地址，但可以用来替代当前μPC的低位地址
- 测试条件字段有4位，可规定16种测试条件

水平型与垂直型微指令比较

- 水平型微指令并行操作能力强，效率高，灵活性强，垂直型微指令则较差
- 水平型微指令执行一条指令的时间短，垂直型微指令执行时间长
- 用水平型微指令解释指令的微程序，有微指令字较长而微程序短的特点，垂直型微指令则相反
- 水平型微指令用户难以掌握，而垂直型微指令与指令比较相似，相对来说，比较容易掌握

例

下列关于RISC的叙述中，错误的是

A．RISC普遍采用微程序控制器

B．RISC大多数指令在一个时钟周期内完成

C．RISC的内部通用寄存器数量相对CISC多

D．RISC的指令数、寻址方式和指令格式种类相对CISC少

例

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（下址字段法）确定下条微指令的地址，则微指令中下址字段的位数至少是多少位？

解：解释32条指令，微程序中总共的微指令条数至少有：

$32 \times 4 + 2 = 130$

故：微指令中下址字段的位数至少需要8位

例

某计算机的控制器采用微程序控制方式，微指令中的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：操作控制字段共15位

3	2	4	3	3

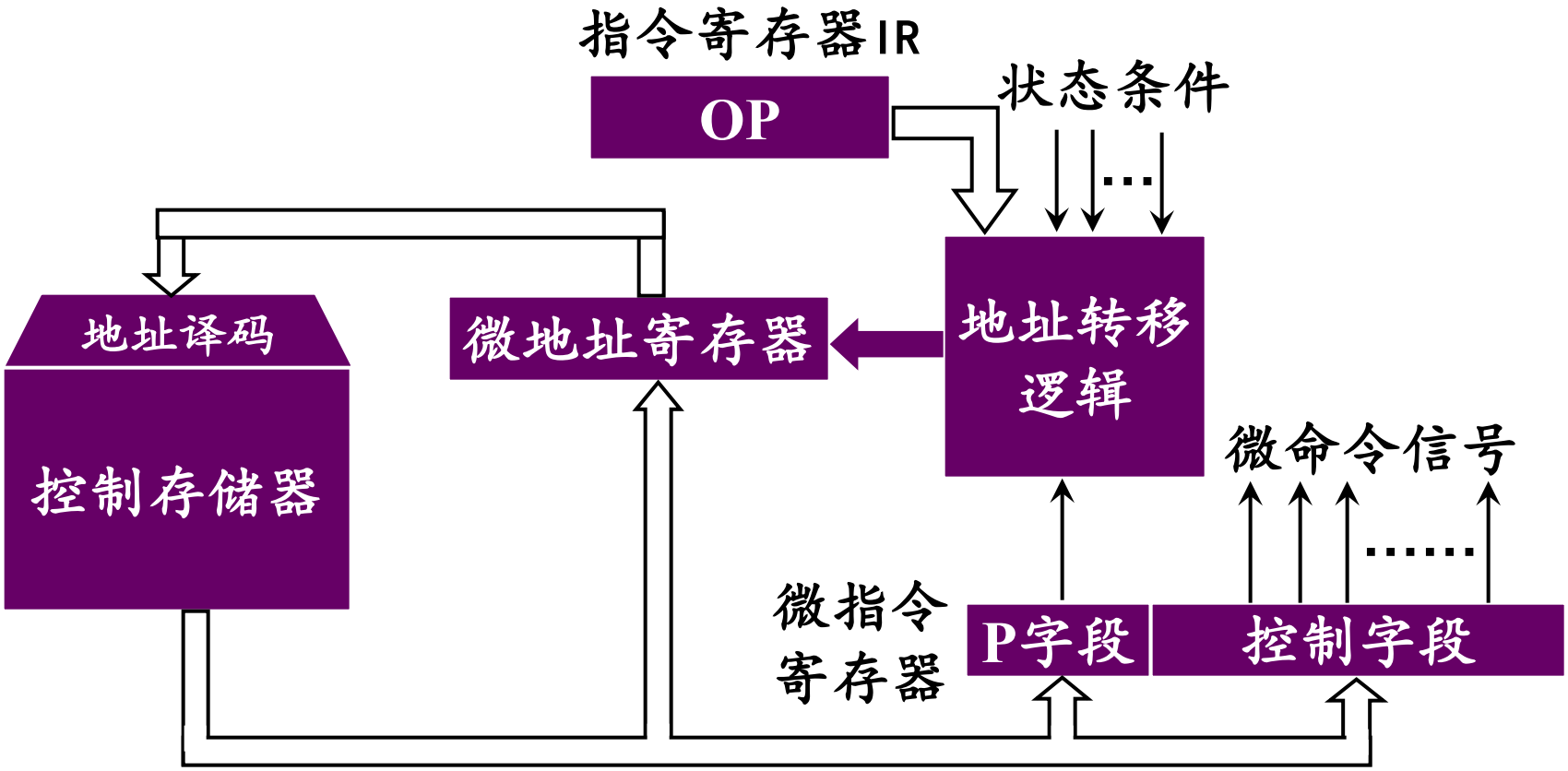

BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS
北京邮电大学

5.5 硬连线控制器

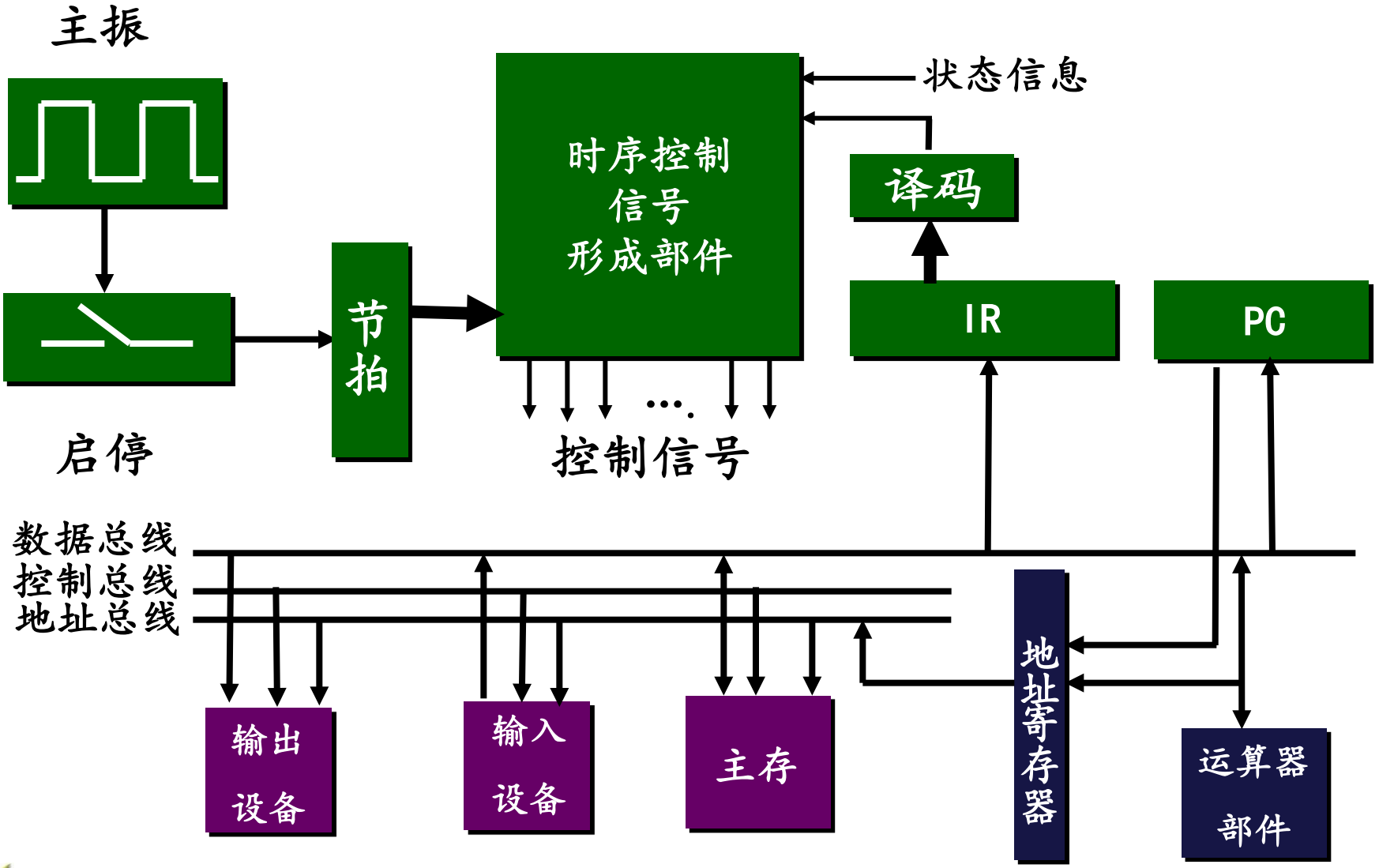
基本概念

- 硬布线控制器（也称为组合逻辑控制器）是早期计算机唯一的设计方案。也是当前RISC计算机和高性能计算机普遍采用的方案
- 控制部件为产生固定时序控制信号的逻辑电路
- 特点
  - 形成这些控制信号的传输延迟时间少，能提高系统的执行速度
  - 设计控制CPU各功能部件运行所需要的时序控制信号的逻辑比较复杂
  - 变更设计比较困难
  - 与微程序控制相比，硬布线控制的速度较快。因为微程序控制中每条微指令都要从控制存储器中读取；而硬布线控制取决于电路延迟

回顾 微程序控制器原理框图

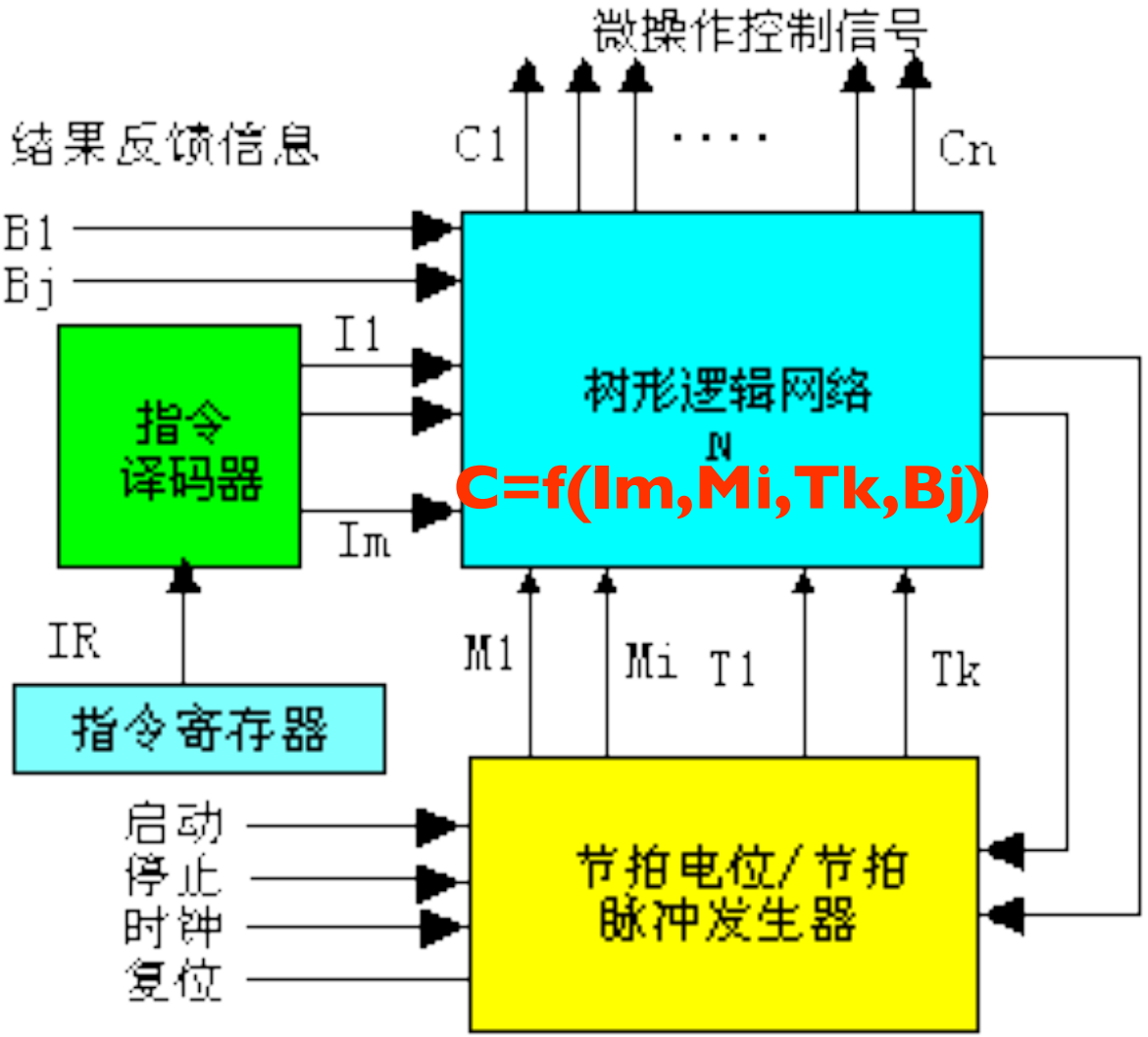


硬连线控制器原理框图


主振
状态信息
时序控制
信号
形成部件
译码
节
拍
IR
PC
启停
控制信号
数据总线
控制总线
地址总线
输出
设备
输入
设备
主存
地址寄存器
运算器
部件

硬连线控制器的结构图

- 逻辑网络的输入信号来源有三个：
- (1)来自指令操码译码器的输出Im
- (2)来自执行部件的反馈信息Bj
- (3)来自时序产生器的时序信号，包括节拍电位信号M和节拍脉冲信号T



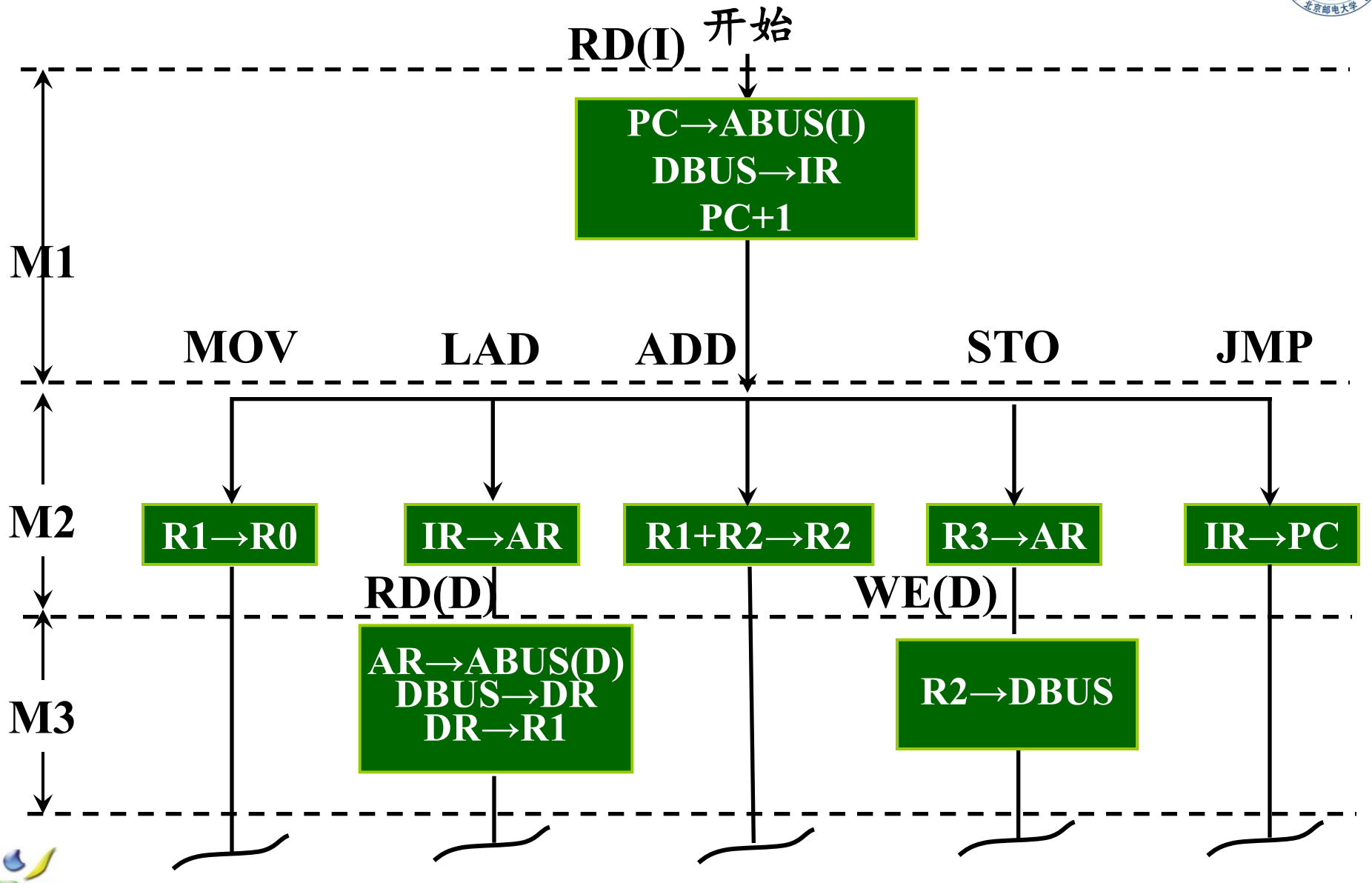
指令执行流程

- 在硬连线控制器中，通常时序产生器除了产生节拍脉冲信号外，还应当产生节拍电位信号（与微程序控制器不同）
- 在一个指令周期中，要顺序执行一系列微操作，这就需要设置若干个节拍电位，来保证这些微操作的执行的先后顺序。

