处理器

Part 2：流水线

本讲提纲

- 单周期CPU和多周期CPU
  - 单周期CPU特点
  - 多周期CPU特点
- 流水线概述
  - 流水线概念
  - 流水线实现原理
  - 流水线性能指标
  - MIPS流水线的实现思路

本讲提纲

- 单周期CPU和多周期CPU
  - 单周期CPU特点
  - 多周期CPU特点
- 流水线概述
  - 流水线概念
  - 流水线实现原理
  - 流水线性能指标
  - MIPS流水线的实现思路
- 流水线冒险
  - 结构冒险、数据冒险、控制冒险

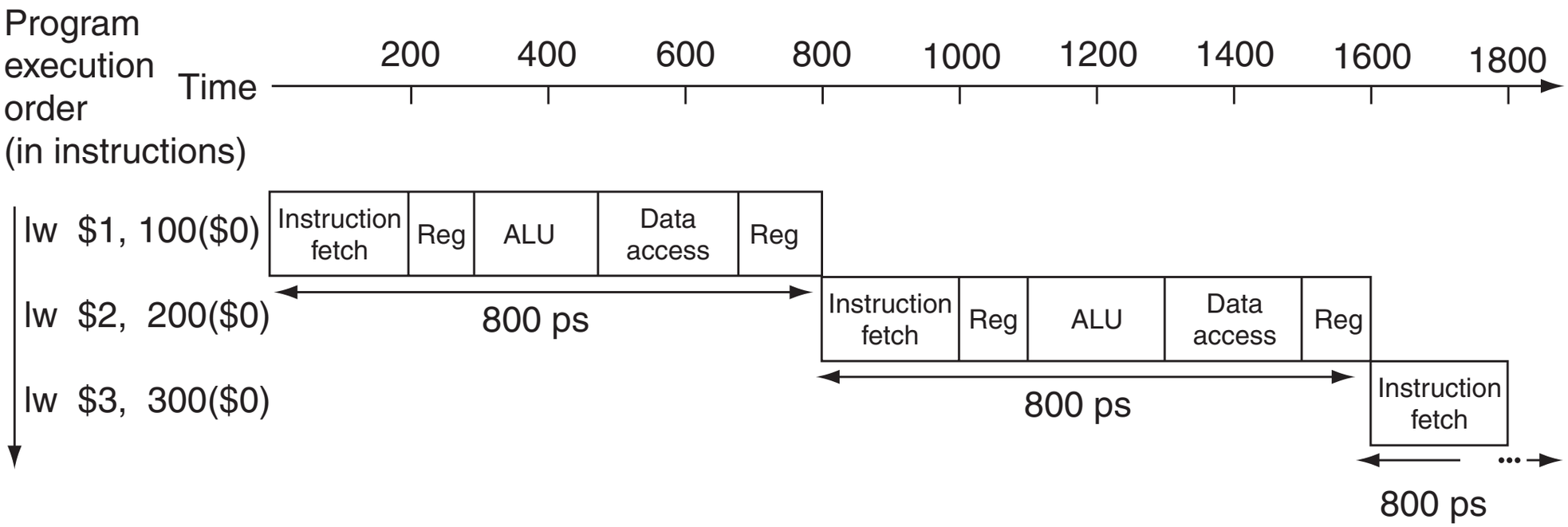
单周期CPU

计算机一条指令的执行时间被称为指令周期，一个CPU时钟时间被称为 CPU周期 (在某些计算机中，还可再把一个CPU周期区分为几个更小的步骤，称其为节拍)。执行每条指令平均使用的 CPU周期个数 被称为 CPI

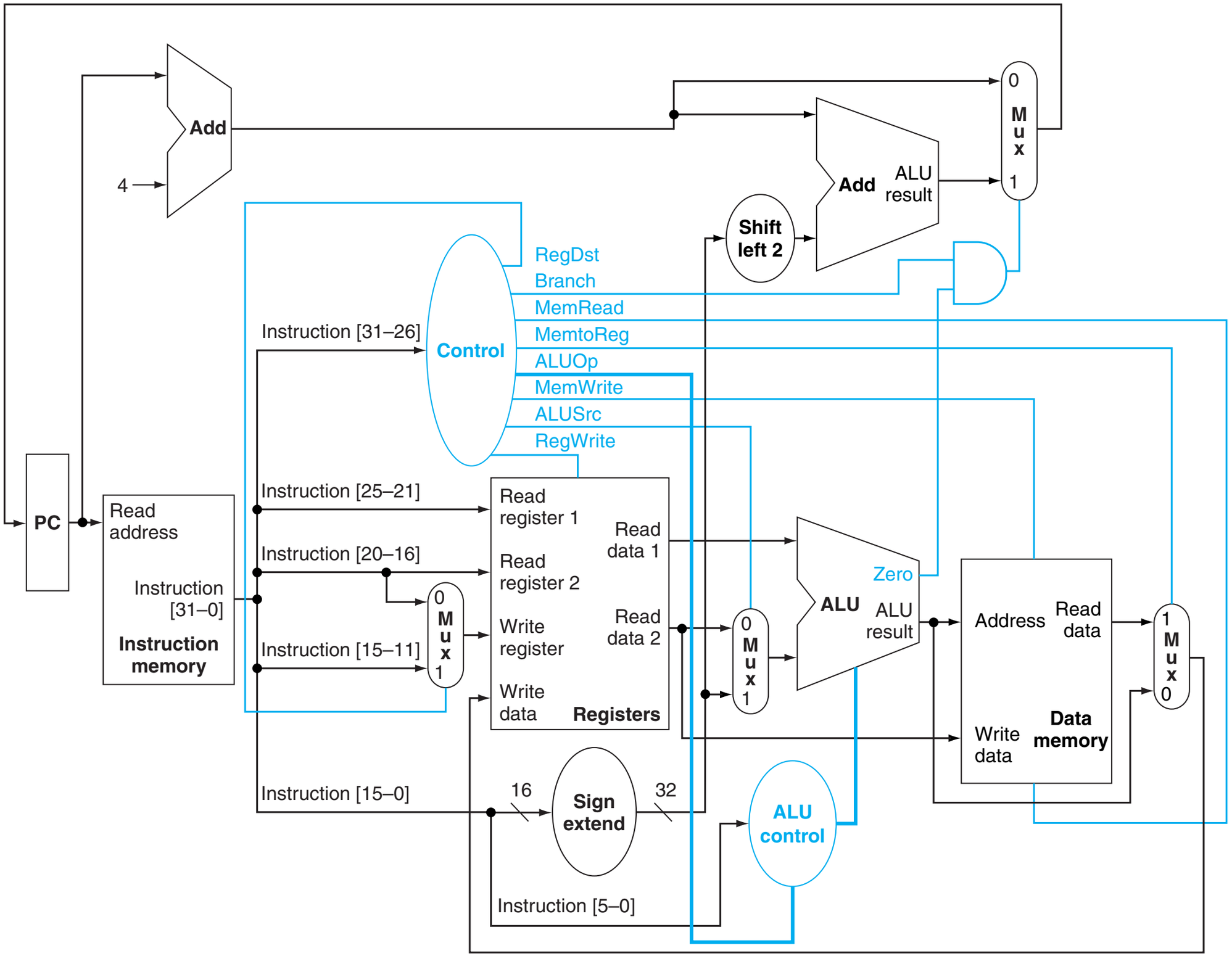
全部指令都选用 一个 CPU周期 完成的系统被称为单周期CPU，指令串行执行，前一条指令结束后才启动下条指令。每条指令都用 5个步骤的时间完成，控制各部件运行的信号在整个指令周期不变化。单周期CPU 用于早期计算机，系统性能和资源利用率很低，相对当前技术变得不再实用。

执行指令的基本步骤

- 取指:从指令存储器中读指令(地址:PC)
- 读出一或两个源寄存器的值(寄存器组)
- 进行指令规定的运算(ALU)或计算地址
- 读/写数据存储器
- 将结果写入目的寄存器



单周期CPU


Add
4
0
M
u
x
1
Add
ALU
result
Shift
left 2
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
PC
Read
address
Instruction
[31–0]
Instruction
memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read
register 1
Read
register 2
Write
register
Write
data
Read
data 1
Read
data 2
Registers
16
Sign
extend
32
ALU
control
Zero
ALU
ALU
result
Address
Read
data
Write
data
Data
memory
1
M
u
x
0

单周期CPU的控制信号

- 每条指令占用一个时钟周期
  - 取指令后分析指令,并给出整个执行期间的全部信号
  - 不需要状态信息,在时钟的结束的边沿写入结果
- 控制对象
  - ALU的运算
  - 寄存器组和存储器的写入
  - 多路选通器
- 时钟周期开始时读取指令
  - 与具体指令无关

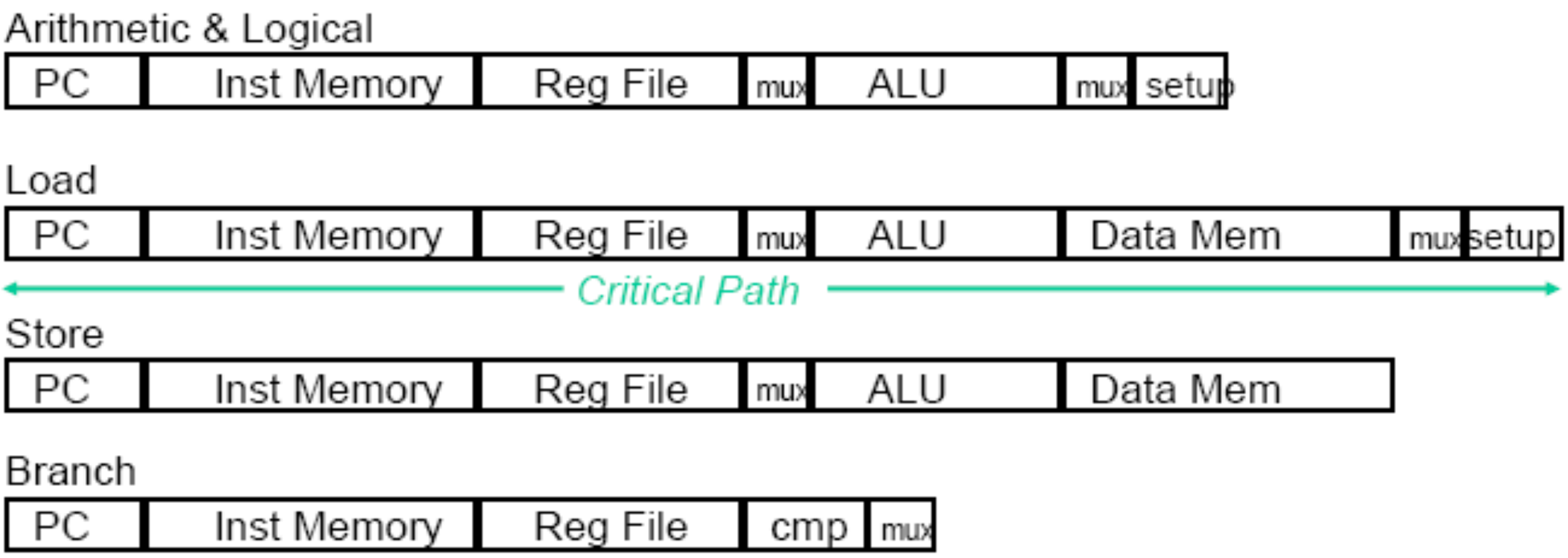
单周期CPU的特点

- 优点
  - 每条指令占用一个时钟周期
  - 逻辑设计简单,时钟设计也简单
- 缺点
  - 各组成部件的利用率不高
    - 各部件大部分时间在等待
  - 时钟周期应满足执行时间最长指令的要求
    - Load指令
- CPI =1