例：若C3为主存读操作控制信号，在取指令或取操作数（例如：LAD）时该信号都有效

C3=M1+M3∧LAD）

硬连线控制器的指令周期流程图



微操作控制信号的产生

- 在微程序控制器中，微操作控制信号由微指令产生，并且可以重复使用
- 而在硬布线控制器中，某一微操作控制信号由布尔表达式描述的输出函数产生
- 设计微操作控制信号的方法：
  - 根据所有机器指令流程图，寻找出产生同一个微操作信号的所有条件
  - 与适当的节拍电位和节拍脉冲组合，写出其布尔代数表达式并进行简化
  - 用门电路或可编程器件来实现

例3（1）

根据上图，写出以下控制信号RD(I)、RD(D)、WE(D)、LDPC、LDIR、LDAR、LDDR、PC+1、LDR2的逻辑表达式。其中每个操作控制信号的含义：

RD（I）－指存读命令　RD（D）－数存读命令

WE（D）－数存写命令　LDPC－打入PC

LDIR－打入IR　　LDAR－打入数存地址寄存器

LDDR－打入数据缓冲寄存器　　PC＋1－PC加1

LDR2－打入R2寄存器

例3（2）

解：设M1、M2和M3为节拍电位，T1、T2、T3和T4为一个CPU周期中的节拍脉冲信号，MOV、LAD、ADD、STO和JMP分别表示对应机器指令的OP操作码译码输出信号，则有如下逻辑表达式：

RD（I）＝M1（电位信号）　RD（D）＝M3LAD（电位信号）

WE（D）＝M3T3STO（脉冲信号）

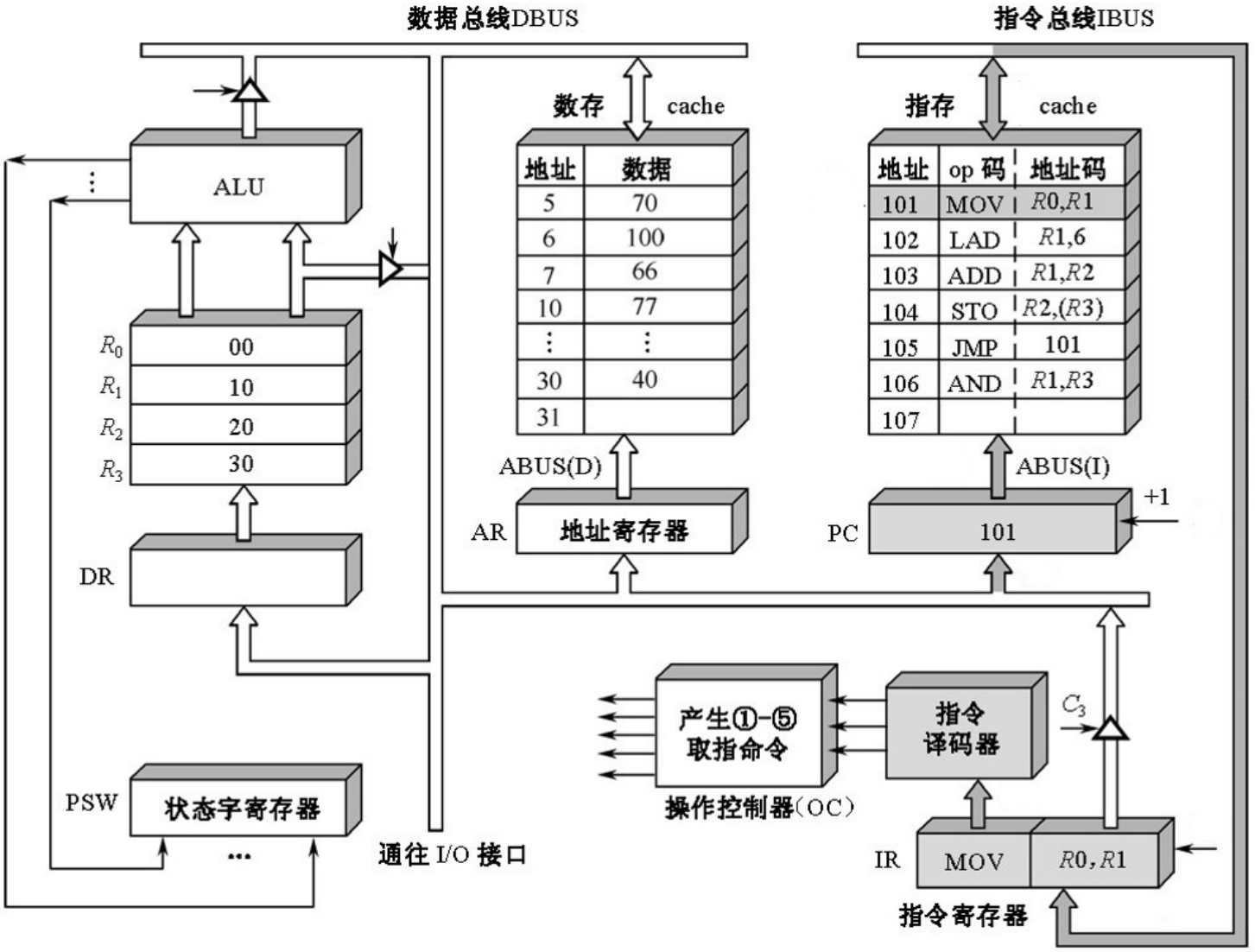
LDPC＝M1T4＋M2T4JMP

LDIR＝M1T4　　LDAR＝M2T4（LAD＋STO）

LDDR＝M2T3（MOV＋ADD）＋M3T3LAD　　PC+1＝M1T3

LDR2＝M2T4ADD

例3（3） CPU模型

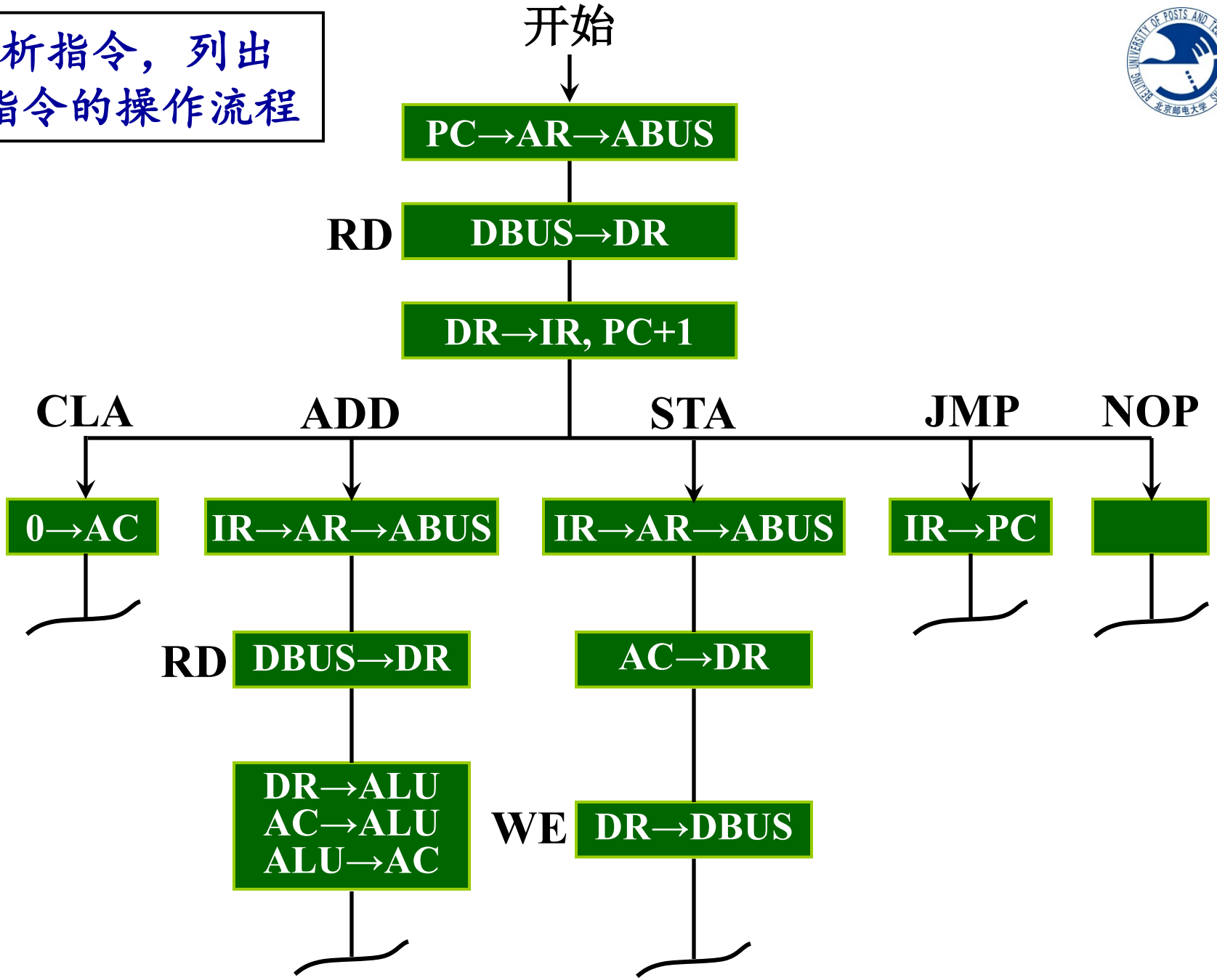

数据总线DBUS
指令总线IBUS
数存
cache
指存
cache
地址 数据
5 70
6 100
7 66
10 77
30 40
31
地址 op 码 地址码
101 MOV R0,R1
102 LAD R1,6
103 ADD R1,R2
104 STO R2,(R3)
105 JMP 101
106 AND R1,R3
107
ALU
R0 00
R1 10
R2 20
R3 30
DR
ABUS(D)
AR 地址寄存器
ABUS(I)
PC 101
+1
产生①-⑤
取指命令
指令
译码器
C3
操作控制器(OC)
PSW 状态字寄存器
通往 I/O 接口
IR MOV R0,R1
指令寄存器

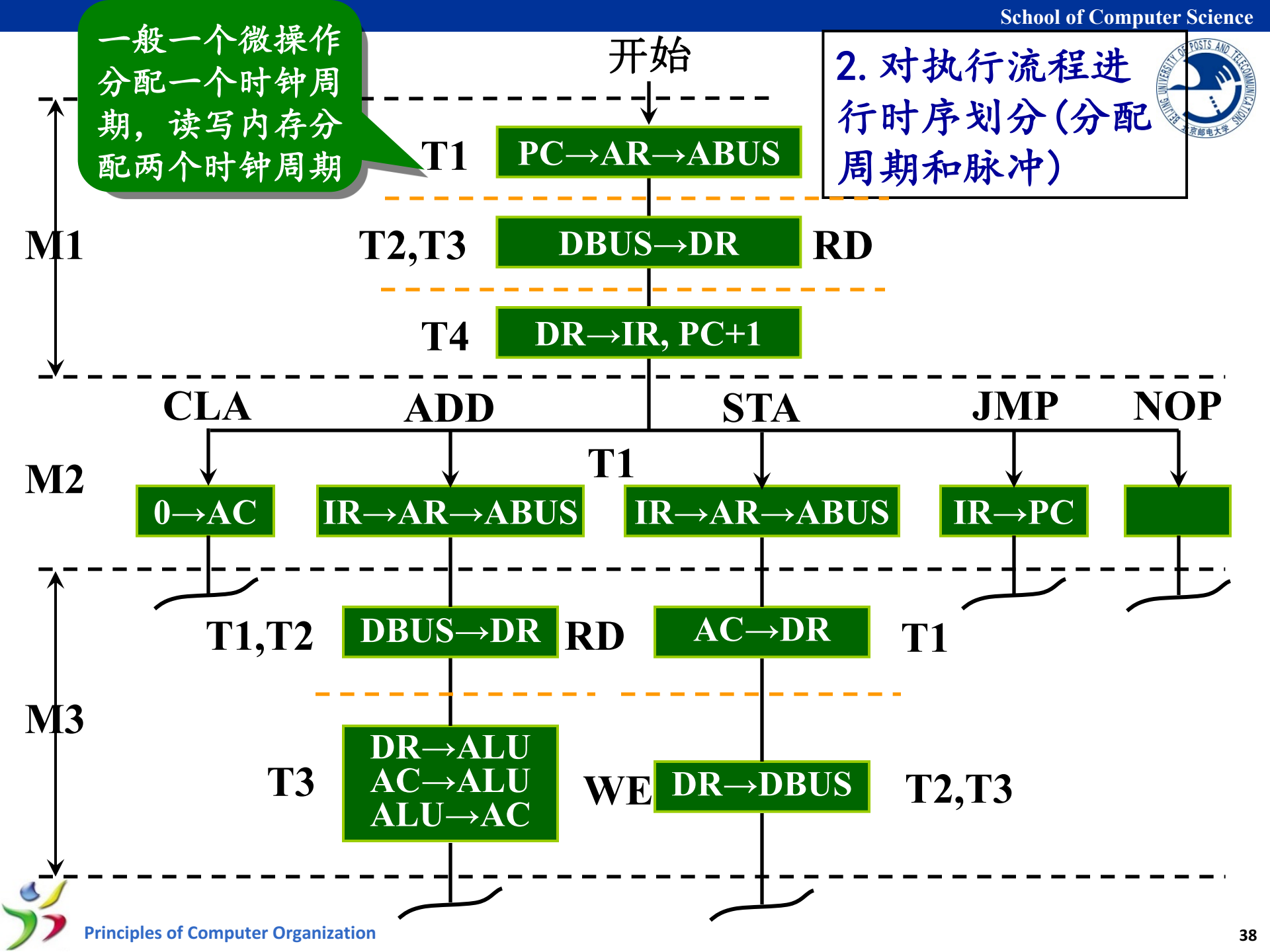
设计举例

设计下图所示模型机的控制器，设该机指令系统仅为五条指令，采用硬布线控制器，指令如下：

CLA	清零指令	指令功能： 0→AC
ADD AC,M	加法指令	(AC)+(M)→AC
STA M	存数指令	(AC)→M
JMP m	跳转指令	m→PC
NOP	空指令	延时

1. 分析指令，列出所有指令的操作流程




一般一个微操作分配一个时钟周期，读写内存分配两个时钟周期
2. 对执行流程进行时序划分(分配周期和脉冲)
开始
M1
T1
PC→AR→ABUS
T2,T3
DBUS→DR
RD
T4
DR→IR, PC+1
CLA
ADD
STA
JMP
NOP
M2
T1
0→AC
IR→AR→ABUS
IR→AR→ABUS
IR→PC
T1,T2
DBUS→DR
RD
AC→DR
T1
M3
T3
DR→ALU
AC→ALU
ALU→AC
WE
DR→DBUS
T2,T3

设计举例

3. 总结综合所有微操作如下表

C0	PC→AR→ABUS	C6	DR→DBUS
C1	PC+1→PC	C7	DBUS→DR
C2	IR→PC	C8	AC→DR
C3	IR→AR→ABUS	C9	AC→ALU
C4	DR→IR	C10	ALU→AC
C5	DR→ALU	C11	0→AC

设计举例

4. 列出各微操作的逻辑函数表达式

C0： PC→AR→ABUS = M1 T1

C1： PC+1→PC = M1 T4

C2： IR→PC = M2 T1 JMP

C3： IR→AR→ABUS = M2 T1(ADD + STA)

C4： DR→IR = M1 T4

C5： DR→ALU = M3 T3 ADD

C6： DR→DBUS = M3 (T2+T3) STA

C7： DBUS→DR = M1 (T2+T3) + M3 (T1+T2) ADD

C8： AC→DR = M3 T1 STA

C9： AC→ALU = M3 T3 ADD

C10： ALU→AC = M3 T3 ADD

C11： 0→AC = M2 T1 CLA

例

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


BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS
北京邮电大学

5.6 流水CPU

并行处理技术

- 早期计算机是冯·诺伊曼体系结构，采用串行处理
  - 计算机的各个操作只能串行地完成
  - 任一时刻只能进行一个操作
- 并行性（Parallelism）概念
  - 问题中具有可以同时进行运算或操作的特性
  - 例：在相同时延的条件下，用n位运算器进行n位并行运算速度几乎是1位运算器进行n位串行运算的n倍（狭义）
- （广义）含义
  - 只要在同一时刻（同时性）或在同一时间间隔内（并发性）完成两种或两种以上性质相同或不同的工作，它们在时间上相互重叠，体现出了并行性

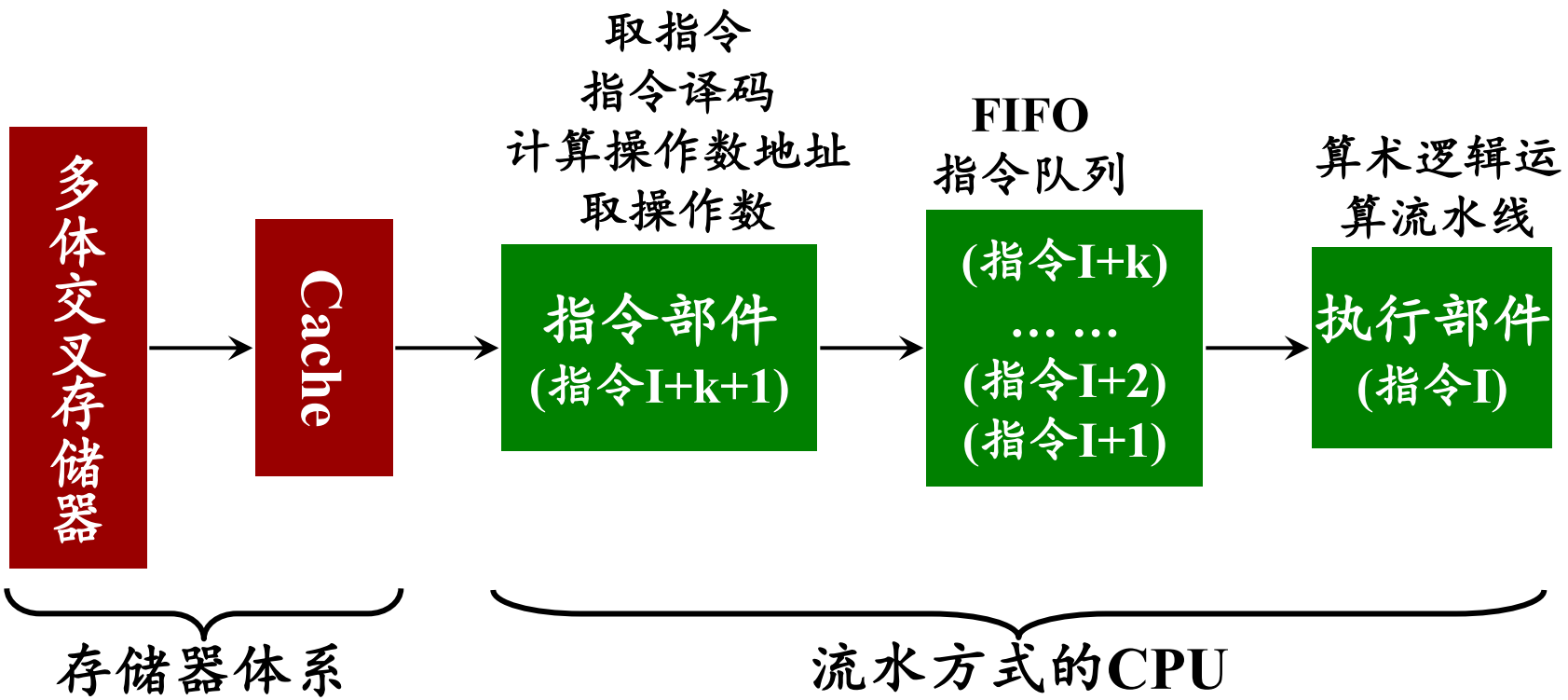
并行处理技术的三种形式

- 时间并行（重叠）：让多个处理过程在时间上相互错开，轮流使用同一套硬件设备的各个部件，以加快硬件周转而赢得速度，实现方式就是采用流水处理部件
- 空间并行（资源重复）：以数量取胜
  - 它能真正的体现同时性
  - LSI和VLSI为其提供了技术保证
  - 如：多处理器系统和多计算机系统
- 时间+空间并行
  - Pentium中采用了超标量流水线技术

流水CPU的结构

- 流水计算机的系统组成
  - 存储器体系：主存采用多体交叉存储器；Cache
  - 流水方式CPU：指令部件、指令队列、执行部件
    - 指令部件
    - 指令队列：FIFO寄存器栈
    - 执行部件：可以有多个采用流水线方式构成的算术逻辑部件构成，可以将定点运算部件和浮点运算部件分开

流水计算机系统组成原理示意图

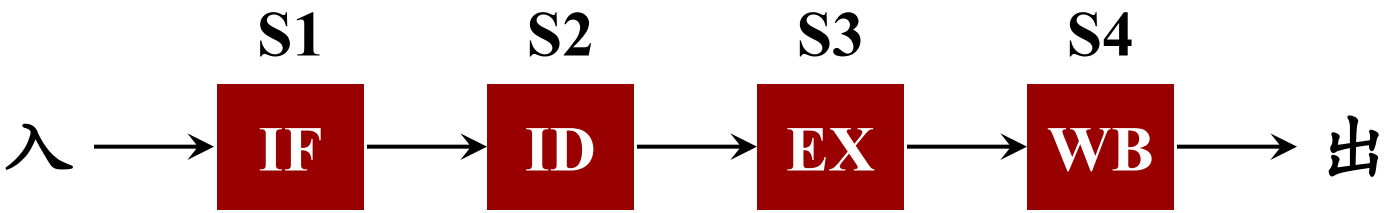


指令处理的分解

- 为了实现流水，把输入的任务（或过程）分割为一系列子任务，并使各子任务能在流水线的各个阶段并发地执行
- 当任务连续不断地输入流水线时，在流水线的输出端便连续不断地吐出执行结果，从而实现了子任务级的并行性
- IF（Instruction Fetch取指）
- ID（Instruction Decode指令译码）
- EX（Execution执行）
- WB（Write Back写回）

4级指令流水线

- 一个指令的流水线阶段



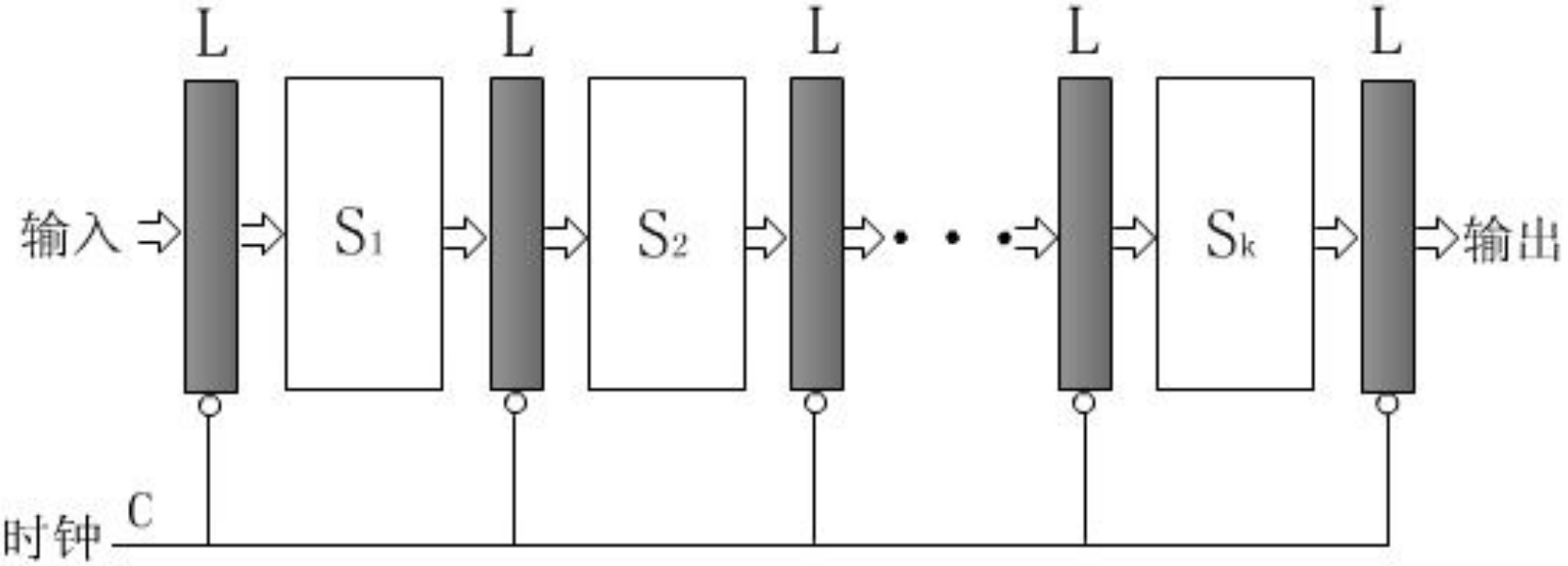
- 各阶段之间设有高速缓冲寄存器，以暂时保存上一过程段子任务处理的结果
- 在统一的时钟信号控制下，数据从一个阶段流向相邻下一阶段

流水硬件基本结构

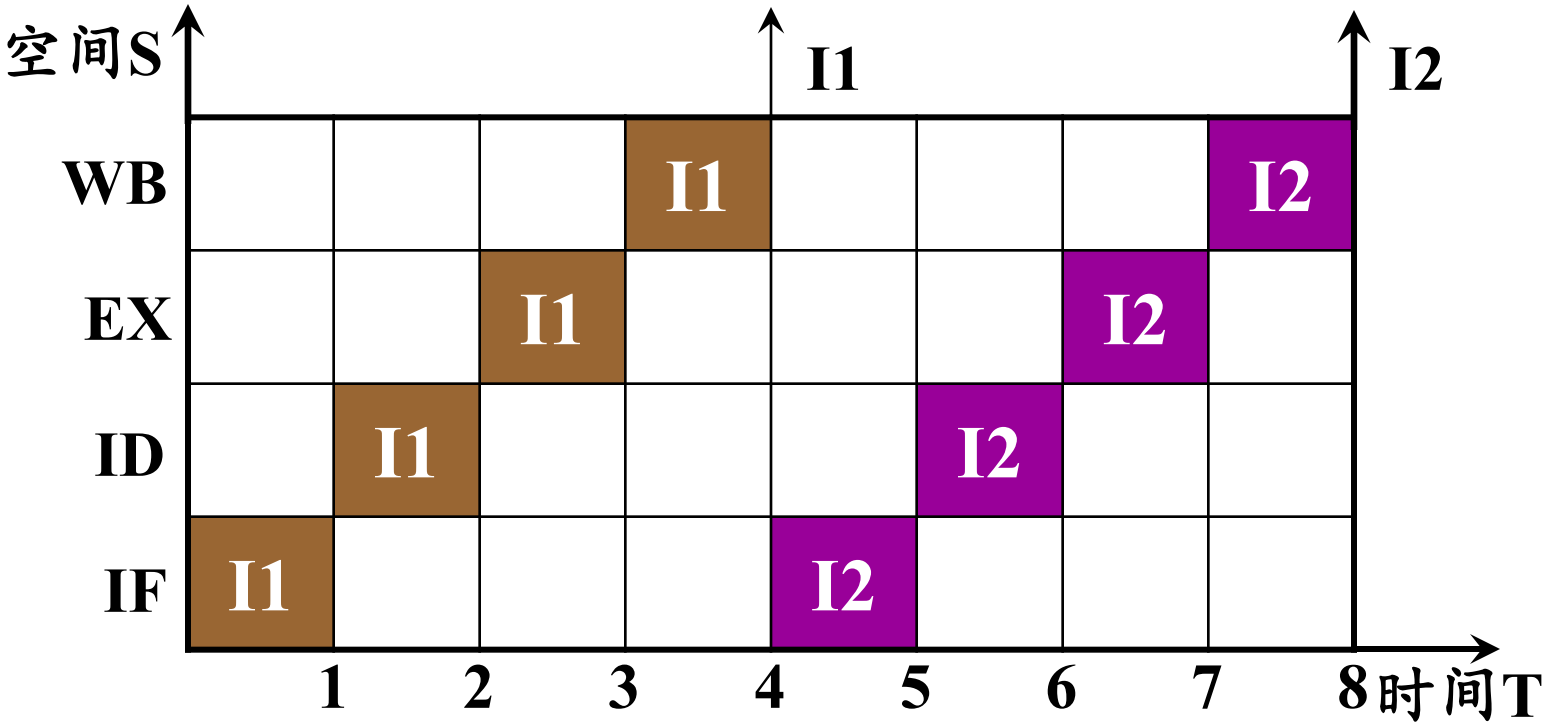
设阶段S_i所需的时间为τ_i，缓冲寄存器的延时为τ_l，线性流水线的时钟周期定义为

$$\tau = \max\{\tau_i\} + \tau_l = \tau_m + \tau_l$$

流水线处理的频率为 $f = 1/\tau$。

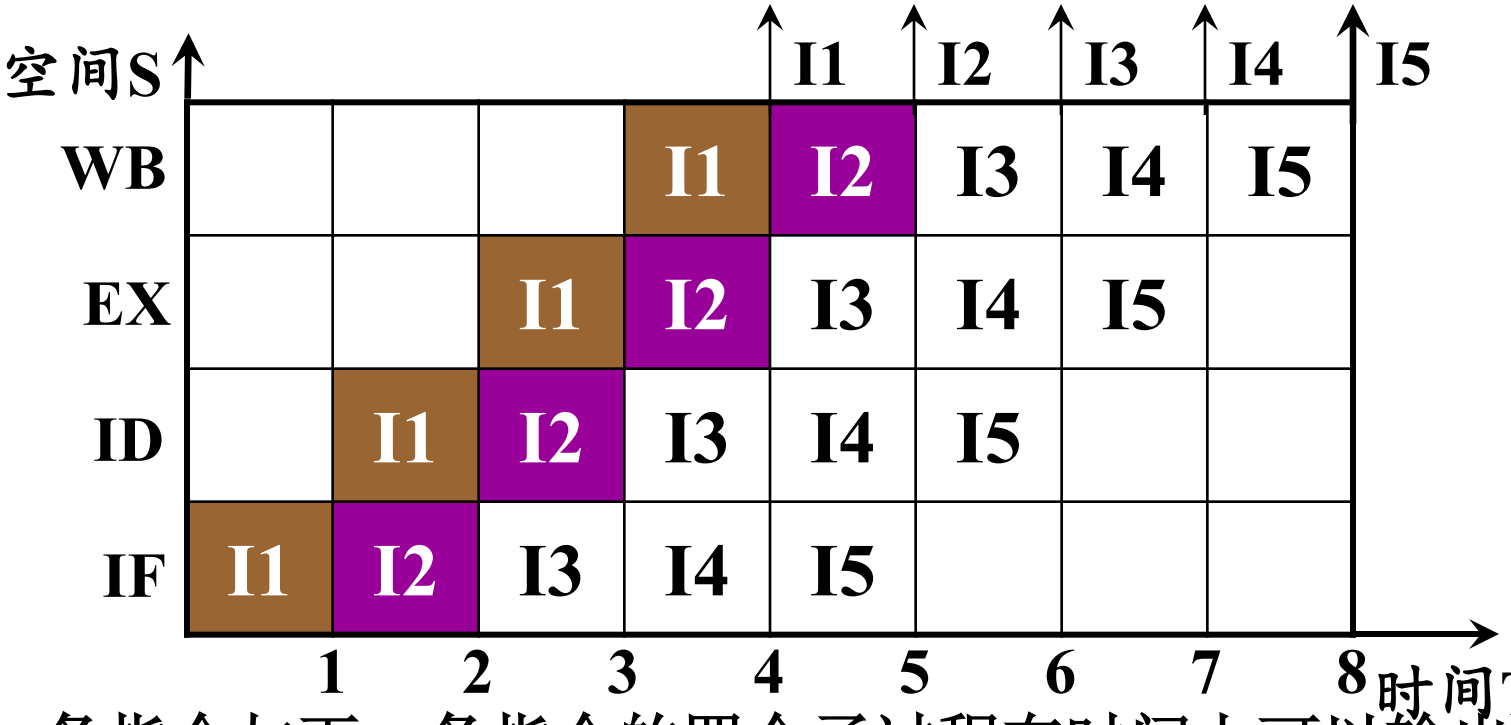


非流水线时空图



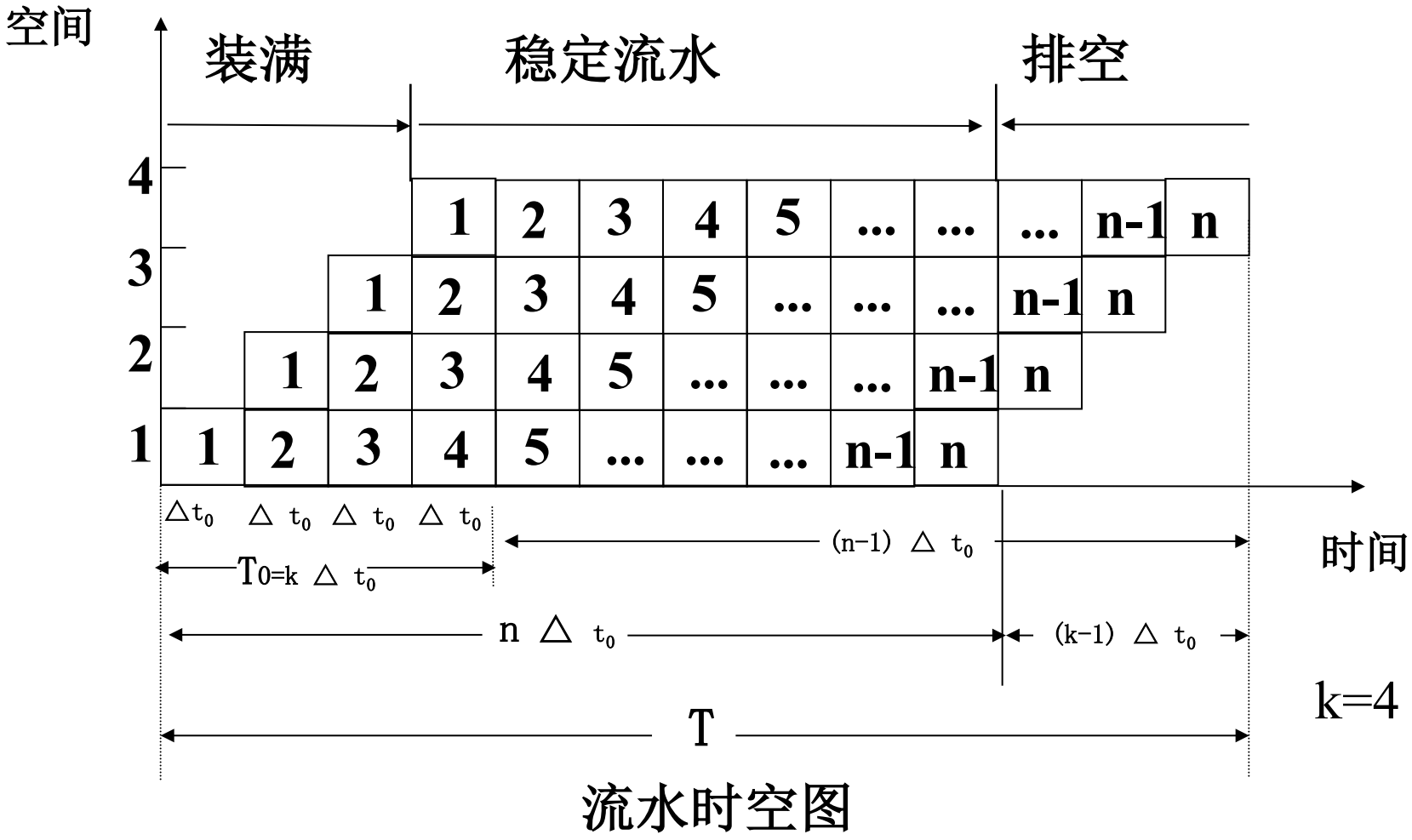
- 上一条指令的四个子过程全部被执行完毕后才能开始下一条指令
- 每隔4个机器时钟周期才有一个输出结果