单周期CPU的性能

- 假定某单周期CPU各主要部件的延迟为:
  - 存储器(Memory):2ns
  - 运算器(ALU/Adder):2ns
  - 寄存器组(Register File):1ns

单周期CPU的性能



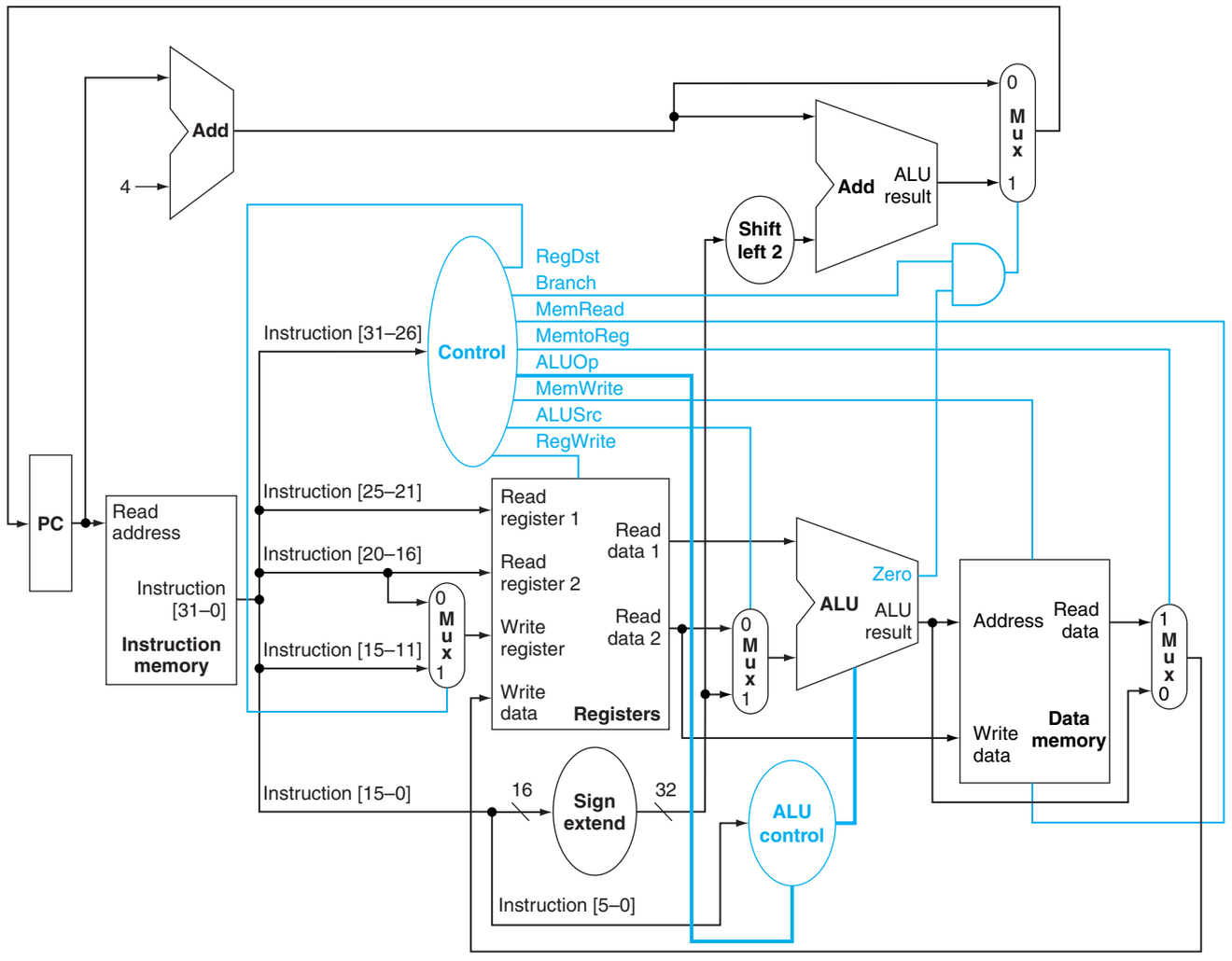
- 指令周期比较长
- 所有指令都必须使用最长的周期

单周期CPU的性能

- 假设某单周期CPU,执行100条指令:
  - 25%的Load指令
  - 10%的Store指令
  - 45%的算逻指令
  - 20%的跳转指令
- 单周期的执行时间
  - 100*8 = 800ns
- 可能的优化
  - 25*8+10*7+45*6+20*5=640ns
  - Speedup=800/640=1.25

单周期CPU的其他问题

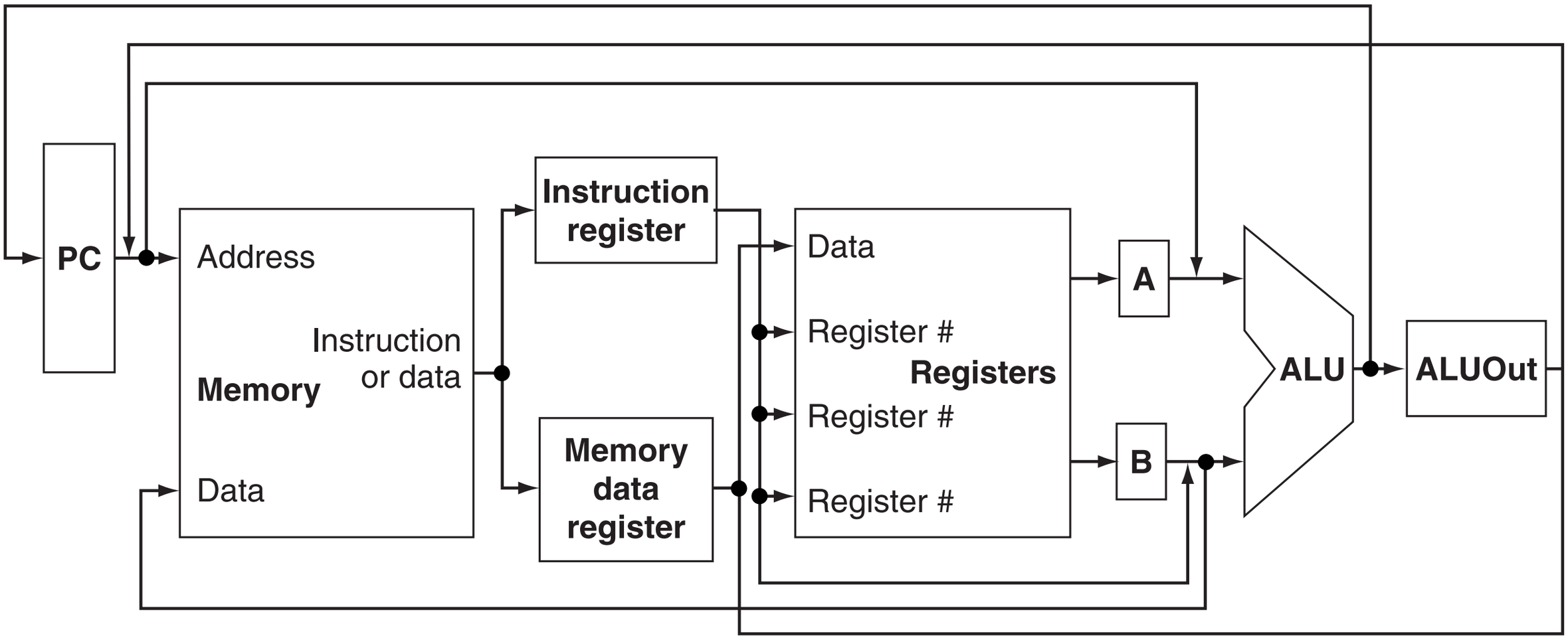
- 指令和数据都存在同一个存储器中
- 许多部件保持数据的时间过长,无法复用
  - 例如, Add是否可以利用ALU?