标量流水线时空图



- 上一条指令与下一条指令的四个子过程在时间上可以输出重叠执行
- 每一个时钟周期就可以输出一个结果
- 对比
  - 流水计算机在8个单位时间中执行了5条指令
  - 非流水线计算机在8个单位时间中仅执行了2条指令

线性流水线的三个阶段

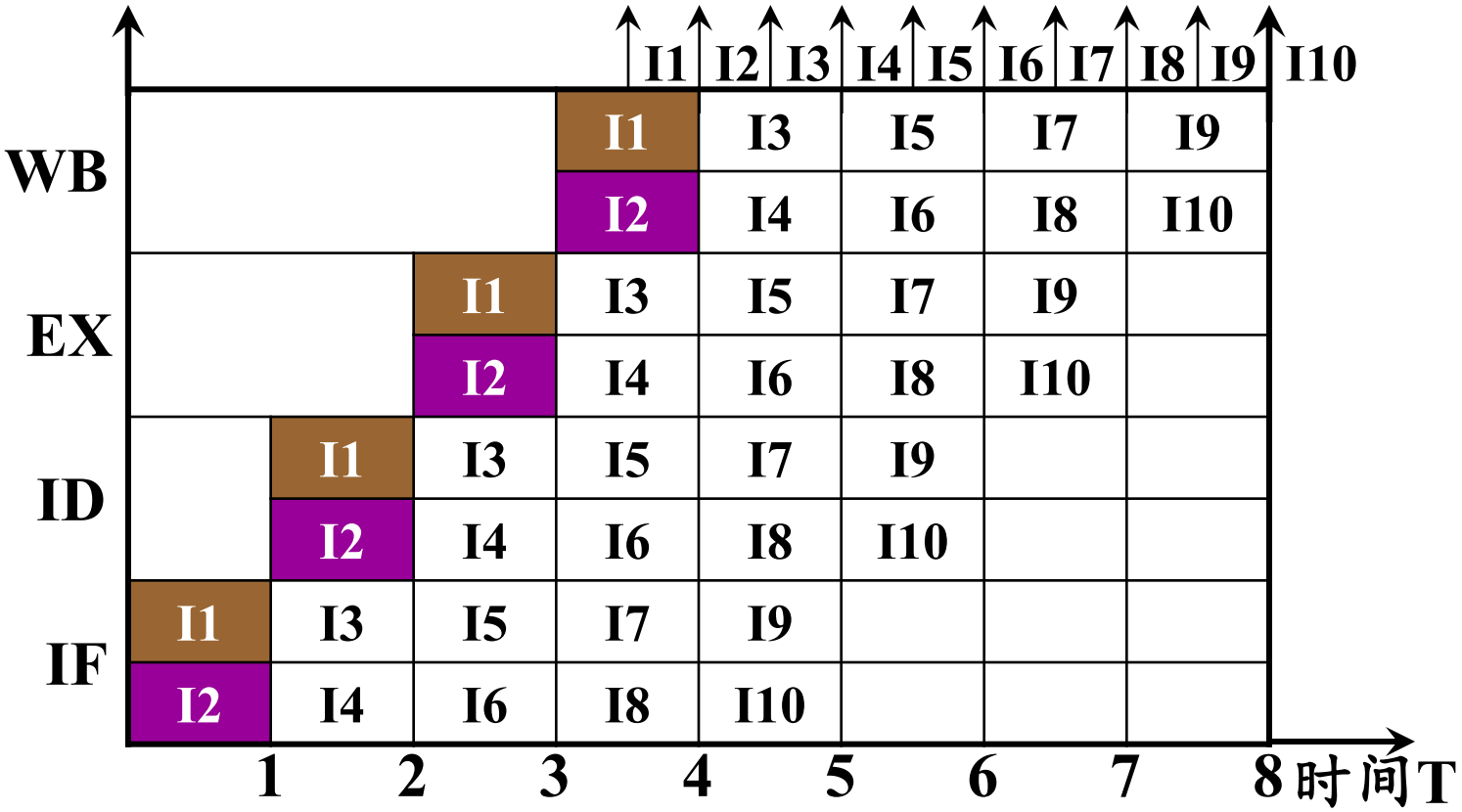


流水时空图

流水技术原理

- 在流水线（Pipelining）中必须是连续的任务，只有不断的提供任务才能充分发挥流水线的效率
- 把一个任务分解为几个有联系的子任务。每个子任务由一个专门的功能部件实现
- 在流水线中的每个功能部件之后都要有一个缓冲寄存器，或称为锁存器
- 流水线中各段的时间应该尽量相等，否则将会引起“堵塞”和“断流”的现象
- 流水线需要有装入时间和排空时间，只有当流水线完全充满时，才能充分发挥效率

超标量流水线时空图



流水线分类

- 指令流水线
  - 指令步骤的并行
- 算术流水线
  - 运算操作步骤的并行
  - 如流水加法器、流水乘法器、流水除法器等
- 处理机流水线
  - 程序步骤的并行，也称为宏流水线
  - 多台级联的处理机构成流水线的各过程段，每台处理机负责某一特定的任务
  - 处理机流水线应用在多机系统中

流水线的性能指标（1）

- 流水线的加速比（speedup ratio）：衡量在不使用流水线和使用流水线两种情况下完成同样任务所用的时间比：

$$S=T_0/T_k$$

假设k级流水线的时钟周期为 τ，则

不使用流水线顺序执行n个任务所需时间：

$$T_0 = n \cdot k \cdot \tau$$

使用k级流水线完成n个连续任务所需时间：

$$T_k = k \cdot \tau + (n-1)\,\tau = (k+n-1) \cdot \tau$$

故：$S = n \cdot k \cdot \tau/(k+n-1)\,\tau = n \cdot k/(k+n-1)$

在n＞＞k的情况下，最大加速比为k

流水线的性能指标（2）

- 吞吐率

  $TP = n / T_k$

  其中n为指令条数，T_k为执行n条指令所用时间

$$TP = \frac{n}{T_k} = \frac{n}{(k+n-1)\tau}$$

- 最大吞吐率

$$TP_{max} = \lim_{n\to\infty} \frac{n}{(k+n-1)\tau} = \frac{1}{\tau}$$

流水线的特点

- 流水线操作并不能加快任何一条指令的执行过程，只能加快连续一串指令的执行过程
- 多个不同任务同时操作，使用不同资源
- 潜在加速比= 流水线级数
- 流水线的速率受限于最慢的流水段
- 流水段的执行时间如果不均衡，那么加速比就会降低
- 开始填充流水线的时间和最后排放流水线的时间降低了加速比
- 可能产生相关（冒险），将导致流水线暂停

流水线冒险

流水线存在一种情况，在下一个时钟周期中下一条指令不能执行，这种情况称为冒险（hazard）

- 资源相关（结构冒险structural hazard）：资源相关是指多条指令进入流水线后在同一机器时钟周期内争用同一个资源所发生的冲突
- 数据相关（数据冒险data hazard）：下一条指令会用到当前指令计算出的结果
  - 写后读RAW（Read After Write）：后面指令用到前面指令所写的数据
  - 写后写WAW（Write After Write）：后面指令覆盖前面指令所写的单元
  - 读后写WAR（Write After Read）：后面指令覆盖前面指令所读的单元
- 控制相关（控制冒险 control hazard）：由转移指令引起。即为了确保预取正确指令而导致的延迟

资源相关实例

- 两条指令同时访问内存发生资源相关冲突

指令 \ 时钟	1	2	3	4	5	6	7	8
I1(Load)	IF	ID	EX	MEM	WB			
I2		IF	ID	EX	MEM	WB		
I3			IF	ID	EX	MEM	WB	
I4				IF	ID	EX	MEM	WB
I5					IF	ID	EX	MEM

- 解决冲突的2种典型方法
  - 第I4条指令停顿一拍后再启动
  - 增设一个存储器，将指令和数据分别放在两个存储器中

数据相关实例

- ADD R1,R2,R3 ;(R2)+(R3)→(R1)
- SUB R4,R1,R5 ;(R1)-(R5)→(R4)
- AND R6,R1,R7 ;(R1)∧(R7)→(R6)

指令 \ 时钟	1	2	3	4	5	6	7	8
ADD	IF	ID	EX	MEM	WB			
SUB		IF	ID	EX	MEM	WB		
AND			IF	ID	EX	MEM	WB	

- 解决冲突的2种典型方法：
  - 可以推迟后继指令对相关单元的读操作
  - 数据转发（data forwarding）：将结果直接从一个流水线阶段传到较早阶段的技术，也成为数据旁路（data bypassing）或前推（ forwarding ）

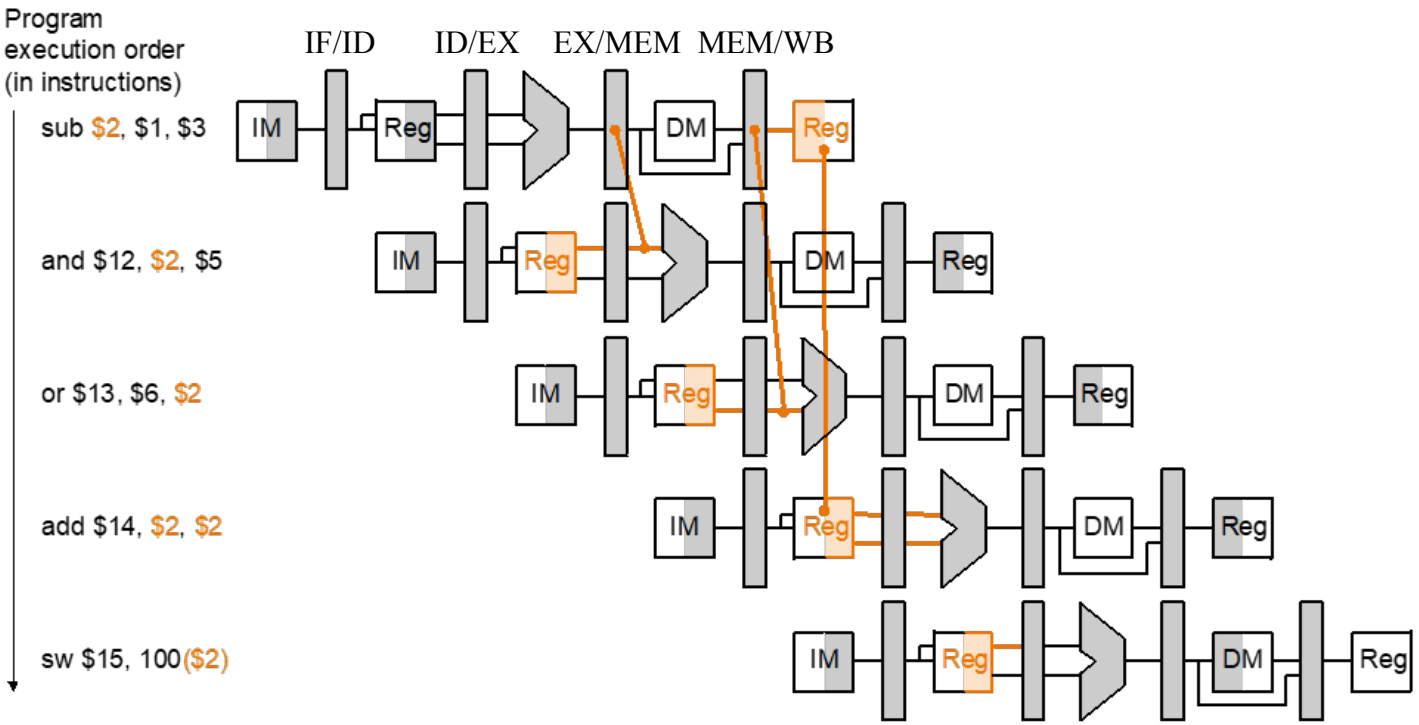
数据转发

- 输入到ALU的数据可以来自于ID/EX、或流水线后续的寄存器
- 使用控制信号和多路选择器选择合适的输入至ALU

Time (in clock cycles)

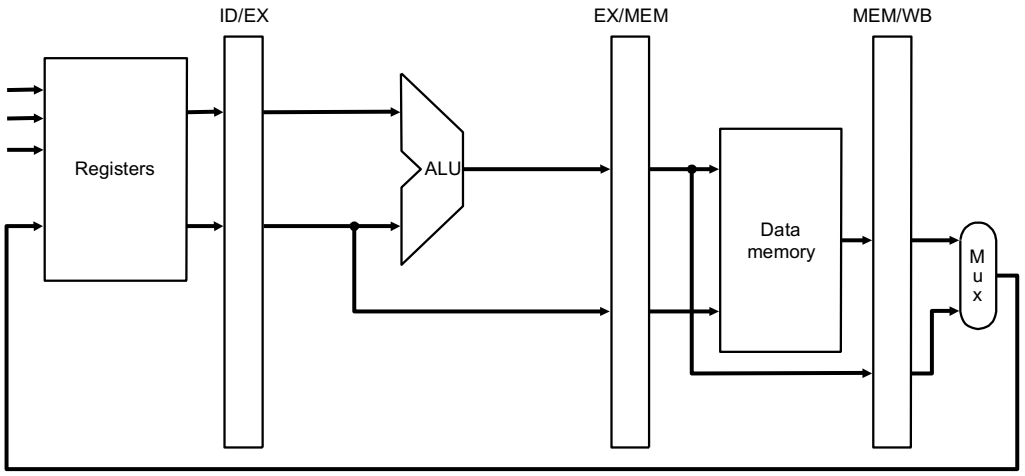
	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2 :	10	10	10	10	10/−20	−20	−20	−20	−20
Value of EX/MEM :	X	X	X	−20	X	X	X	X	X
Value of MEM/WB :	X	X	X	X	−20	X	X	X	X

```
sub $2,  $1,  $3
and $12, $2,  $5
or  $13, $6,  $2
add $14, $2,  $2
sw  $15, 100($2)
```



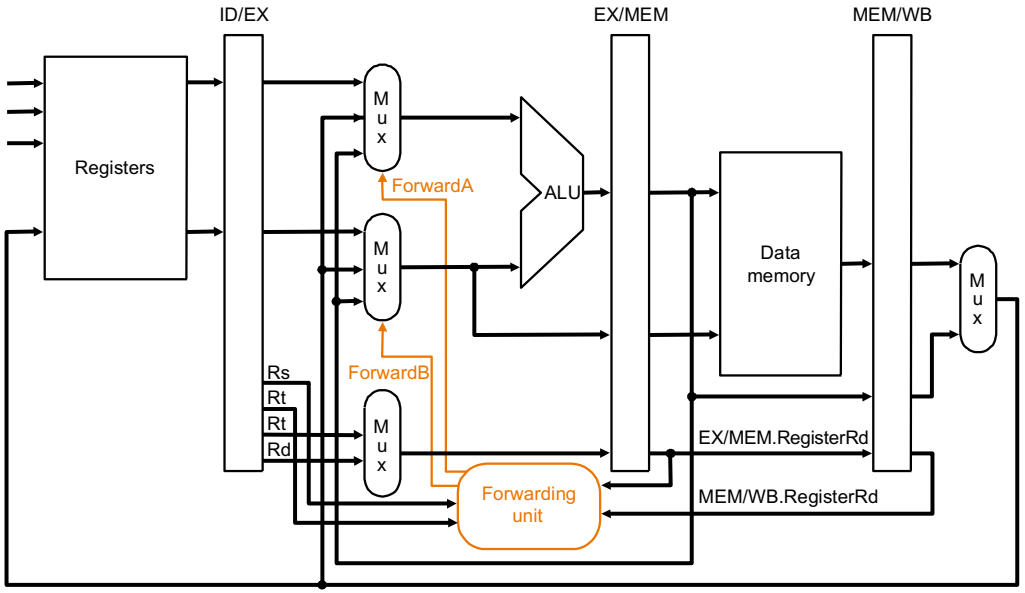
数据转发的实现

未使用数据转发



a. No forwarding

增加数据转发



b. With forwarding

控制相关（1）

- 控制相关冲突是由转移指令引起的。在执行转移指令时，依据转移条件，可能顺序执行，也可能转移执行。后者将使流水线发生断流。

```
if  (i==0)
      k=1;
   else
      k=2;
```

```
          CMP i,0
          BNE else
then:MOV k,1
          BR next
else: MOV k,2
       next:
```

控制相关（2）

- 常用以下两种转移处理技术：
  - 分支延迟时间槽（branch delay slot）
    - 从条件转移指令进入流水线到得到条件码结果这个时段（称之为分支延迟时间槽），其后续若干指令也进入流水线，如果条件转移指令发生转移，则那些后续已进入流水线并译码的指令需要废弃，导致流水线断流。
    - 解决方法：在延迟槽中安排如下指令：转移指令之前的指令或转移指令目标处的指令。
    - 这种重排指令序列的处理由编译程序来完成，且指令执行顺序的变动不能影响到程序的正确性。