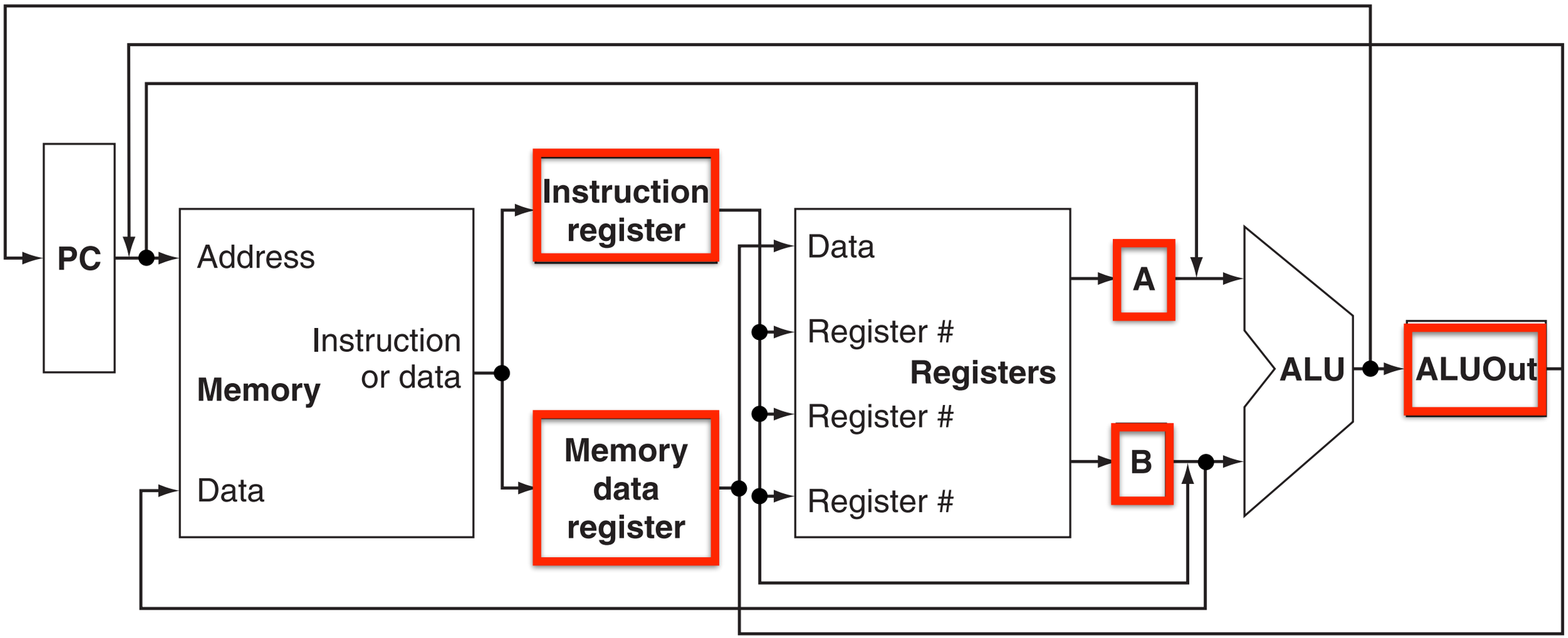
多周期CPU

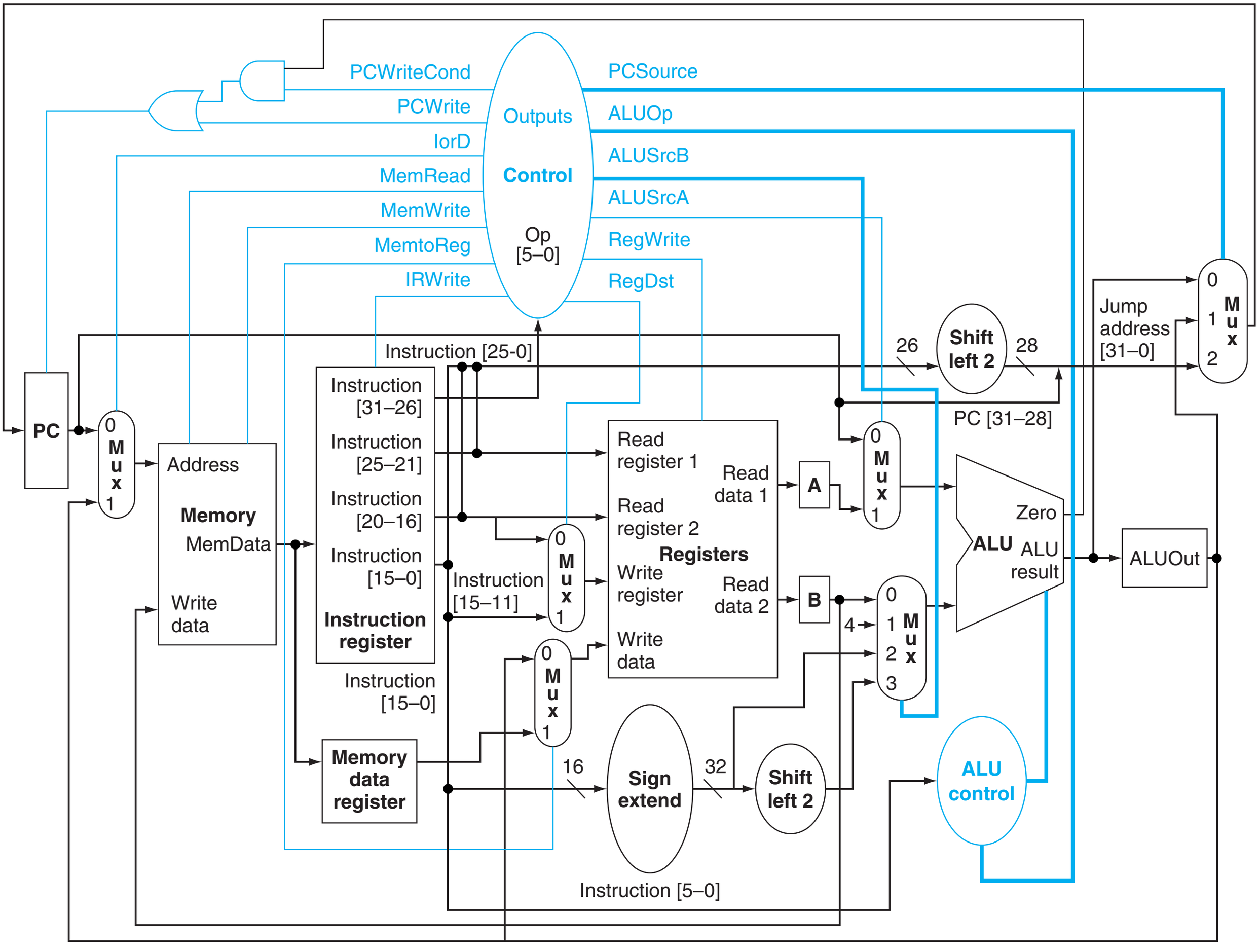
- 将指令执行过程分解成多个步骤
  - 和单周期CPU基本相同
- 每个步骤占用一个时钟周期
  - 尽量平衡各步骤间的延迟
  - 尽量限制每个步骤使用单一的主要部件
  - 控制器仅提供当前步骤所需要的控制信号
- 各步骤间应该
  - 保存好下一步骤要用到的值
  - 引入“新”的内部寄存器
  - 转到下一步骤执行
    - 引入状态标记当前步骤
    - 有限自动机