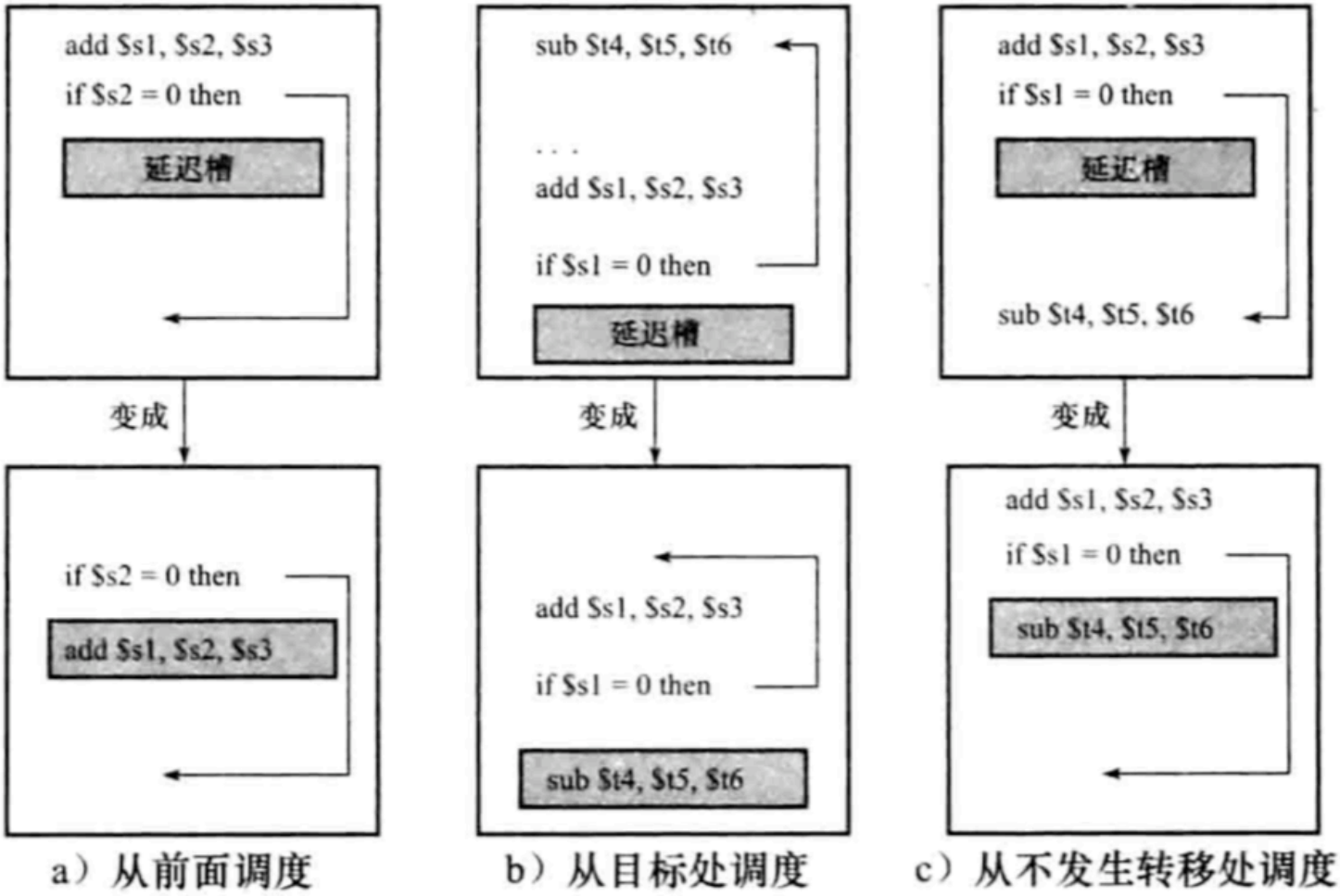
控制相关（3）

- 分支预测（Branch Prediction）：猜测分支方向并根据猜测开始取指的技术。
  - 静态分支预测：不考虑特定条件转移指令的运行时历史情况（最近是否发生过转移、发生转移的频度等），只是简单地假设该条件转移指令总是发生转移，或总是不发生转移。
  - 动态分支预测：用硬件方法来实现，是依据一条转移指令过去的行为来预测该指令将来的行为。处理器需要维护一张分支历史表BHT，该表使用条件转移指令在存储器地址的低位部分进行索引，表中每一项记录了最近一次是否发生转移，甚至目标地址等内容。即根据该指令上次的分支情况进行预测；如果预测错误，就改变历史表的相应位。

分支延迟时间槽的指令重排

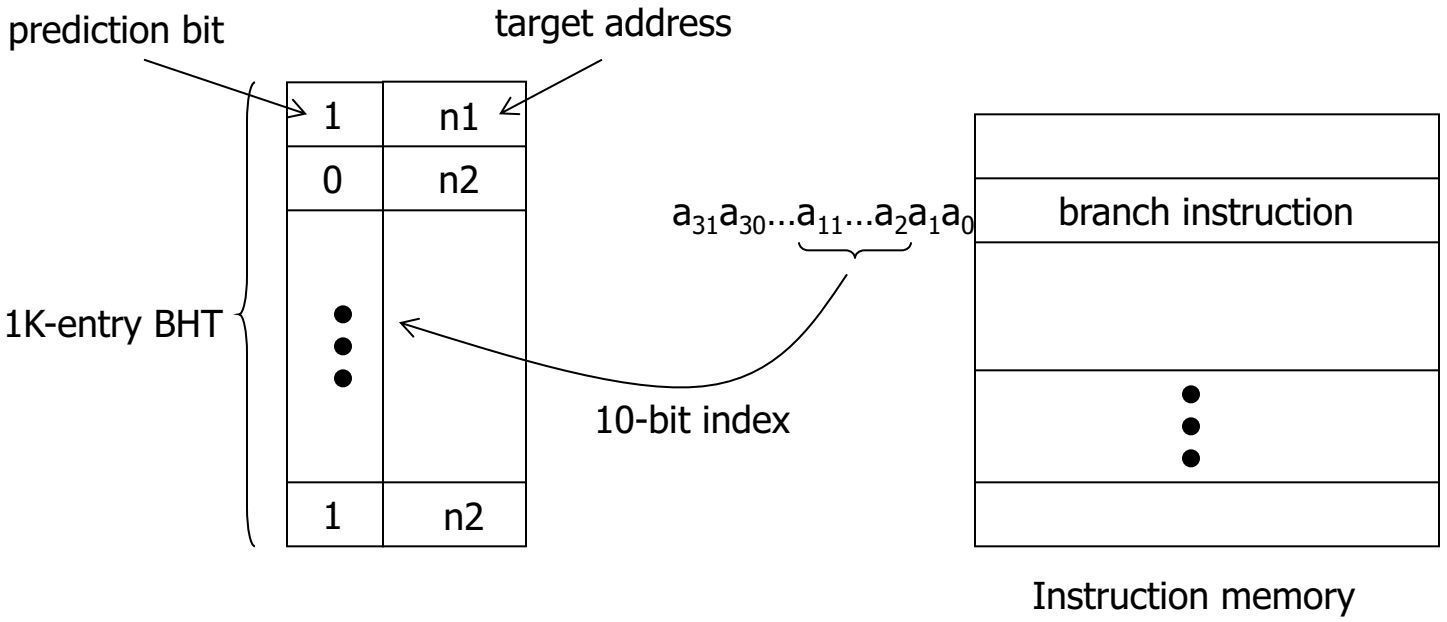
重排前的代码

重排后的代码



动态分支预测

1 = branch was last taken
0 = branch was last not taken



例4

流水线中有三类数据相关冲突：写后读相关(RAW)；读后写相关(WAR)；写后写相关(WAW)。判断以下三组指令各存在哪种类型的数据相关：

(1) I1: ADD R1,R2,R3 ； (R2) + (R3)->R1

I2: SUB R4,R1,R5 ； (R1) - (R5)->R4

(2) I3: STA M(x),R3 ； (R3)->M(x),M(x)是存储器单元

I4: ADD R3,R4,R5 ； (R4)+(R5)->R3

(3) I5: MUL R3,R1,R2 ； (R1)×(R2)->R3

I6: ADD R3,R4,R5 ； (R4)+(R5)->R3

例4解

第(1)组指令中，I1指令运算结果应先写入R1，然后在I2指令中读出R1内容。由于I2指令进入流水线，变成I2指令在I1指令写入R1前就读出R1内容，发生RAW相关。

第(2)组指令中，I3指令应先读出R3内容并存入存储单元M(x)，然后在I4指令中将运算结果写入R3。但由于I4指令进入流水线，变成I4指令在I3指令读出R3内容前就写入R3，发生WAR相关。？

第(3)组指令中，如果I6指令的加法运算完成时间早于I5指令的乘法运算时间，变成指令I6在指令I5写入R3前就写入R3，导致R3的内容错误，发生WAW相关。

例

1、下列不会引起指令流水阻塞的是（ ）

A：数据旁路

B：数据相关

C：条件转移

D：资源冲突

2、采用指令Cache与数据Cache分离的主要目的是

A 减低Cache的缺失损失

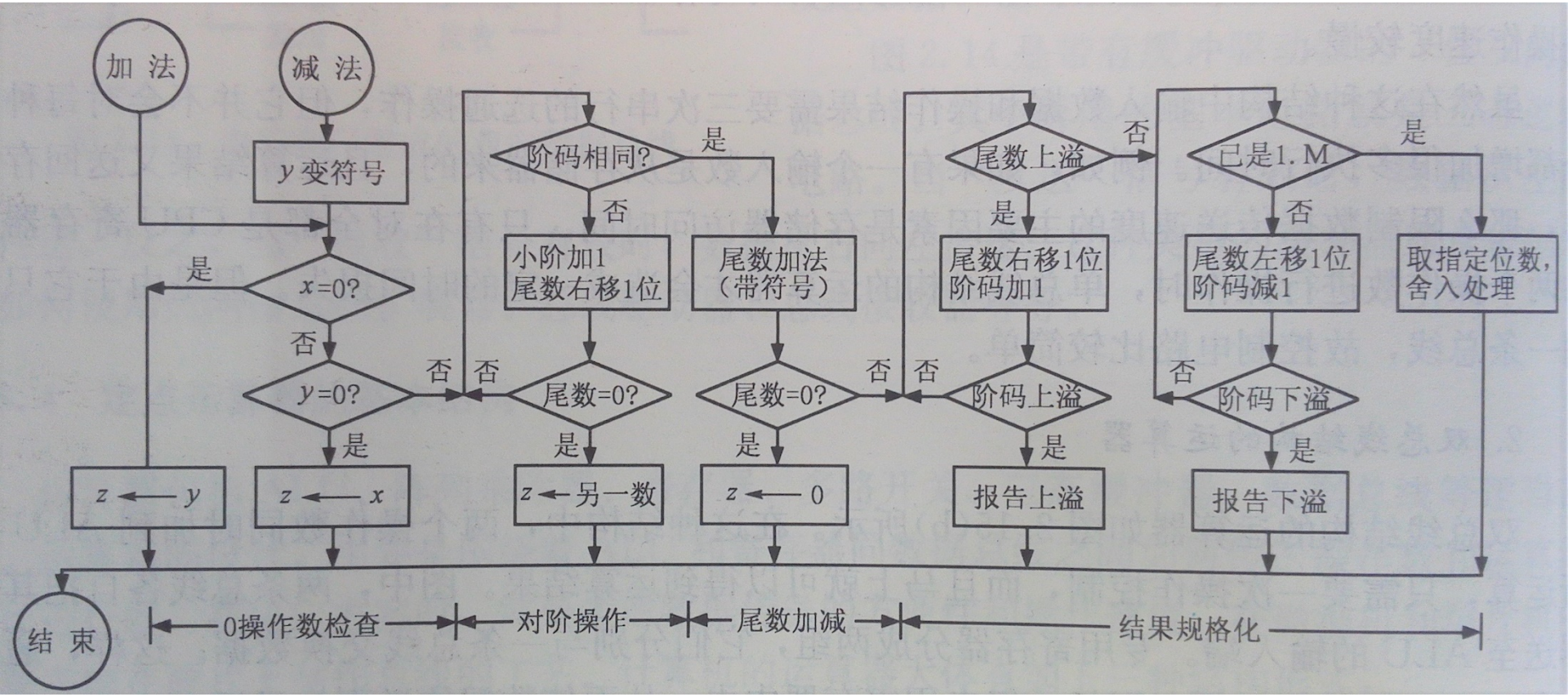
B 提高Cache的命中率

C 减低CPU平均访问时间

D 减少指令流水线资源冲突

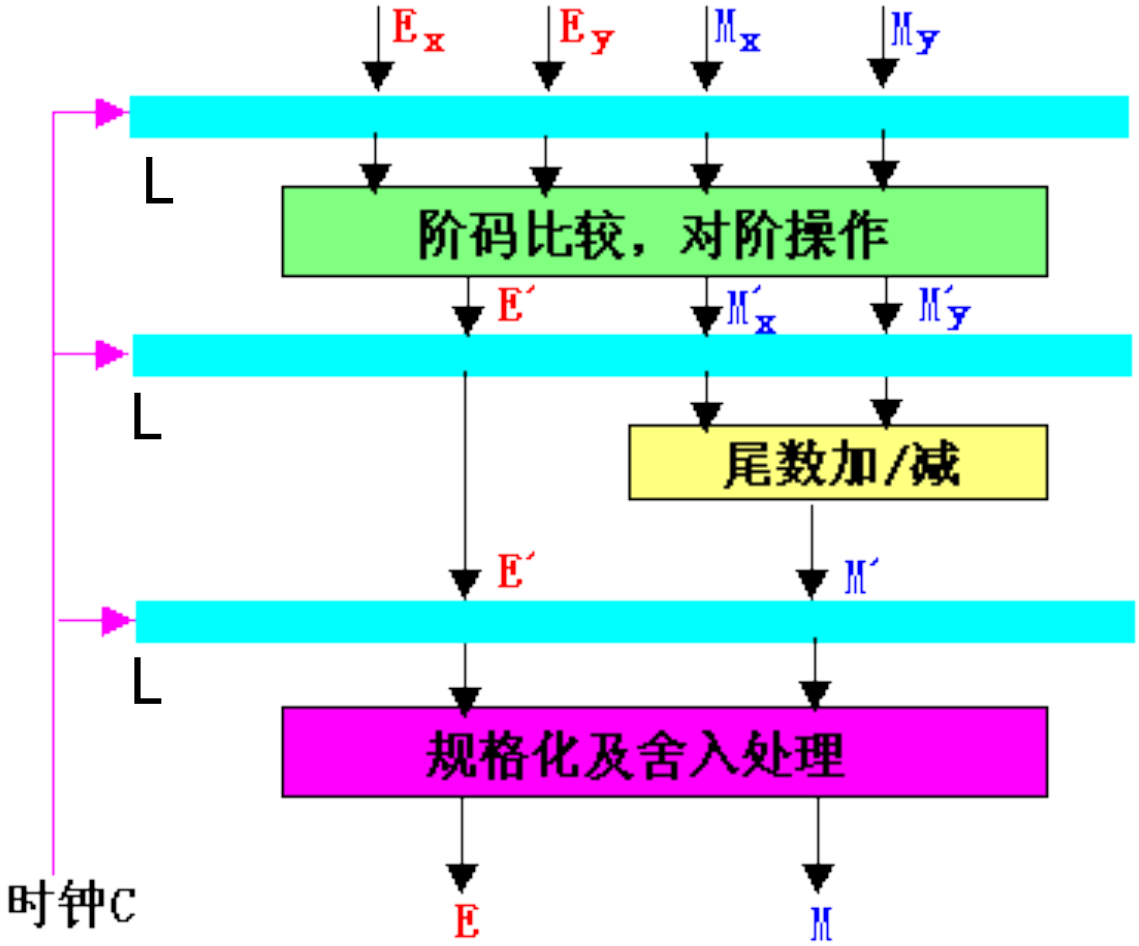
浮点加减运算操作流程

- 分为四步完成



3段流水线浮点加法器（0操作数检查除外）

- 流水线浮点加法器框图



$X=1.1000\times2^2$
$Y=1.1100\times2^4$

$E_x=2$ $E_y=4$
$M_x=1.1000$ $M_y=1.1100$

$E'=4$
$M'_x=0.0110$ $M'_y=1.1100$

$E'=4$ $M'=10.0010$

$E=5$ $M=1.0001$

例32

假设有一个4级流水浮点加法器每个过程段所需的时间为：零操作数检查τ_1＝70ns，对阶τ_2＝60ns，相加τ_3＝90ns，规格化τ_4＝80ns，缓冲寄存器L的延时时间为τ_1＝10ns，求（1）4级流水加法器的加速比为多少？