多周期CPU的Datapath

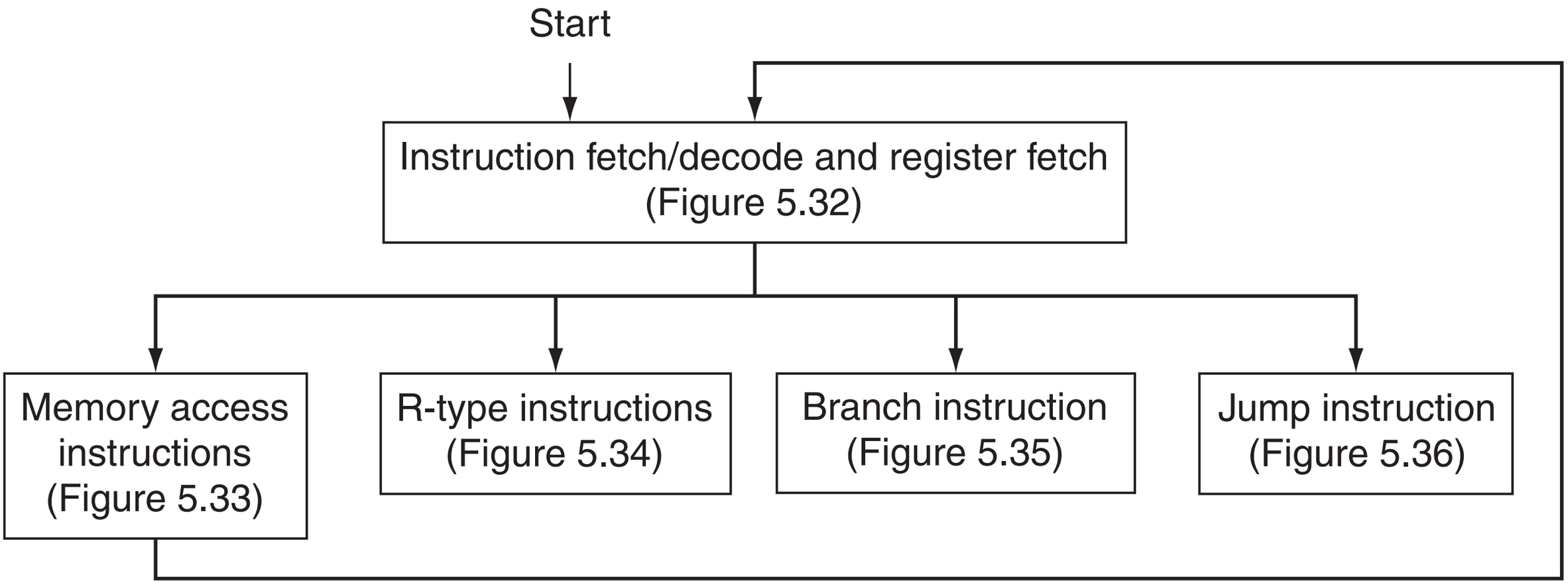


多周期CPU的Datapath

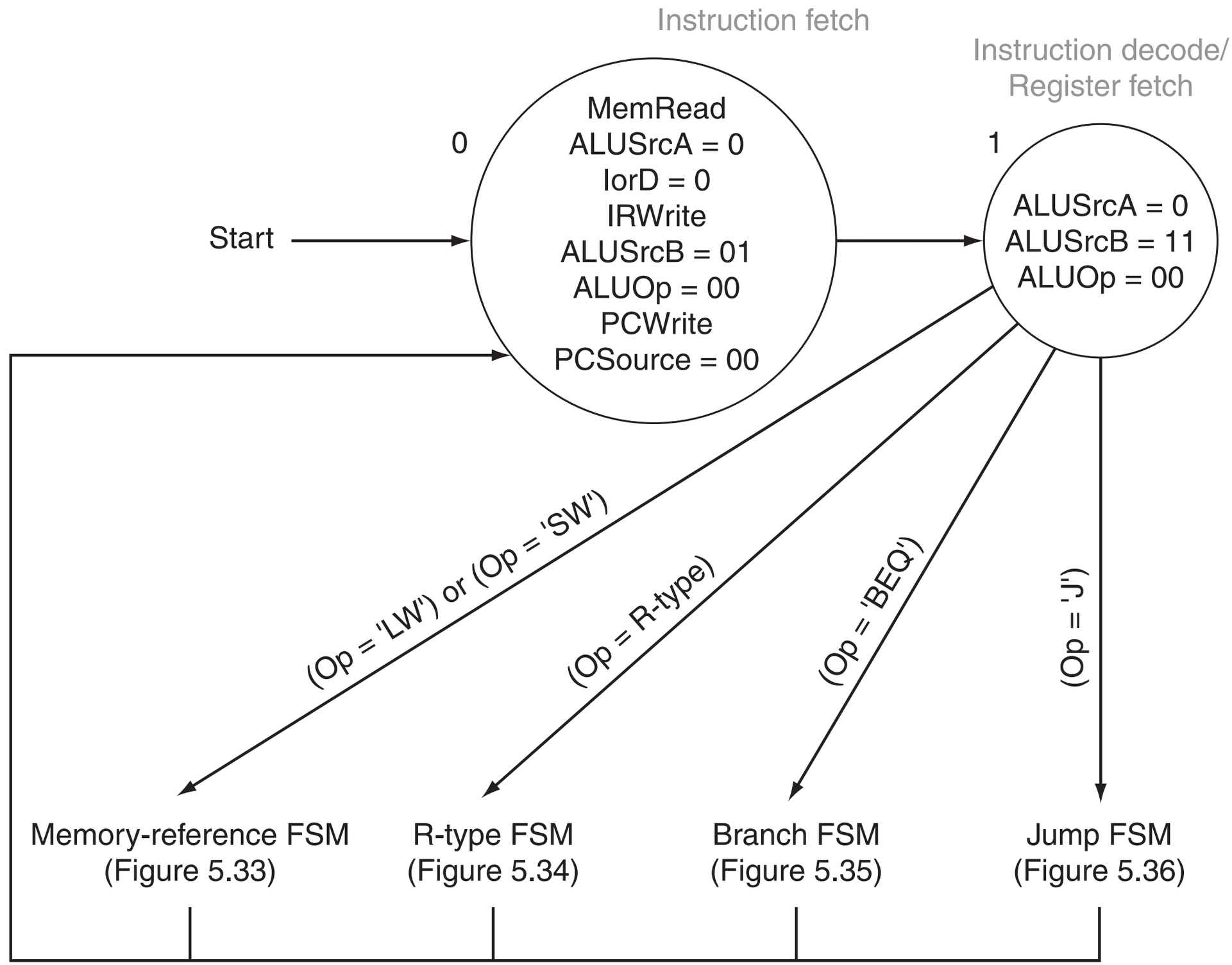


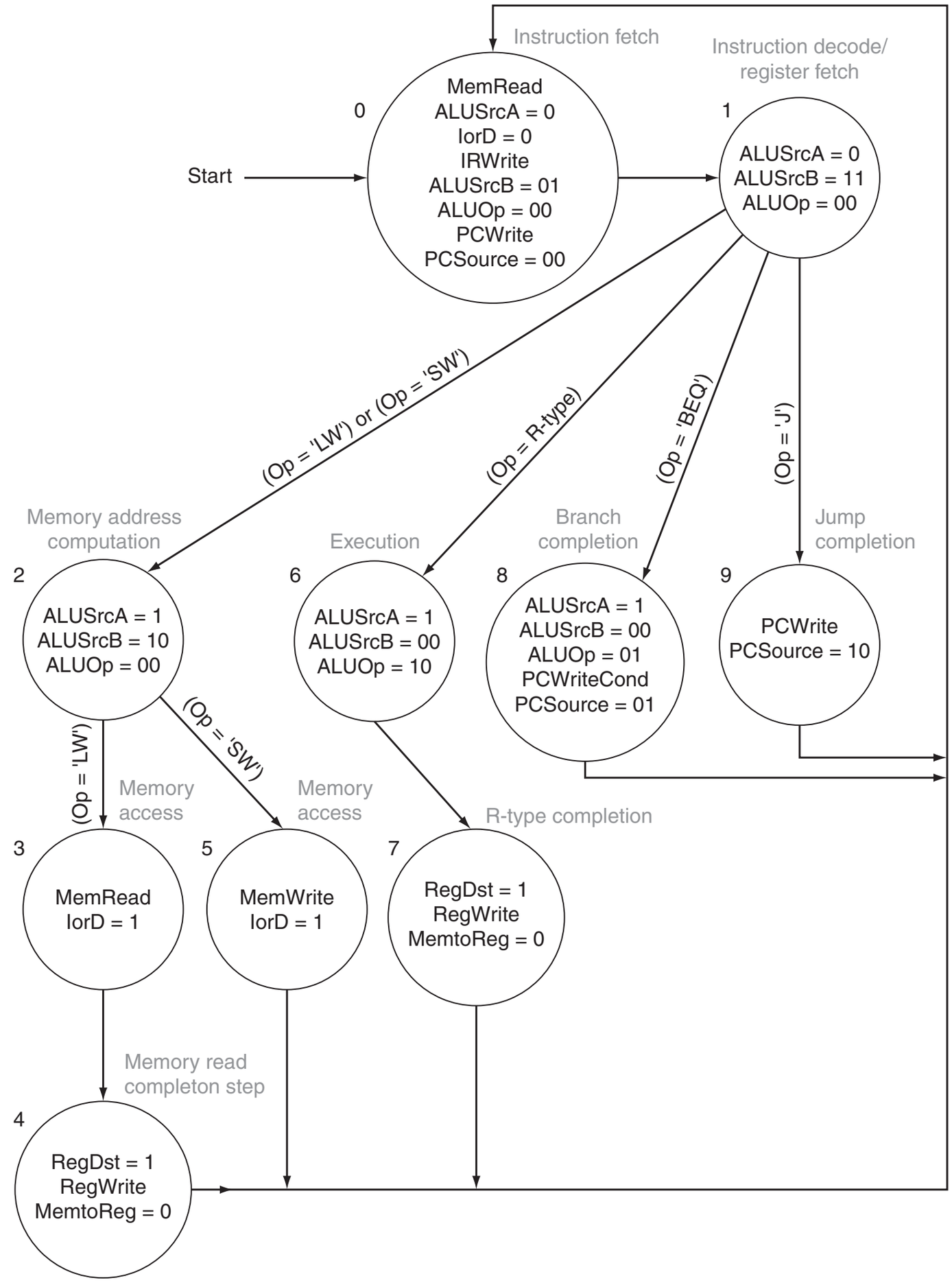

PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
Instruction [25-0]
26
Shift
left 2
28
Jump
address
[31–0]
0
1
2
Mux
PC
0
Mux
1
Address
Memory
MemData
Write
data
Instruction
[31–26]
Instruction
[25–21]
Instruction
[20–16]
Instruction
[15–0]
Instruction
register
Instruction
[15–0]
Instruction
[15–11]
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
Zero
ALU
ALU
result
ALUOut
Memory
data
register
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5–0]
PC [31–28]

有限自动机

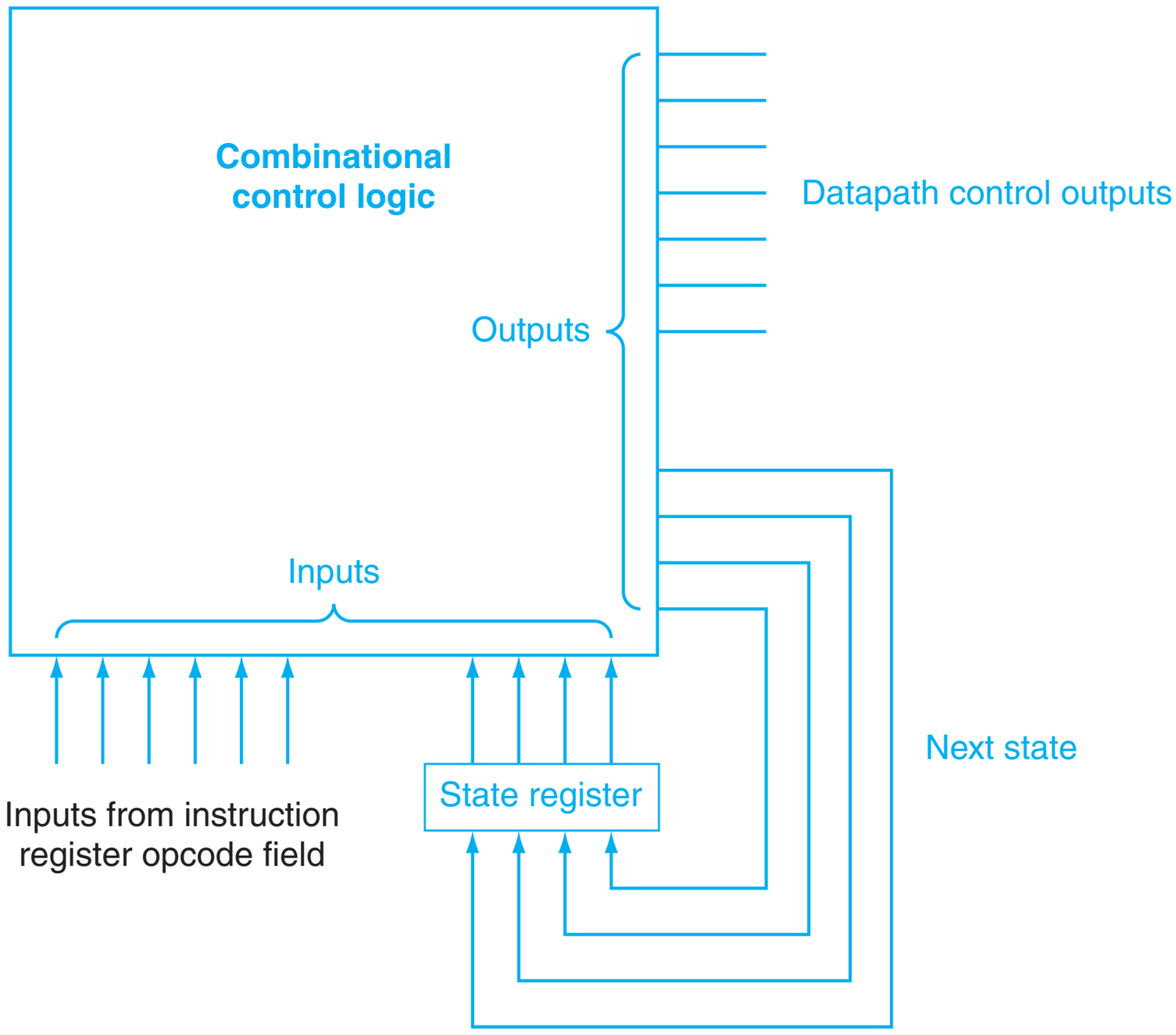

Start
Instruction fetch/decode and register fetch (Figure 5.32)
Memory access instructions (Figure 5.33)
R-type instructions (Figure 5.34)
Branch instruction (Figure 5.35)
Jump instruction (Figure 5.36)

取指与指令译码


Instruction fetch
Instruction decode/
Register fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
Start
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory-reference FSM
(Figure 5.33)
R-type FSM
(Figure 5.34)
Branch FSM
(Figure 5.35)
Jump FSM
(Figure 5.36)


Instruction fetch
Instruction decode/
register fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Start
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory address
computation
Execution
Branch
completion
Jump
completion
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory
access
Memory
access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
Memory read
completon step
4
RegDst = 1
RegWrite
MemtoReg = 0

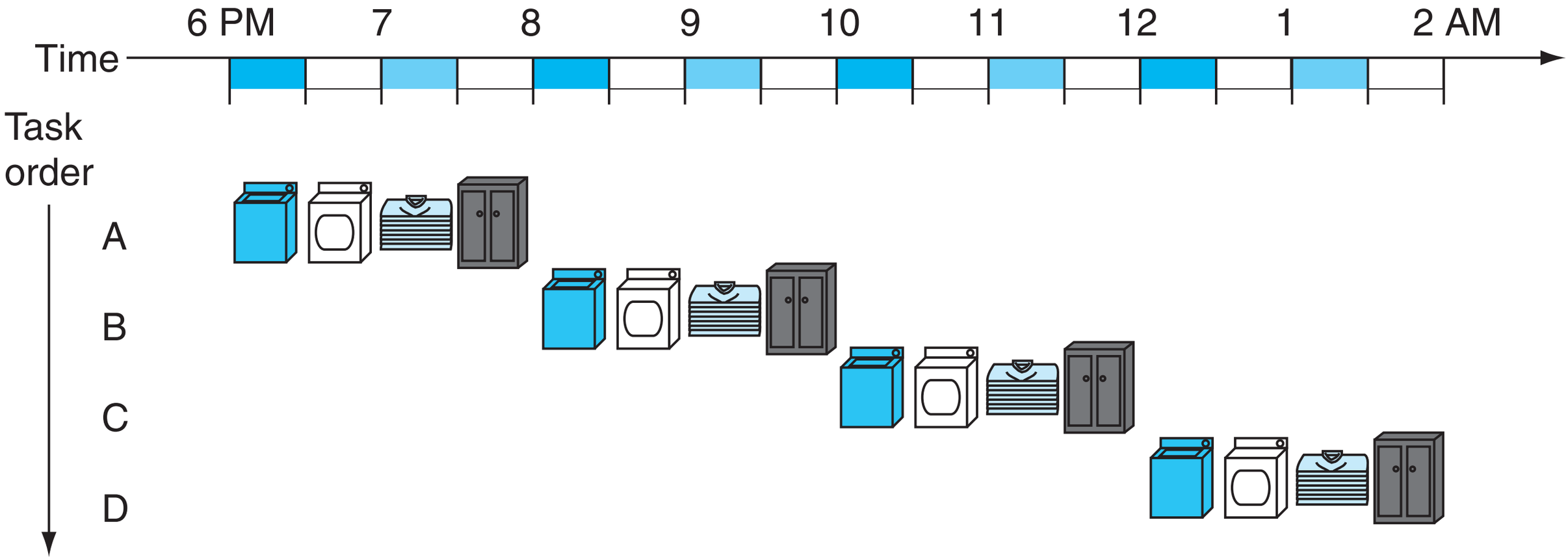
有限自动机


Combinational
control logic
Datapath control outputs
Outputs
Inputs
Next state
State register
Inputs from instruction
register opcode field

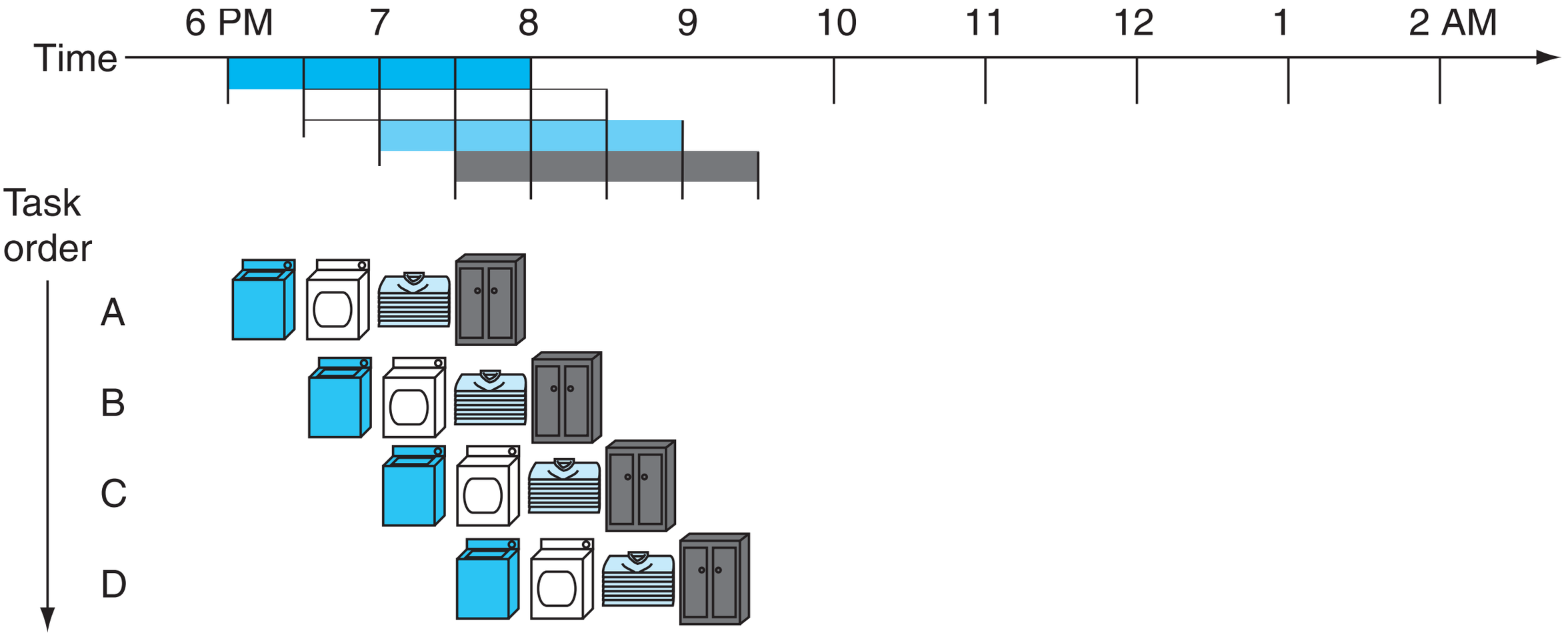
本讲提纲

- 单周期CPU和多周期CPU
  - 单周期CPU特点
  - 多周期CPU特点
- 流水线概述
  - 流水线概念
  - 流水线实现原理
  - 流水线性能指标
  - MIPS流水线的实现思路
- 流水线冒险
  - 结构冒险、数据冒险、控制冒险

流水线概念



流水线概念



流水线概念

- 计算机中的流水线是把一个重复的过程分解为若干个 子过程,每个子过程与其他子过程并行进行。由于这 种工作方式与工厂中的生产流水线十分相似,因此称为流水线技术。
- 提高处理机内部的并行性
  - 空间并行性,即在一个处理机内设置多个独立的操作部件,并且使这些部件并行工作
  - 时间并行性,就是采用流水线技术。流水线技术是一种非常经济、对提高计算机的运算速度非常有效的技术。采用流水线技术只需增加少量硬件就能把计算机的运算速度提高几倍,成为计算机中普遍使用的一种并行处理技术

流水线概念

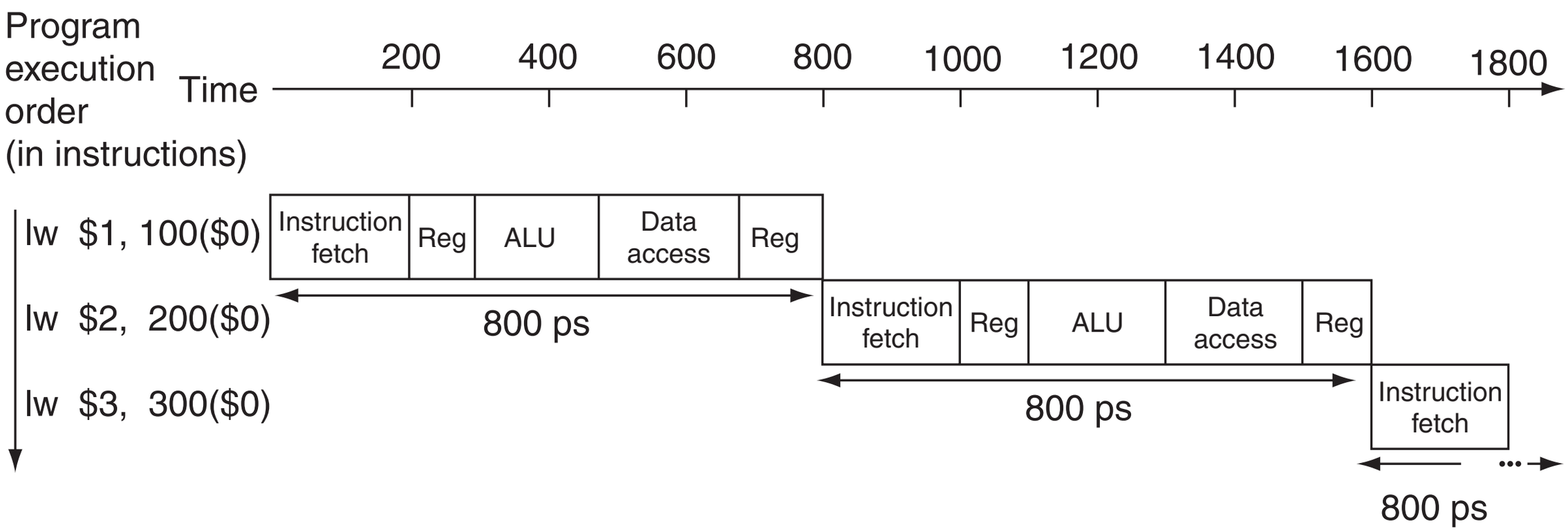
- 计算机各个部分几乎都可以采用流水线技术
  - 指令流水线:指令的执行过程采用流水线
  - 操作部件流水线:运算器中的操作部件,如浮点加法器、浮点乘法器等可以采用流水线
  - 宏流水线:多个计算机之间,通过存储器连接,可以采用流水线

执行指令的基本步骤

- 取指:从指令存储器中读指令(地址:PC)
- 读出一或两个源寄存器的值(寄存器组)
- 进行指令规定的运算(ALU)或计算地址
- 读/写数据存储器
- 将结果写入目的寄存器

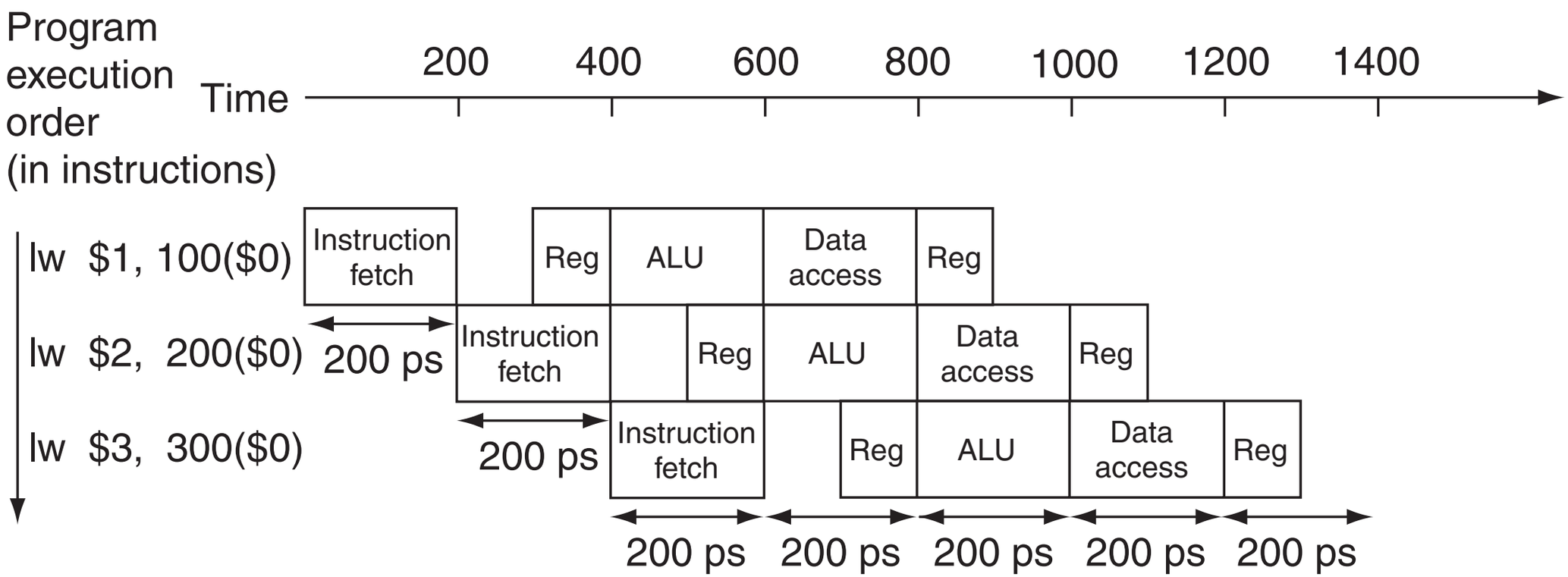
单周期非流水线指令执行过程

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (`lw`)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (`sw`)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (`add`, `sub`, `and`, `or`, `slt`)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (`beq`)	200 ps	100 ps	200 ps			500 ps