（2）如果每个过程段的时间都相同。即都为75ns（包括缓冲寄存器时间）时，加速比是多少？

例32

解：加法器的流水线时钟周期至少为：

τ＝90+10=100ns

不用流水线，所需时间为：

$\tau_1+\tau_2+\tau_3+\tau_4$＝300ns

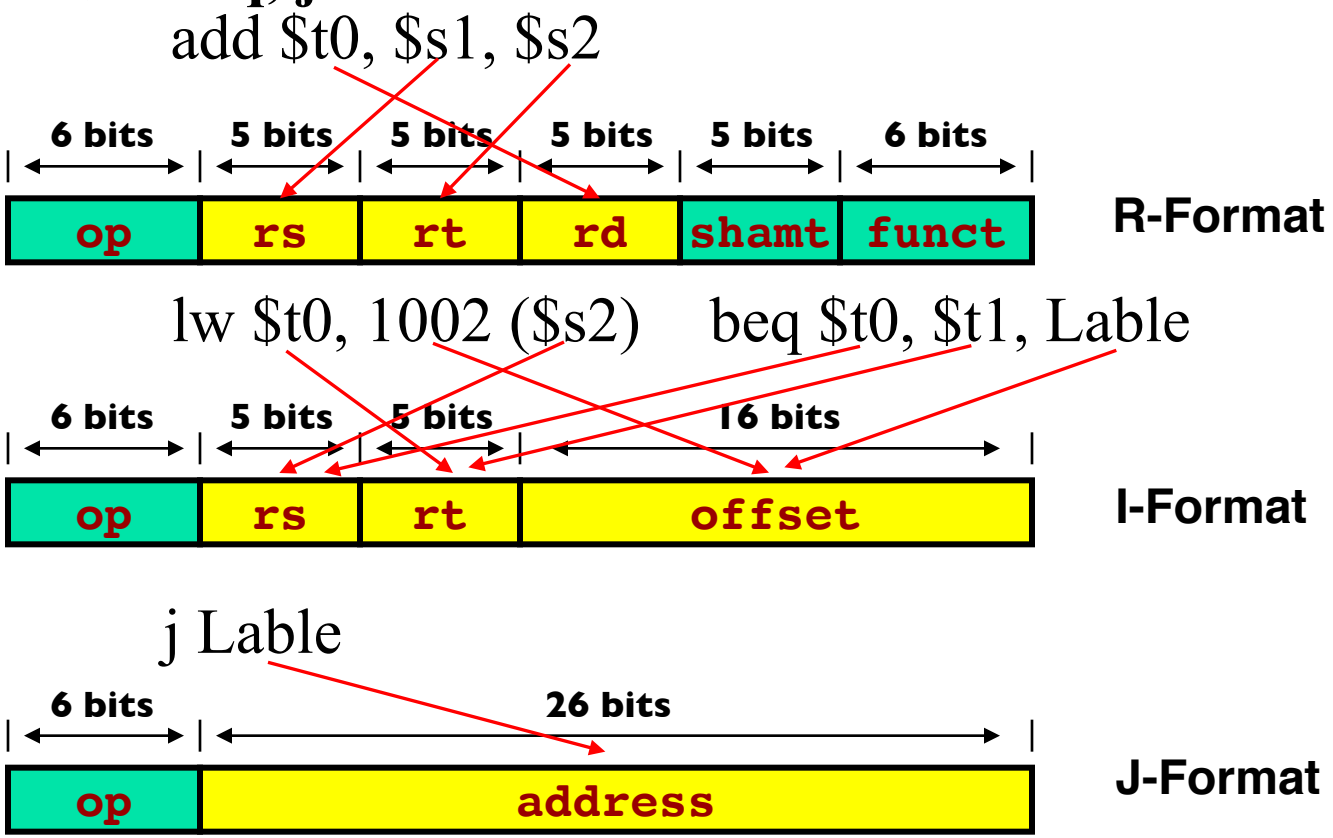
加速比为：C_k＝300/100＝3

若每过程段均为75ns，加速比为：

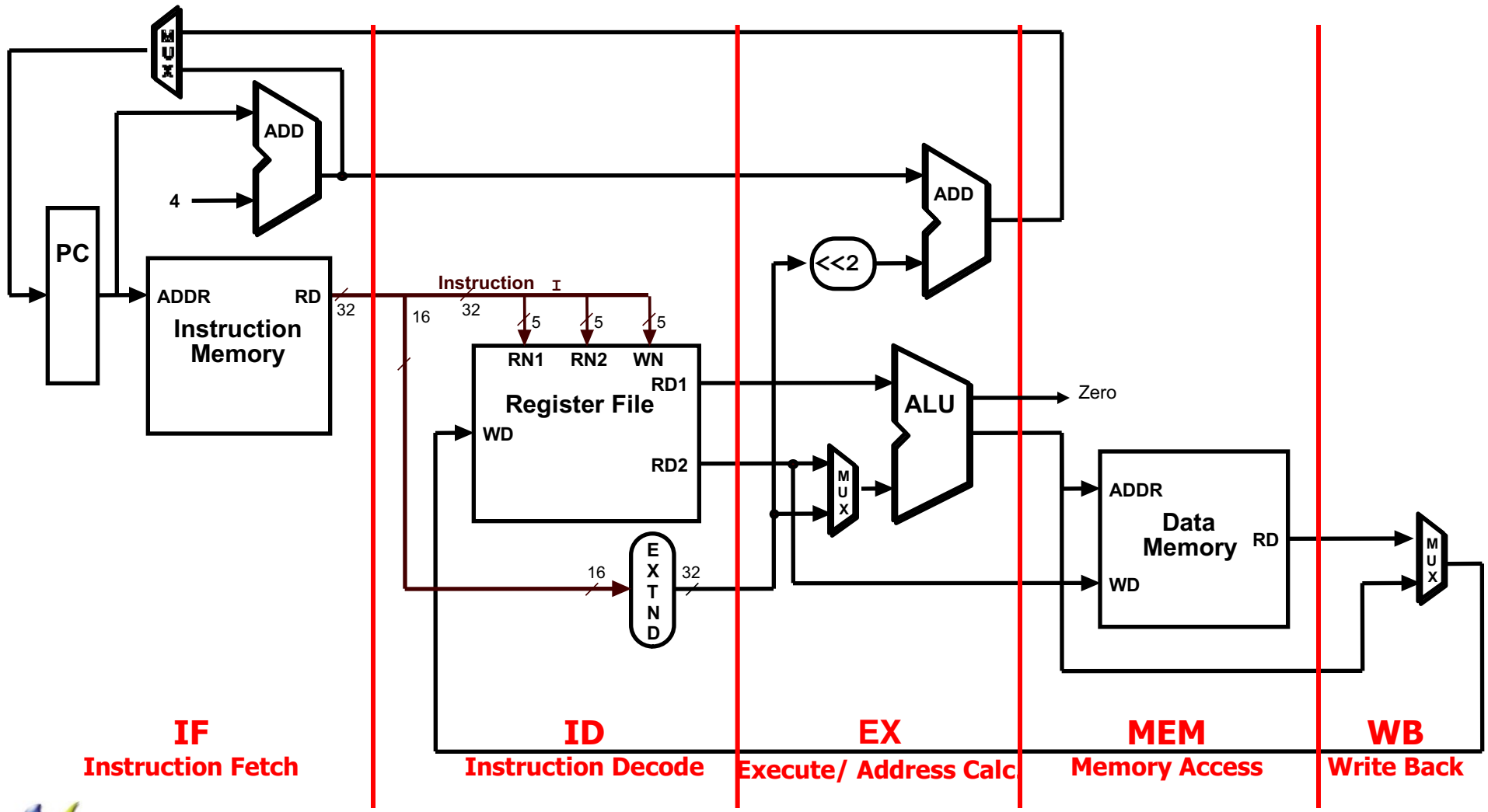
C_k＝300/75＝4

实例 MIPS指令格式

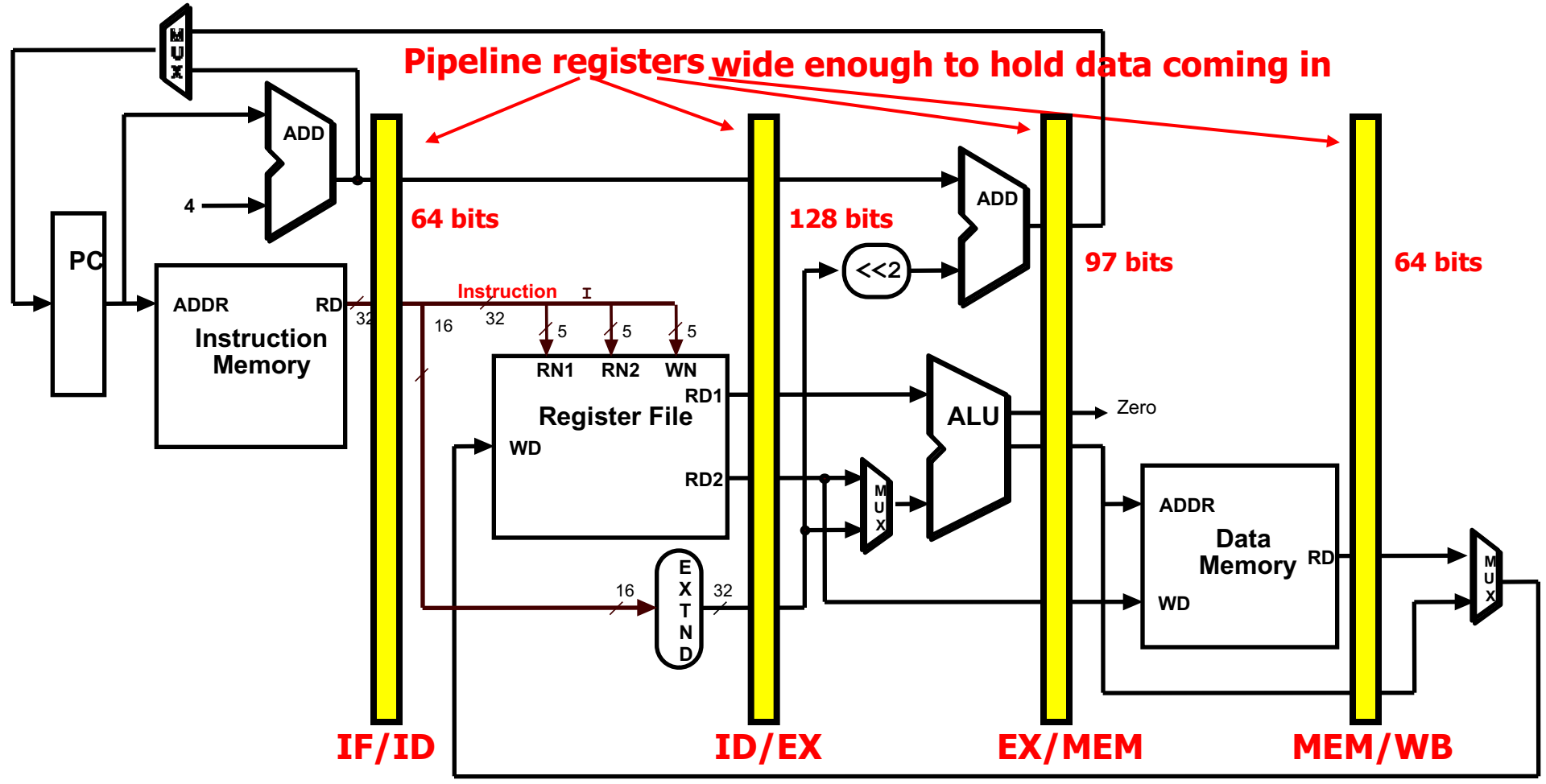
- 算术逻辑指令：add, sub, and, or, slt
- 取数/存数指令：lw, sw
- 控制流指令：beq, j