流水线的指令执行过程

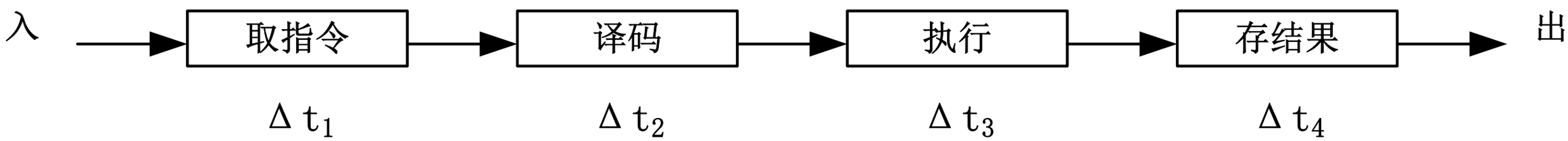
Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (`lw`)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (`sw`)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (`add`, `sub`, `and`, `or`, `slt`)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (`beq`)	200 ps	100 ps	200 ps			500 ps



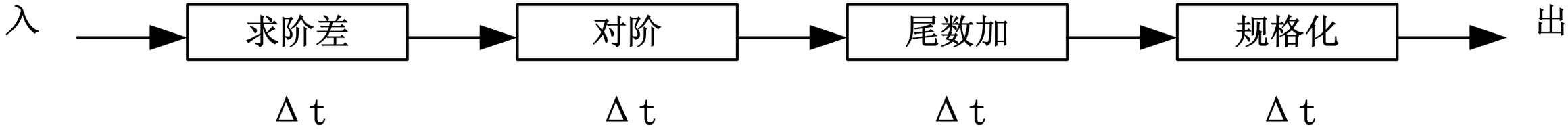
流水线的表示方法

- 流水线的每一个阶段完成一条指令的一部分，不同阶段并行完成不同指令的不同部分。
- 流水线中的每一个阶段称为一个流水阶段。一个流水阶段与另一个流水阶段相连接形成流水线。
- 指令从流水线的一端进入,经过流水线的处理，从另一端流出。目前大部分处理机的指令流水线在3-12段之间。
- 流水线常用的两种表示方法
  - 流水线连接图表示法,各个流水段顺序连接在一起
  - 流水线时空图表示法,直观描述流水线工作过程

流水线连接图表示法



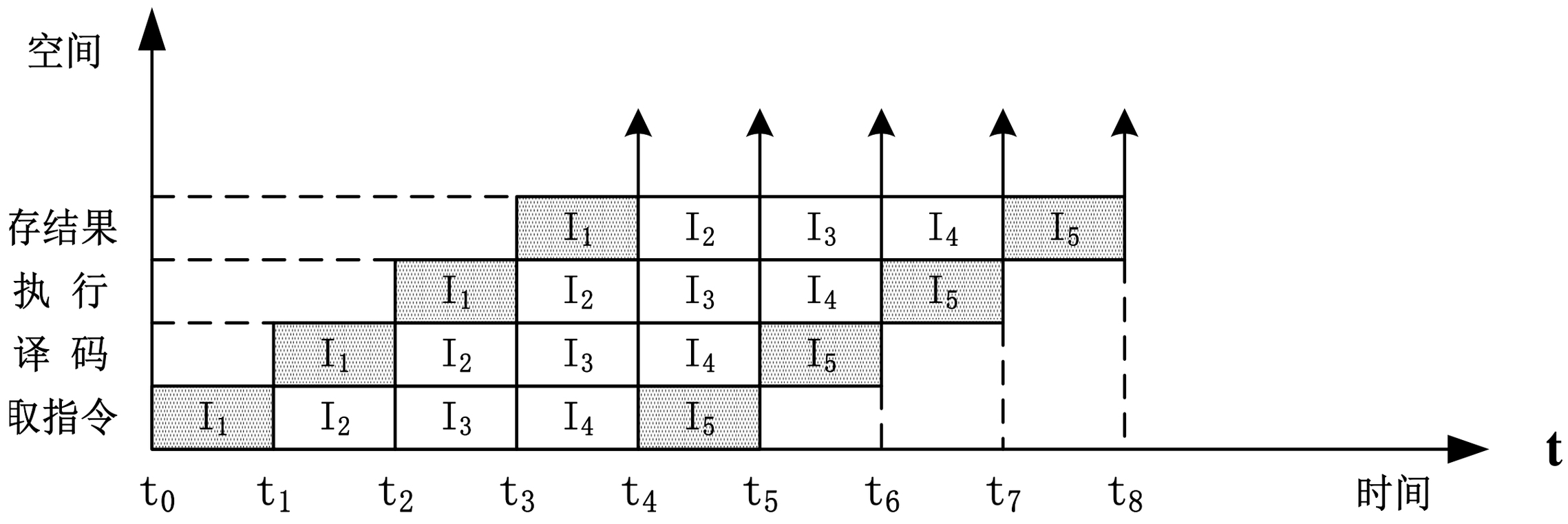
4段指令流水线



浮点加法器流水线

流水线的时空表示法

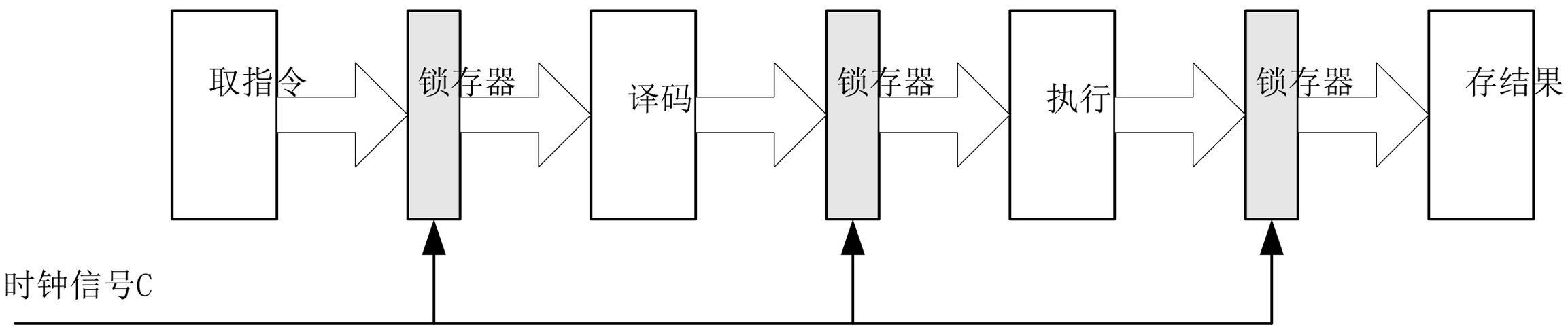
- 横坐标表示时间,也就是输入到流水线中的各个任务在流水线中所经过的时间。纵坐标表示空间,即流水线的每一个流水段。



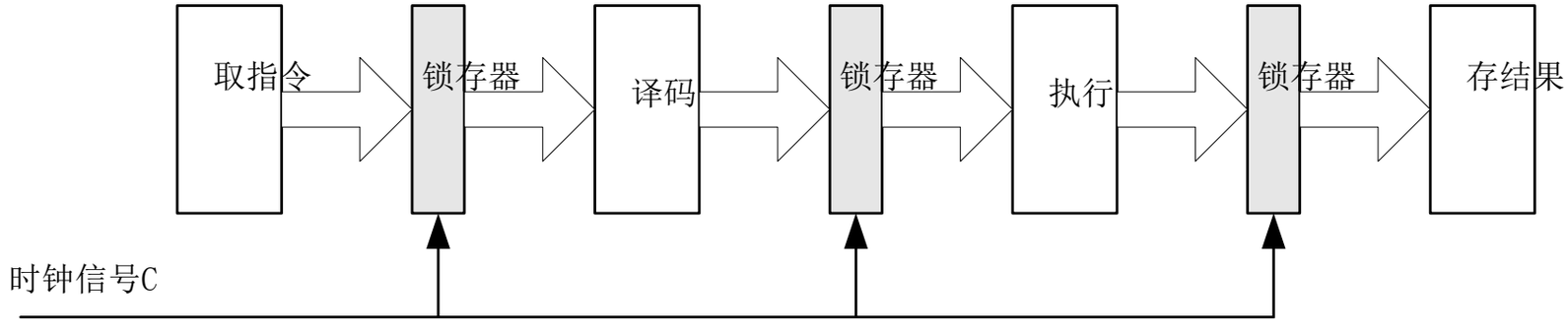
指令流水线时空图

流水线的特点

- 把一个任务(一条指令或一个操作)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来实现。
- 流水线每一个功能段部件后面都要有一个缓冲寄存器,或称为锁存器,其作用是保存本 流水段的结果。



流水线原理



- 在流水线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

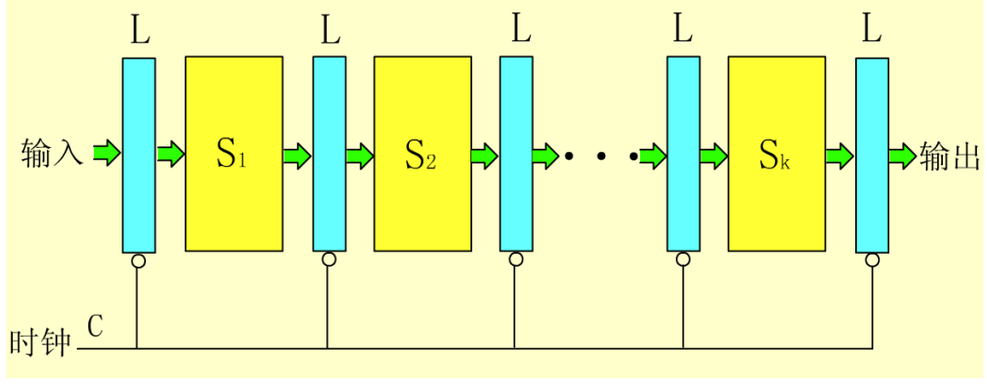
流水线原理

流水线的时钟周期

- 设过程段 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 $f = 1/\tau$



加速比分析

- 一个具有k级过程段的流水线处理n个任务需要的时钟周期数$T_k = k + (n-1)$
- 非流水线硬件处理所需要的始终周期数为： $T_L = nk$
- k级先行流水线的加速比$C_k=T_L/T_k=nk/(k+n-1)$
- 当n>>k时，C_k—>k （理论加速比）

MIPS指令的流水实现

- 指令执行步骤
  - 取指令(IF)
  - 指令译码(ID/RF)
  - 指令执行(EXE)
  - 读存储器(MEM)
  - 写回(WB)
- 各步骤占用的资源
  - IF:IM、PC
  - ID/RF:寄存器组、控制信号生成部件
  - EXE:ALU
  - MEM:DM
  - WB:寄存器组

流水线的实现原理

- 每一条指令的实现至多需要5个时钟周期。这5个时钟周期如下:
  - 取指令周期(IF)
  - 指令译码/读寄存器周期(ID)
  - 执行/有效地址计算周期(EX)
  - 存储器访问/分支完成周期(MEM)
  - 写回周期(WB)

  - 不同类型的指令在以上5个时钟周期中进行的操作 各不相同

IF: Instruction fetch

ID: Instruction decode/ register file read

EX: Execute/ address calculation

MEM: Memory access

WB: Write back

Add

4

0 M u x 1

PC

Address

Instruction

Instruction memory

Read register 1

Read register 2

Registers

Write register

Write data

Read data 1

Read data 2

Shift left 2

ADD

Add result

0 M u x 1

ALU

Zero

ALU result

Address

Read data

Data Memory

Write data

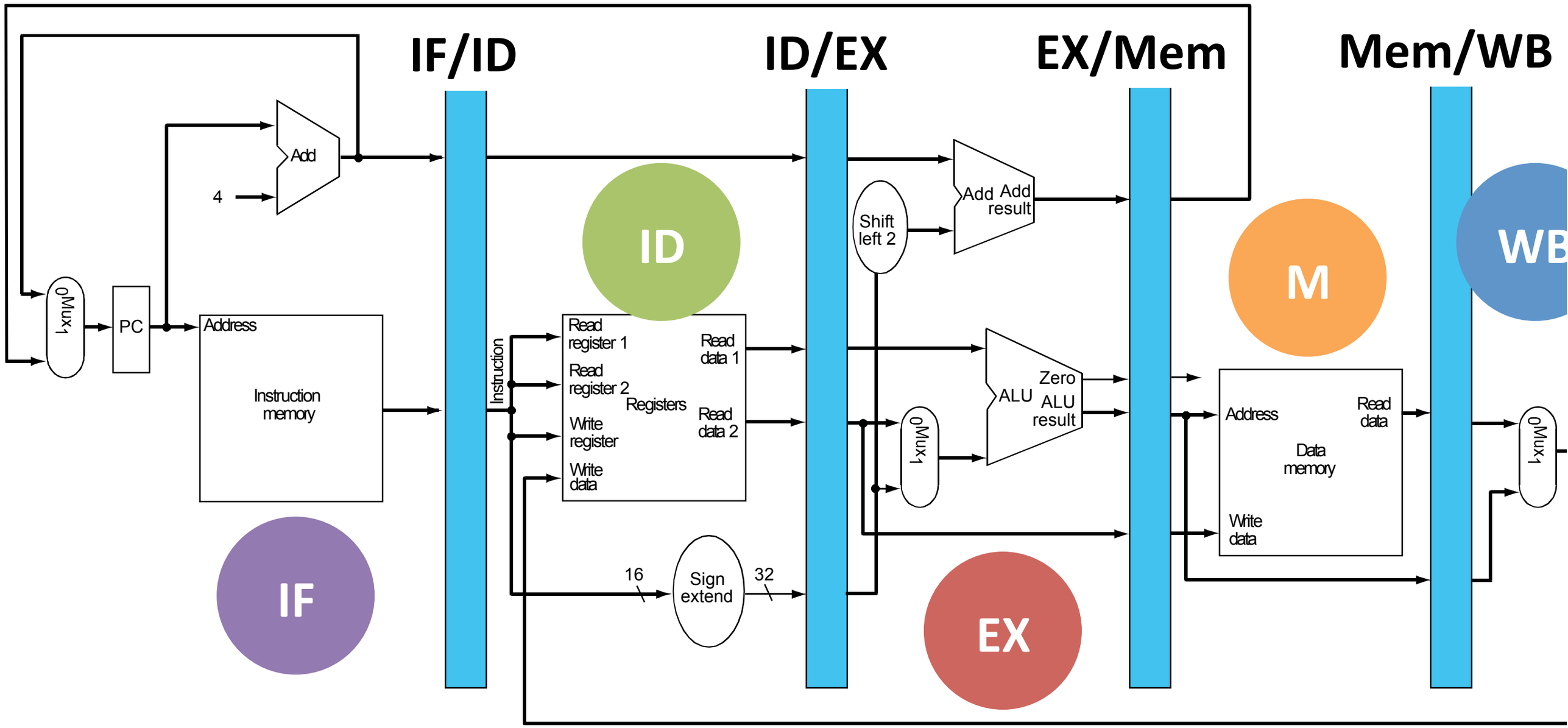
1 M u x 0

16

Sign extend

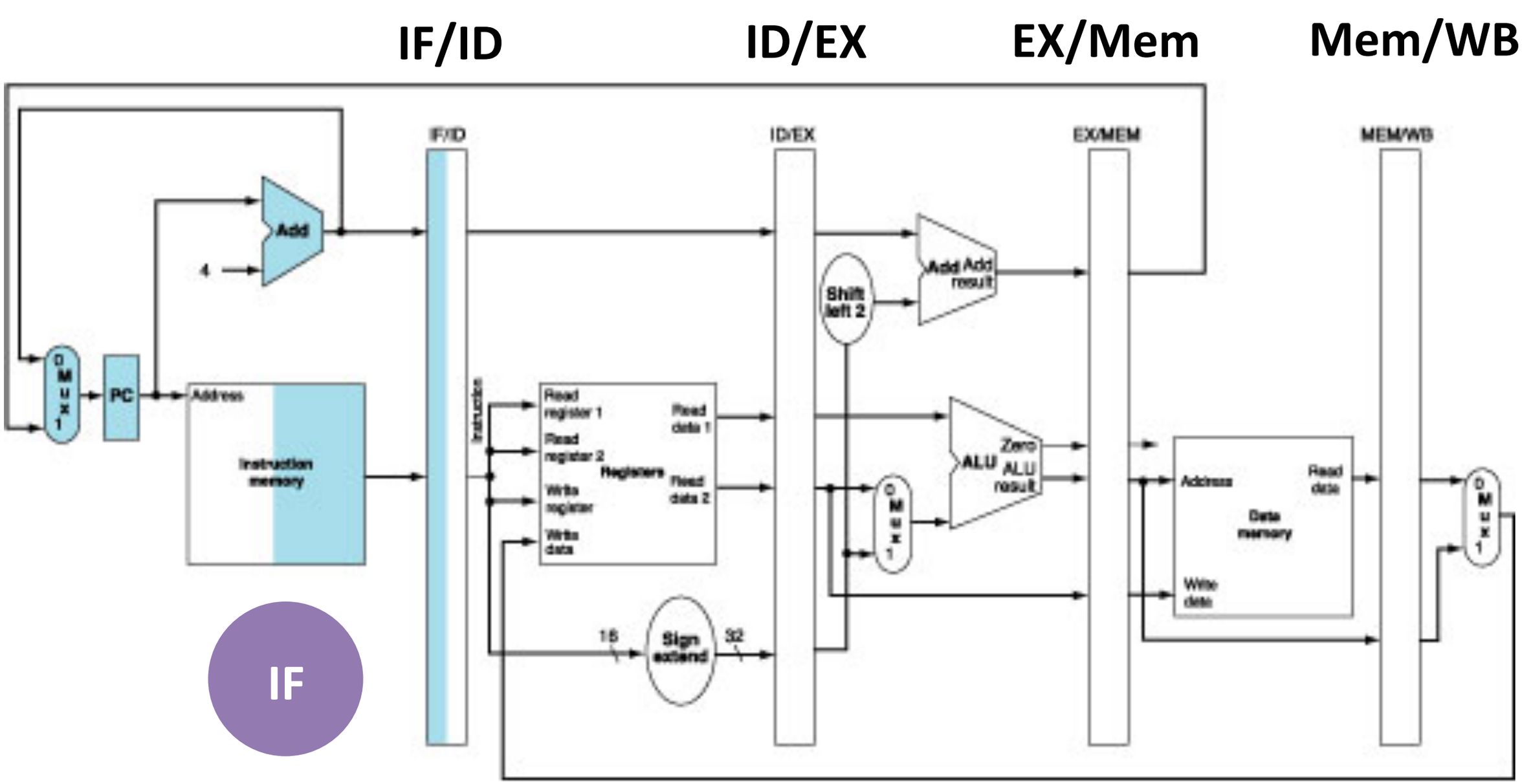
32

流水线寄存器


IF/ID
ID/EX
EX/Mem
Mem/WB
Add
4
M
Mux
0
1
PC
Address
Instruction memory
Instruction
ID
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign extend
32
IF
EX
WB