实例 MIPS数据通路图

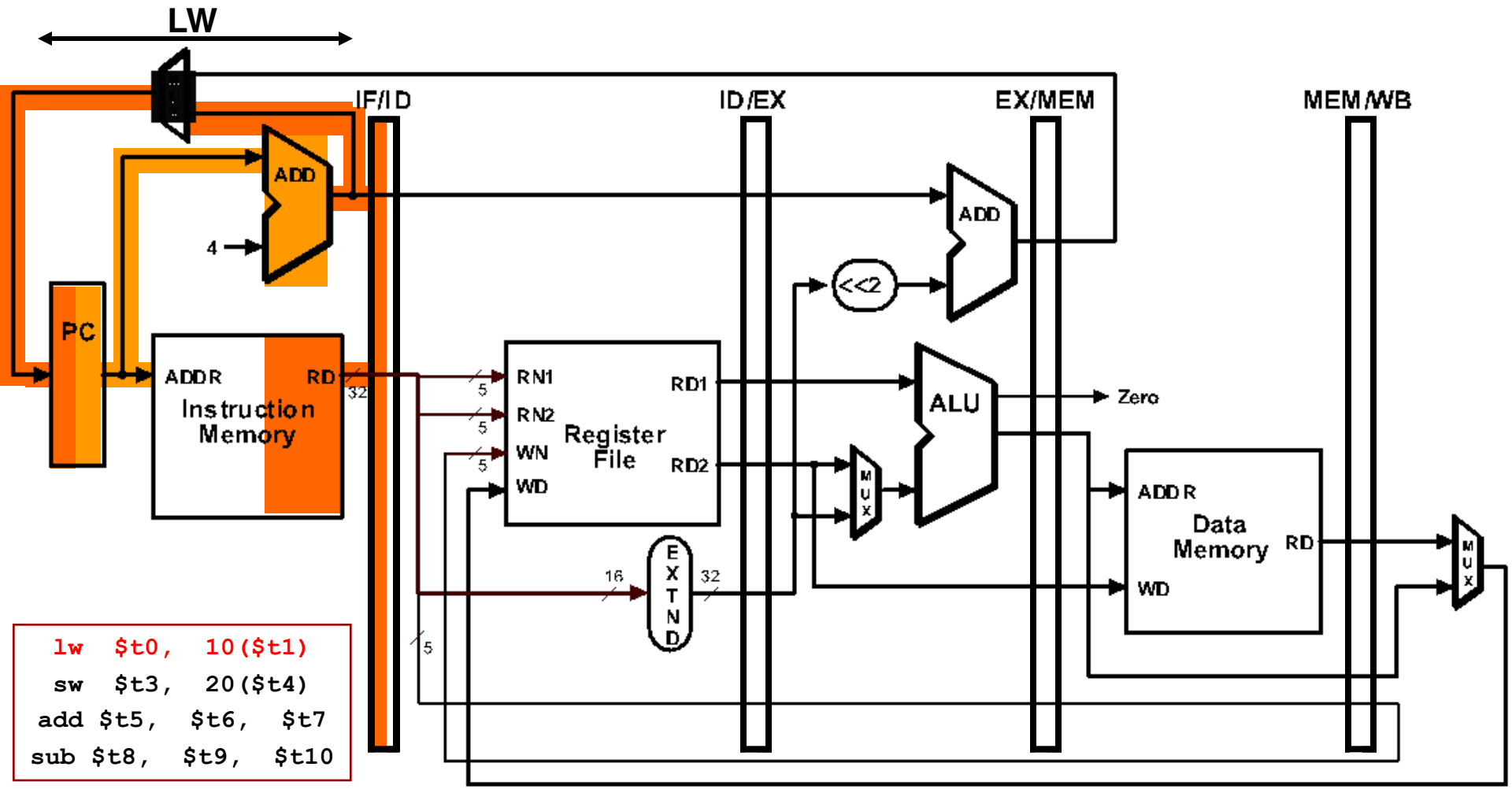


流水线MIPS数据通路图

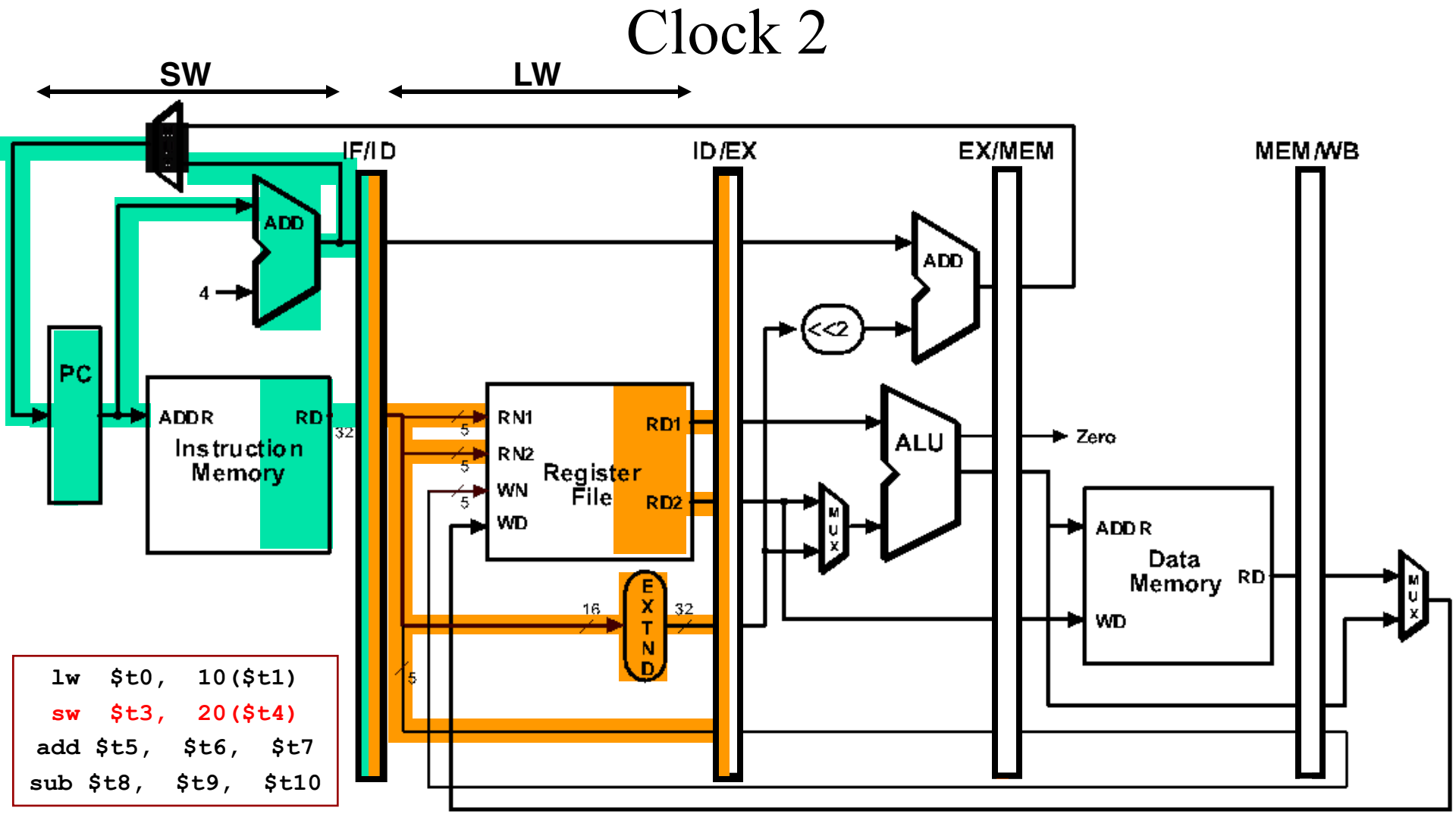


流水线数据通路图

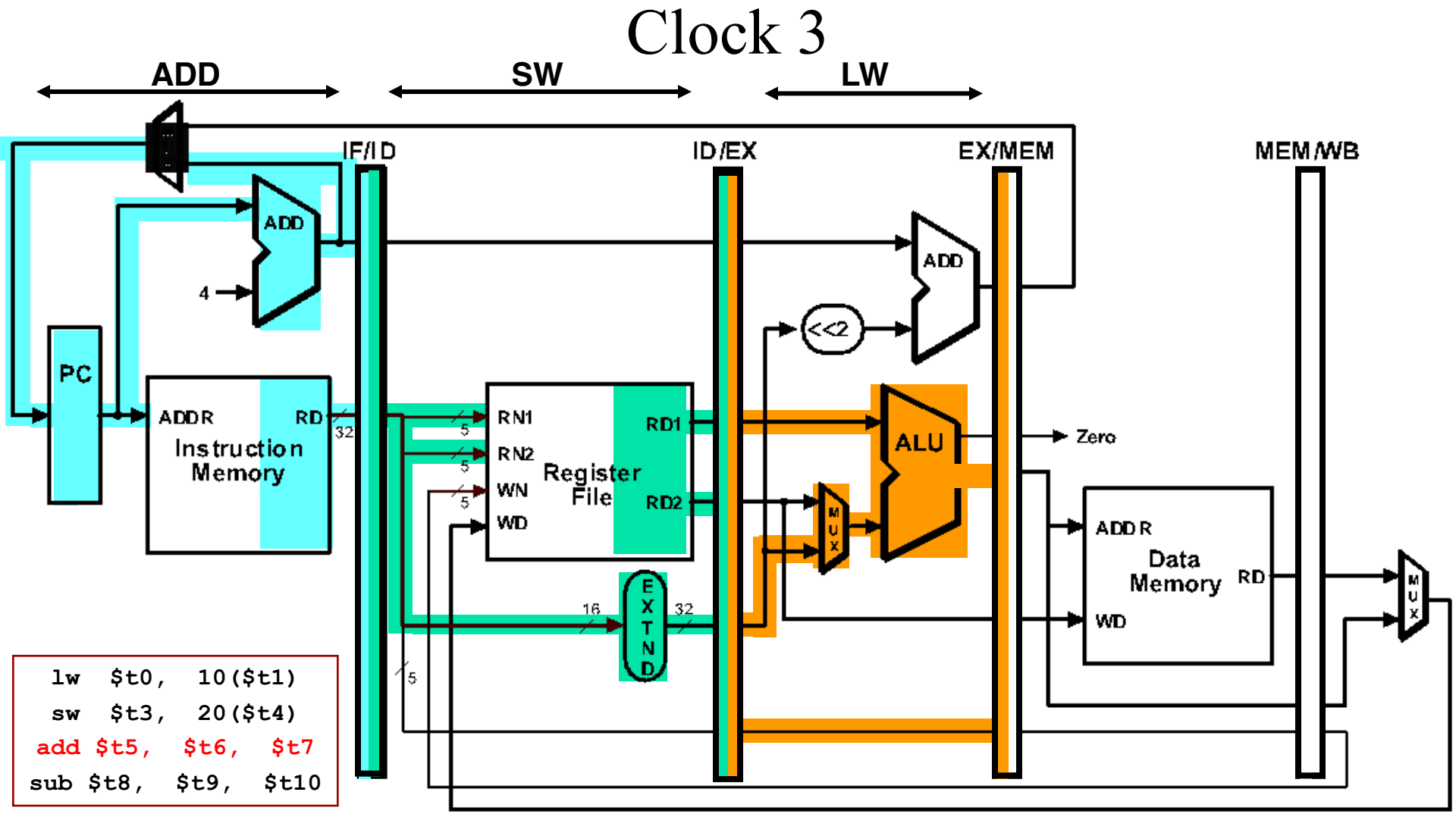
Clock 1



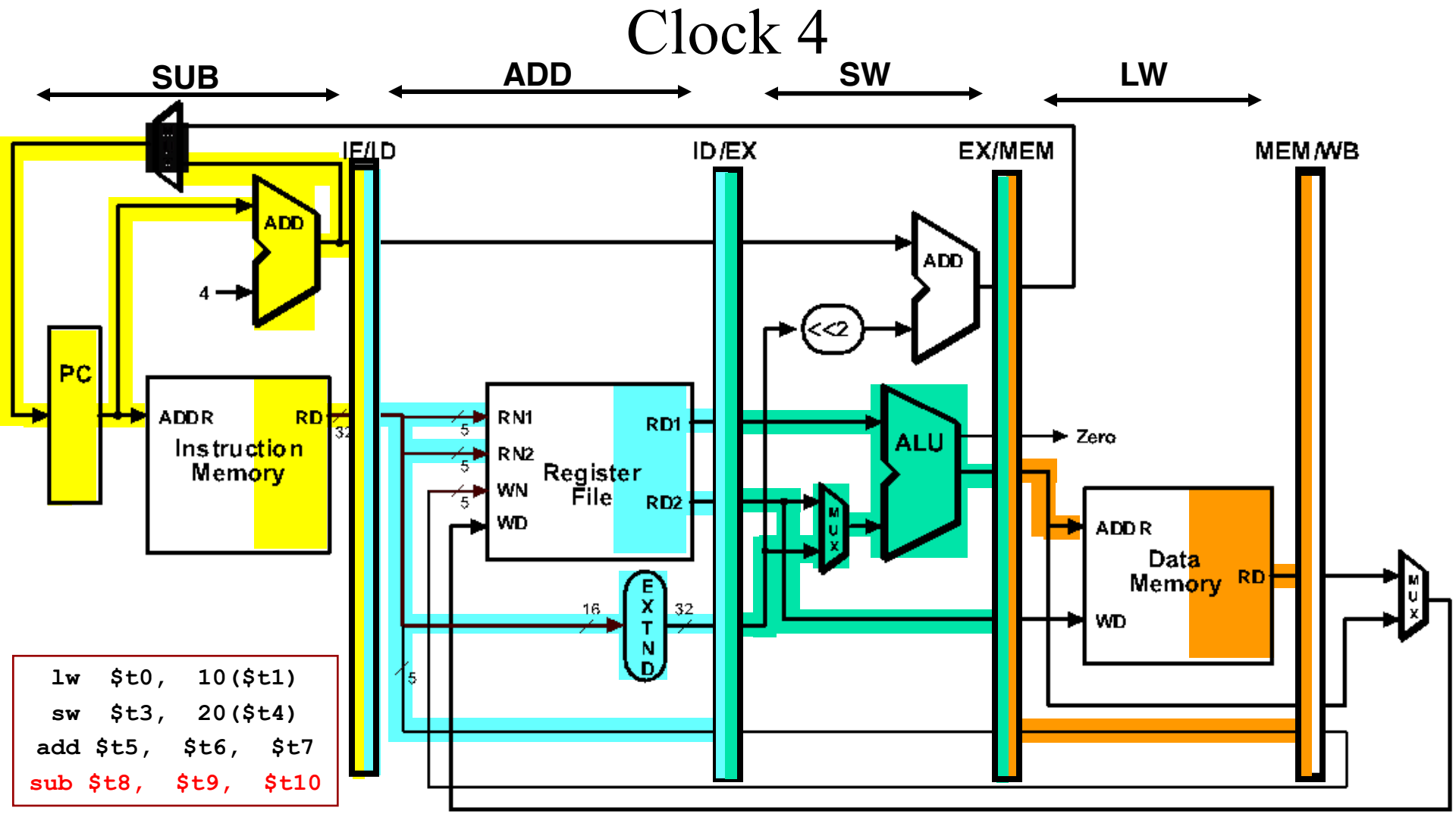
流水线数据通路图


Clock 2
SW
LW
IF/ID
ID/EX
EX/MEM
MEM/WB
ADD
4
PC
ADDR
RD
Instruction Memory
32
5
5
5
RN1
RN2
WN
WD
Register File
RD1
RD2
16
EXTND
32
5
<<2
ADD
ALU
MUX
Zero
ADDR
Data Memory
RD
WD
MUX
lw $t0, 10($t1)
sw $t3, 20($t4)
add $t5, $t6, $t7
sub $t8, $t9, $t10

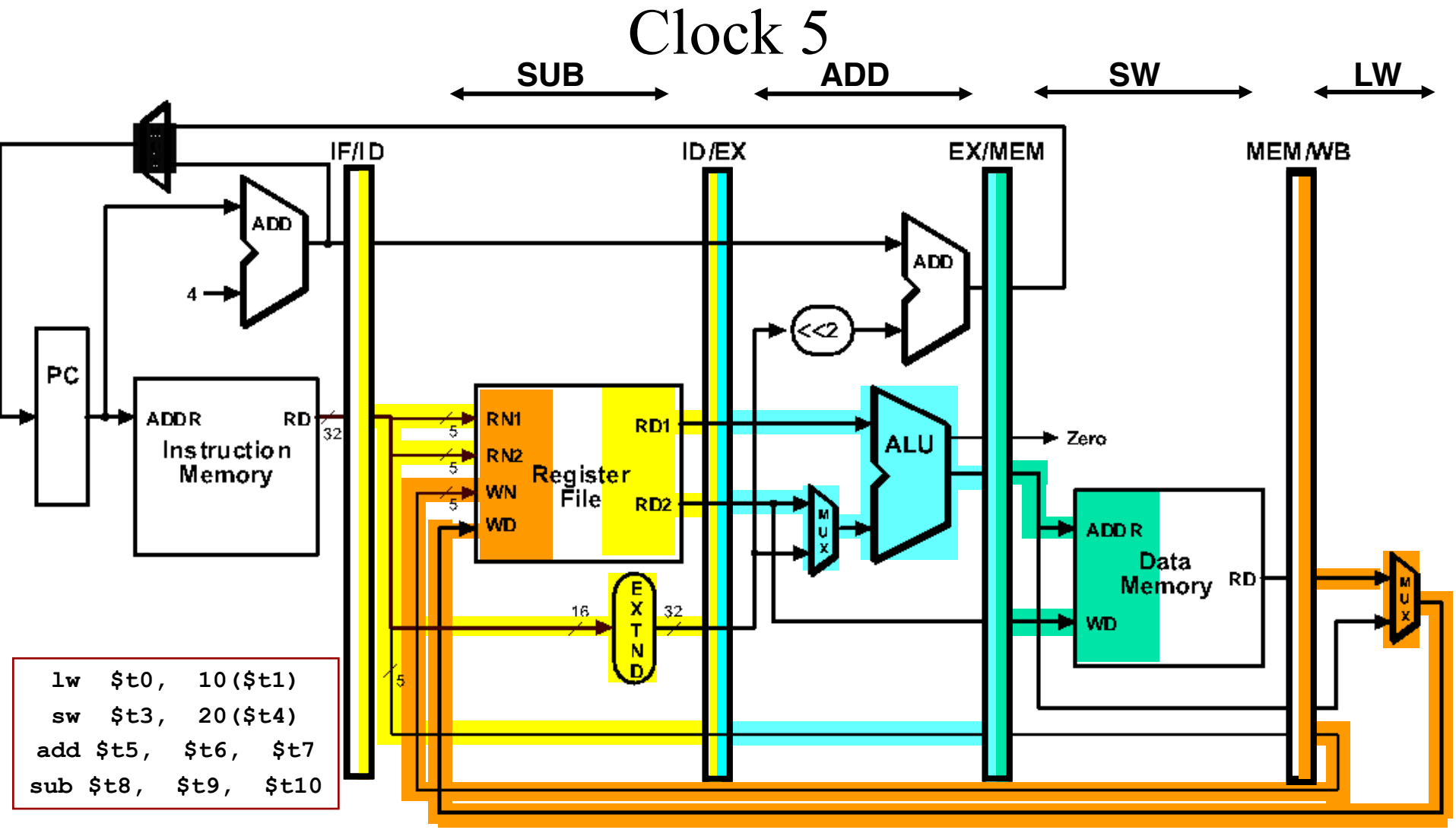
流水线数据通路图


Clock 3
ADD
SW
LW
IF/ID
ID/EX
EX/MEM
MEM/WB
ADD
4
PC
ADDR
RD
Instruction Memory
32
5
5
5
RN1
RN2
WN
WD
Register File
RD1
RD2
16
EXTND
32
5
<<2
ADD
ALU
Zero
M
U
X
ADDR
Data Memory
RD
WD
M
U
X
lw $t0, 10($t1)
sw $t3, 20($t4)
add $t5, $t6, $t7
sub $t8, $t9, $t10

流水线数据通路图


Clock 4
SUB
ADD
SW
LW
IF/ID
ID/EX
EX/MEM
MEM/WB
ADD
4
PC
ADDR
RD
Instruction Memory
32
5
RN1
RN2
WN
WD
Register File
RD1
RD2
<<2
ADD
ALU
Zero
MUX
16
EXTND
32
ADDR
Data Memory
RD
WD
MUX
lw $t0, 10($t1)
sw $t3, 20($t4)
add $t5, $t6, $t7
sub $t8, $t9, $t10

流水线数据通路图


Clock 5
SUB
ADD
SW
LW
IF/ID
ID/EX
EX/MEM
MEM/WB
ADD
4
PC
ADDR
RD
Instruction Memory
32
5
RN1
RN2
WN
WD
Register File
RD1
RD2
<<2
ADD
ALU
Zero
MUX
16
EXTND
32
ADDR
Data Memory
RD
WD
MUX
lw $t0, 10($t1)
sw $t3, 20($t4)
add $t5, $t6, $t7
sub $t8, $t9, $t10

5.7 RISC CPU

RISC处理器特点

- 1981年美国加州大学伯克利分校的David Patterson主持设计了第一台精简指令系统计算机（RISC：Reduced Instruction Set Computer）

- RISC的三个要素
  - 一个有限的简单的指令集
  - CPU具有大量的通用寄存器
  - 强调对指令流水线的优化

- SUN, SPARC(1987)
- MIPS, SGI:MIPS(1986)
- HP, PA-RISC,
- IBM与Motorola, PowerPC
- DEC、Compaq, Alpha AXP
- IBM RS6000(1990) first Superscalar RISC

RISC处理器特点（1）

- 使用等长指令，如：典型长度是4个字节
- 寻址方式少且简单，一般为2～3种，最多不超过4种，没有存储器间接寻址方式
- 只有取数指令、存数指令能访问存储器，指令中最多出现RR型指令，没有SS型指令
- 指令集中的指令数目一般少于100种，指令格式一般少于4种
- 指令功能简单，控制器多采用硬布线方式

RISC处理器特点（2）

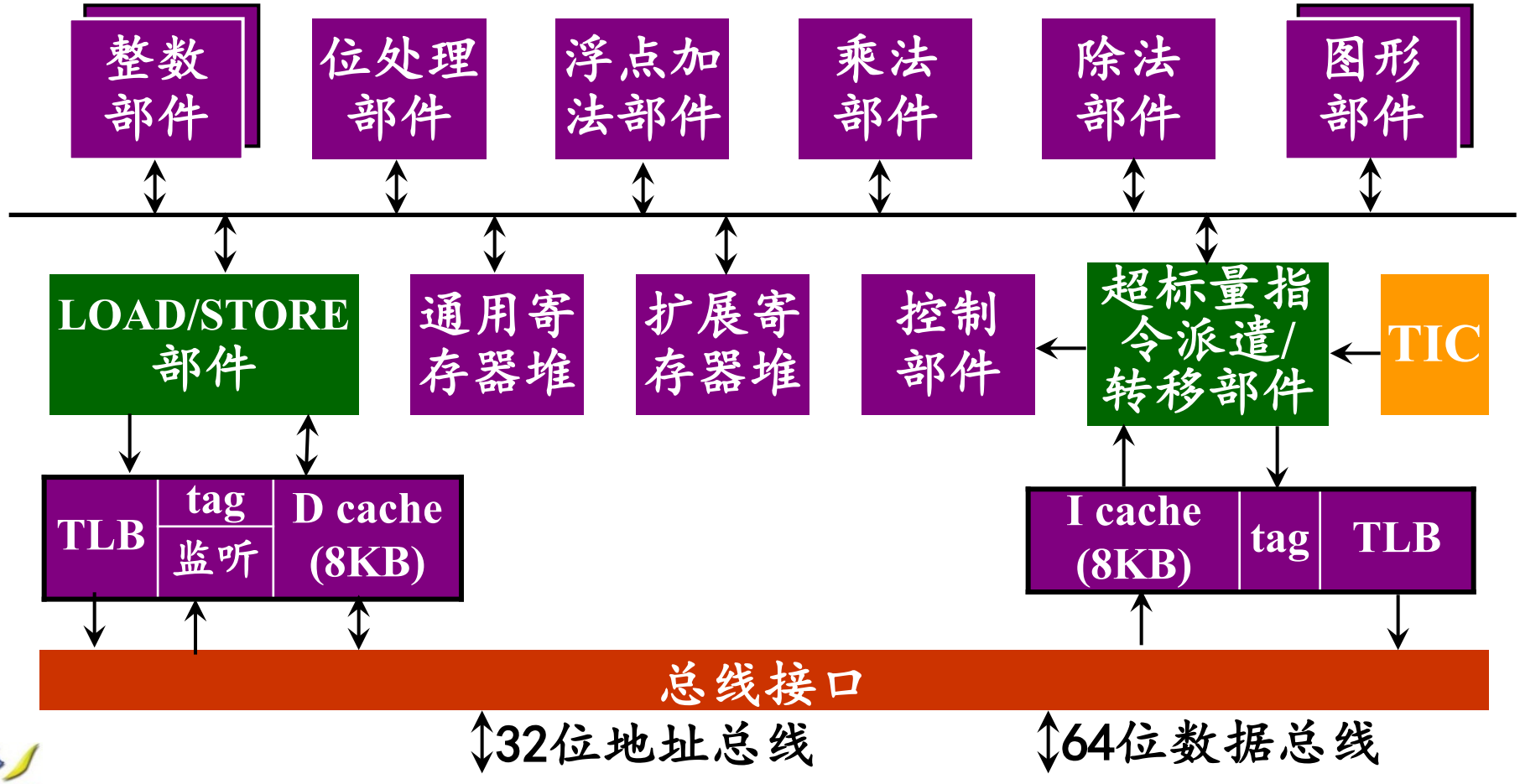
- 平均而言，指令的执行时间为一个时钟周期
- 指令格式中用于指派整数寄存器的个数不少于32个，用于指派浮点数寄存器的个数不少于16个
- 强调通用寄存器资源的优化使用
- 支持指令流水并强调指令流水的优化使用
- RISC技术的复杂性于它的编译程序，因此系统软件开发时间比CISC机器长

MC88110 CPU

- Motorola公司的RISC CPU产品
- 目标是以较好的性能价格比作为PC和工作站的通用微处理器
- 有12个执行功能部件、3个cache和1个控制部件

MC88110 CPU结构框图

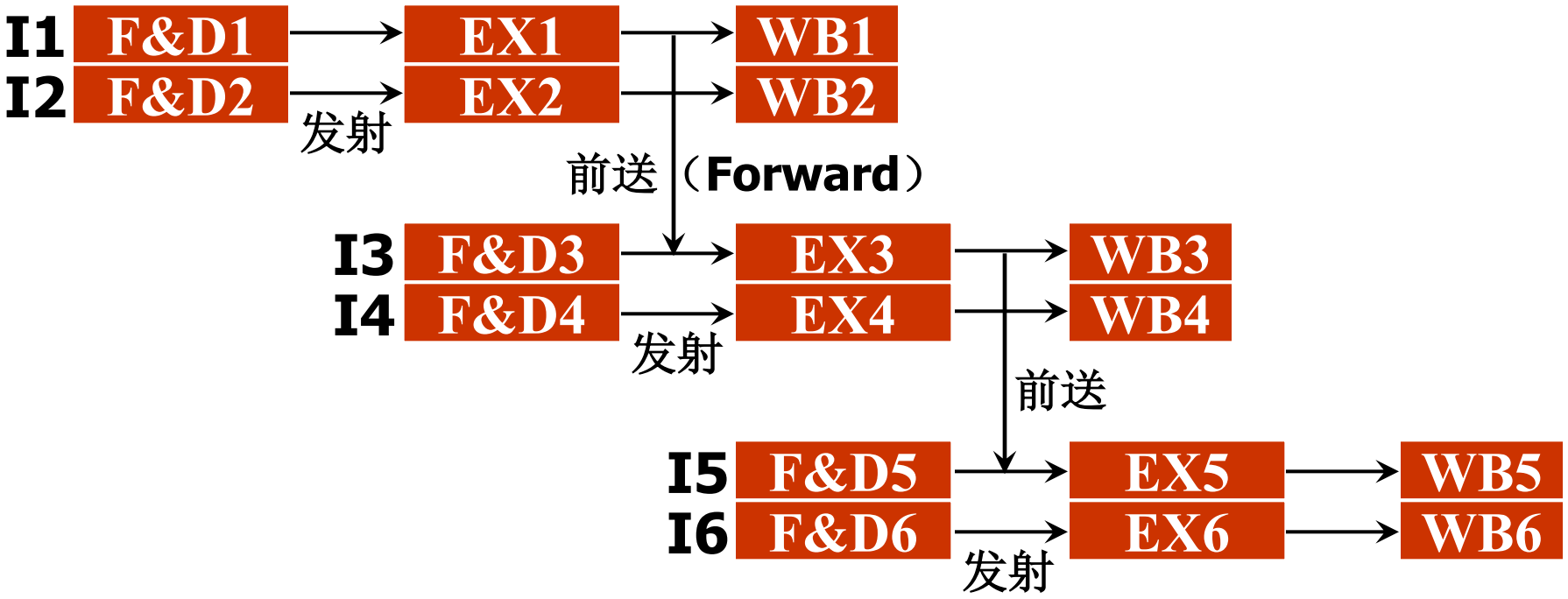
在处理器中的所有这些Cache、寄存器堆、功能部件，通过六条80位宽的内部总线相连接（2条源1总线、2条源2总线、2条目标总线）



MC88110的指令流水线

- MC 88110是超标量流水CPU，在每个机器周期完成2条指令
- 流水线共分为三阶段
  - 取指和译码(F＆D)段：需要一个时钟周期，完成从指令cache取一对指令并译码，并从寄存器堆取操作数，然后判断是否把指令发射到EX段
  - 执行(EX)段：EX段通常只需一个时钟周期，某些指令可能多于一个时钟周期
  - 写回(WB)段
    - EX段执行的结果在WB段写回段寄存器堆，WB段只需时钟周期的一半；
    - 为了解决数据相关冲突，EX段执行的结果同时通过前送（Forwarding）电路提前传送到ALU

MC88110的指令流水线

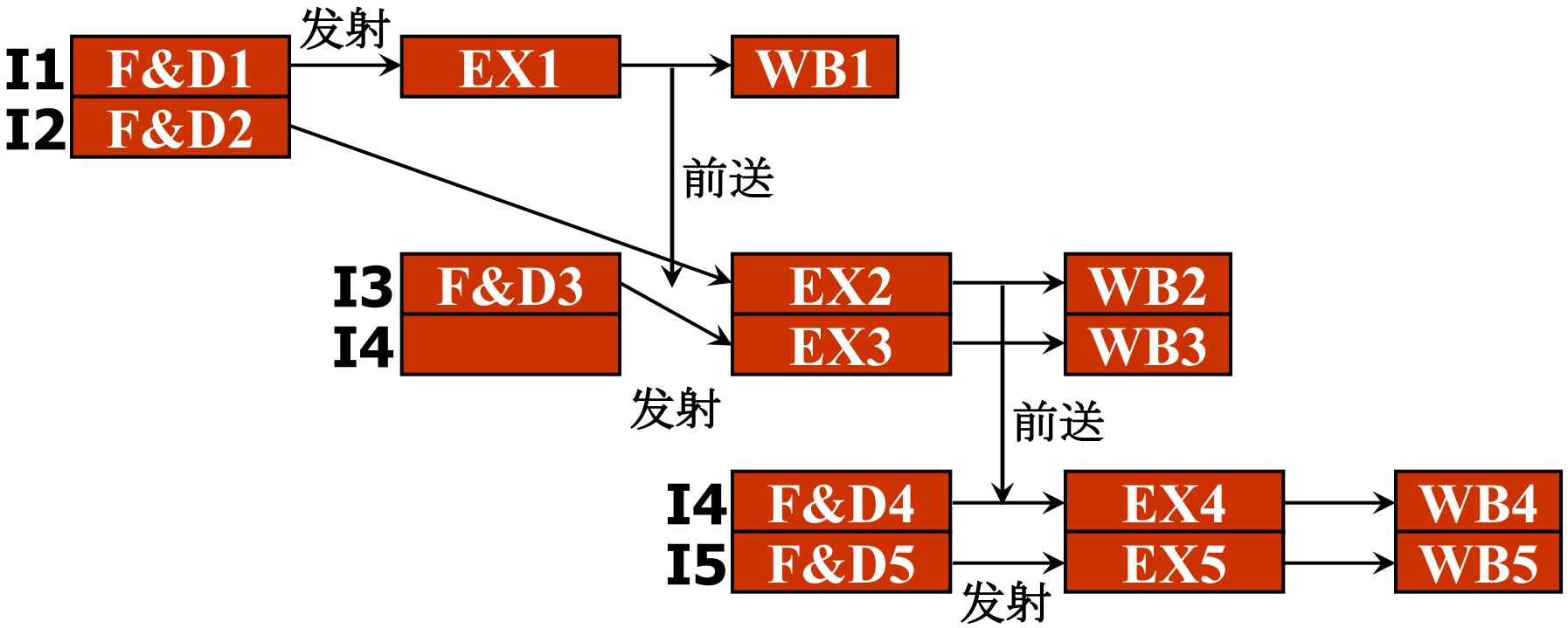


指令动态调度策略（1）

88110采用按序发射、按序完成的指令动态调度策略

- 指令译码后总是力图同一时间发射这两条指令到EX段
- 若这对指令的第一条指令由于资源冲突或数据相关冲突，则这一对指令都不发射
- 若是第一条指令能发射第二条指令不能发射，则只发射第一条指令，而第二条指令停顿并与新取的指令之一进行配对等待发射

指令动态调度策略示例


发射
I1
F&D1
I2
F&D2
EX1
WB1
前送
I3
F&D3
I4
EX2
EX3
WB2
WB3
发射
前送
I4
F&D4
I5
F&D5
EX4
EX5
WB4
WB5
发射

指令动态调度策略（2）

- 为了判定能否发射指令，88110使用了计分牌方法
  - 计分牌是一个位向量， 寄存器堆中每个寄存器都有一个对应位
  - 每当一条指令发射时，它预约的目的寄存器在位向量中的对应位置“1”，表示该寄存器“忙”
  - 当指令执行完毕并将结果写回此目的寄存器时，该位被清除
- 如何实现按序完成呢?
  - 88110提供了一个FIFO指令执行队列，称之为历史缓冲器
  - 每当一条指令发射出去，它的副本就被送到FIFO队尾，队列最多能保存12条指令
  - 只有前面的所有指令执行完，这条指令才到达队首，当它到达队首并执行完毕后才离开队列

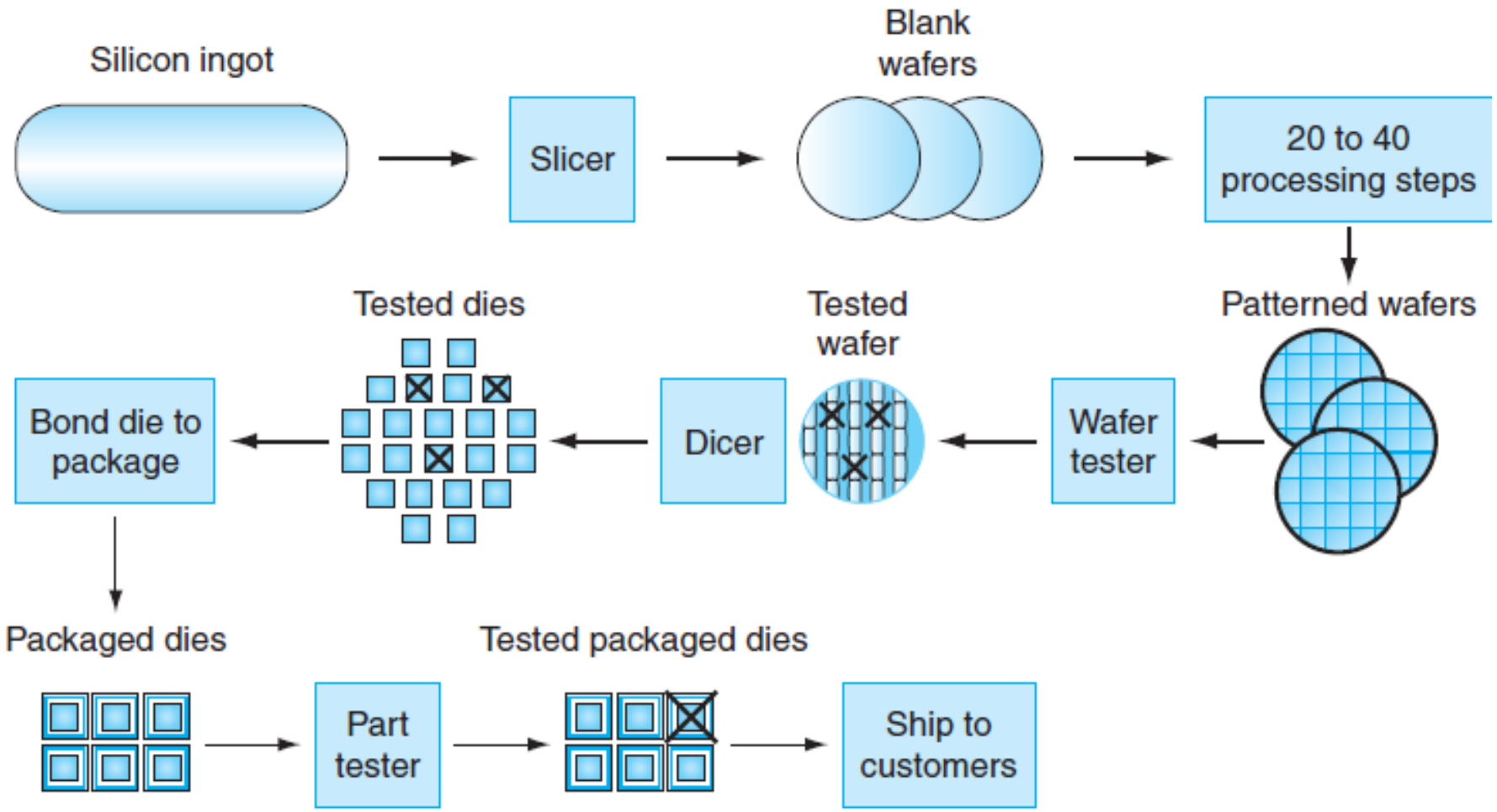
指令动态调度策略（3）

- 对于转移处理，88110使用了延迟转移法和目标指令cache(TIC)法
  - 延迟转移是可选项(.n)，如果采用这个可选项(指令如bcnd.n)，则跟随在转移指令后的指令将被发射；否则，在转移指令发射之后的转移延迟时间内不发射任何指令。是否采用延迟转移是通过编译程序来决定的
  - TIC是一个32项的全相联cache，每项能保存转移目标路径的前两条指令。当一条转移指令译码并命中Cache时，就能在TIC表项中取出目标路径的前两条指令

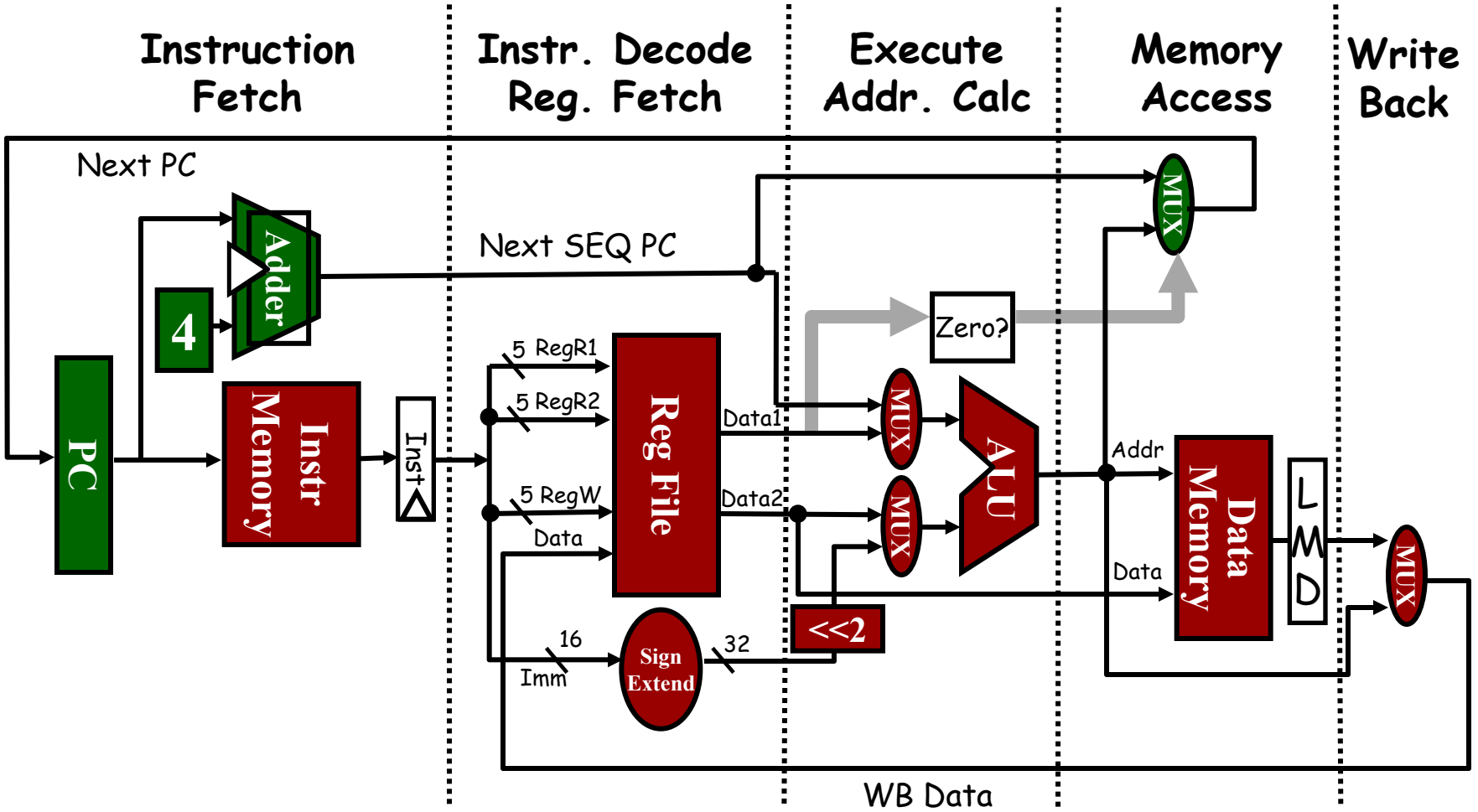
例5

- 教材第180页例5

知识拓展 芯片制造过程


Silicon ingot
Slicer
Blank wafers
20 to 40 processing steps
Patterned wafers
Wafer tester
Tested wafer
Dicer
Tested dies
Bond die to package
Packaged dies
Part tester
Tested packaged dies
Ship to customers

MIPS 数据通路


Instruction Fetch
Instr. Decode Reg. Fetch
Execute Addr. Calc
Memory Access
Write Back
Next PC
Adder
4
PC
Instr Memory
Inst
Next SEQ PC
5 RegR1
5 RegR2
5 RegW
Data
Reg File
Data1
Data2
16
Imm
Sign Extend
32
<<2
MUX
Zero?
ALU
Addr
Data Memory
Data
LMD
WB Data