取指令

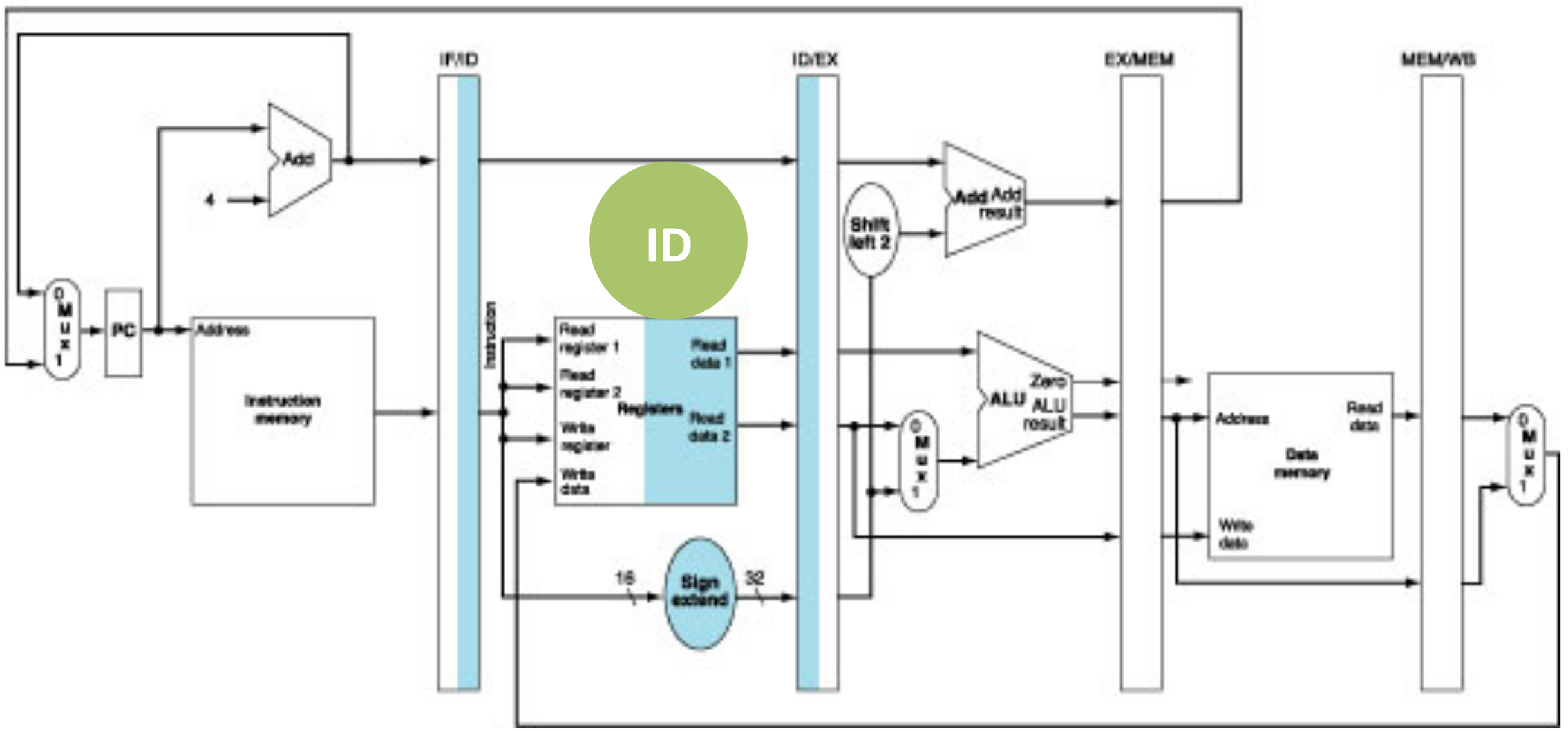
```
lw  $t1, 100($s0)
```



指令译码与读寄存器

```
lw  $t1, 100($s0)
```

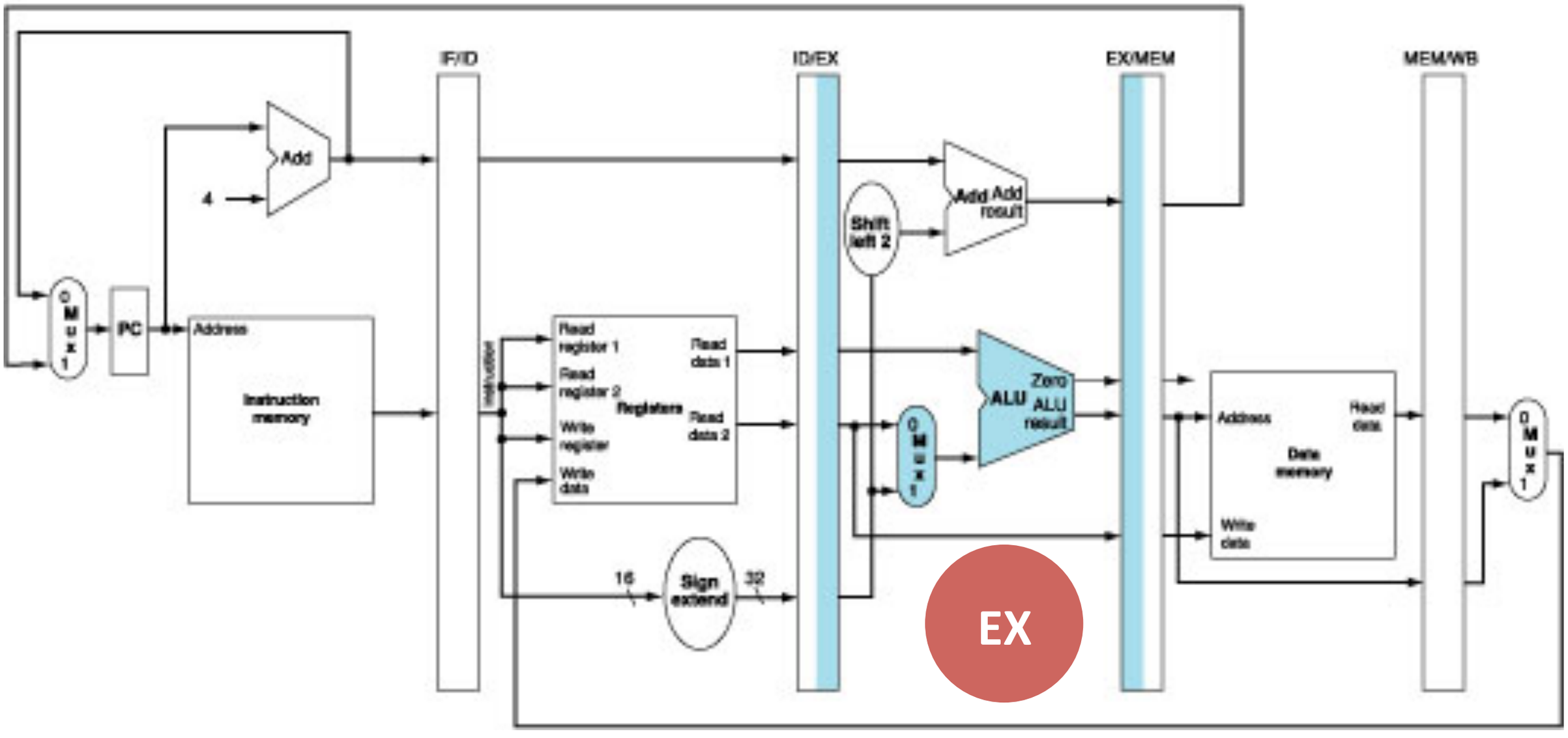
IF/ID　　ID/EX　　EX/Mem　　Mem/WB



执行指令

```
lw  $t1, 100($s0)
```

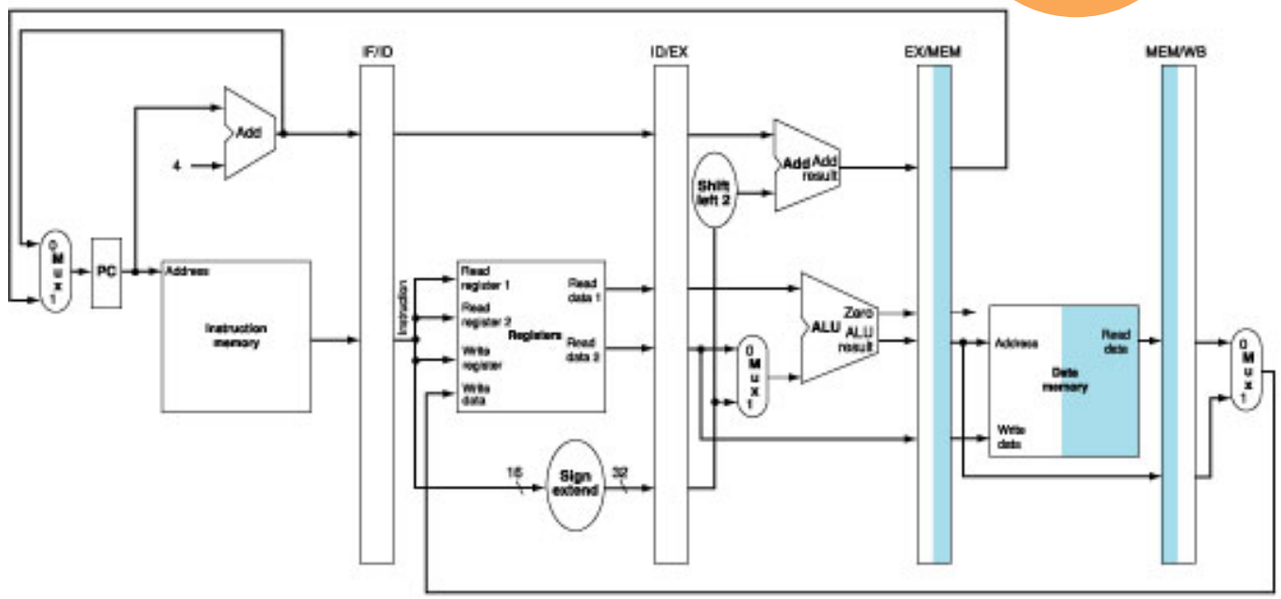
IF/ID ID/EX EX/Mem Mem/WB

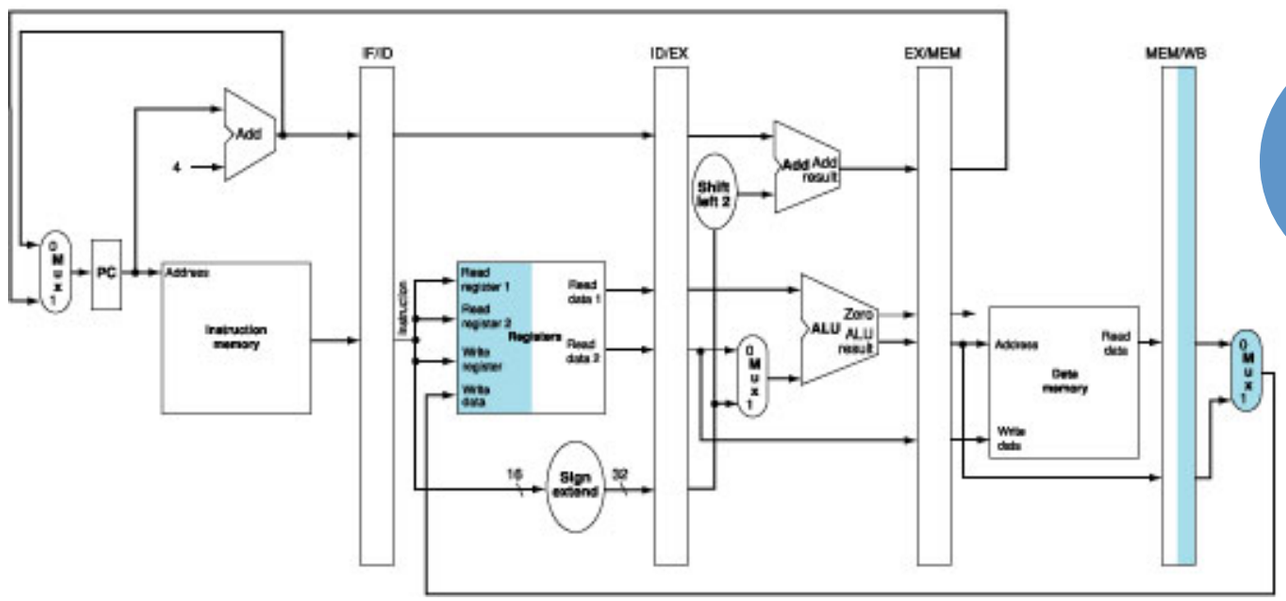


存储器访问、写回

```
lw  $t1, 100($s0)
```

M

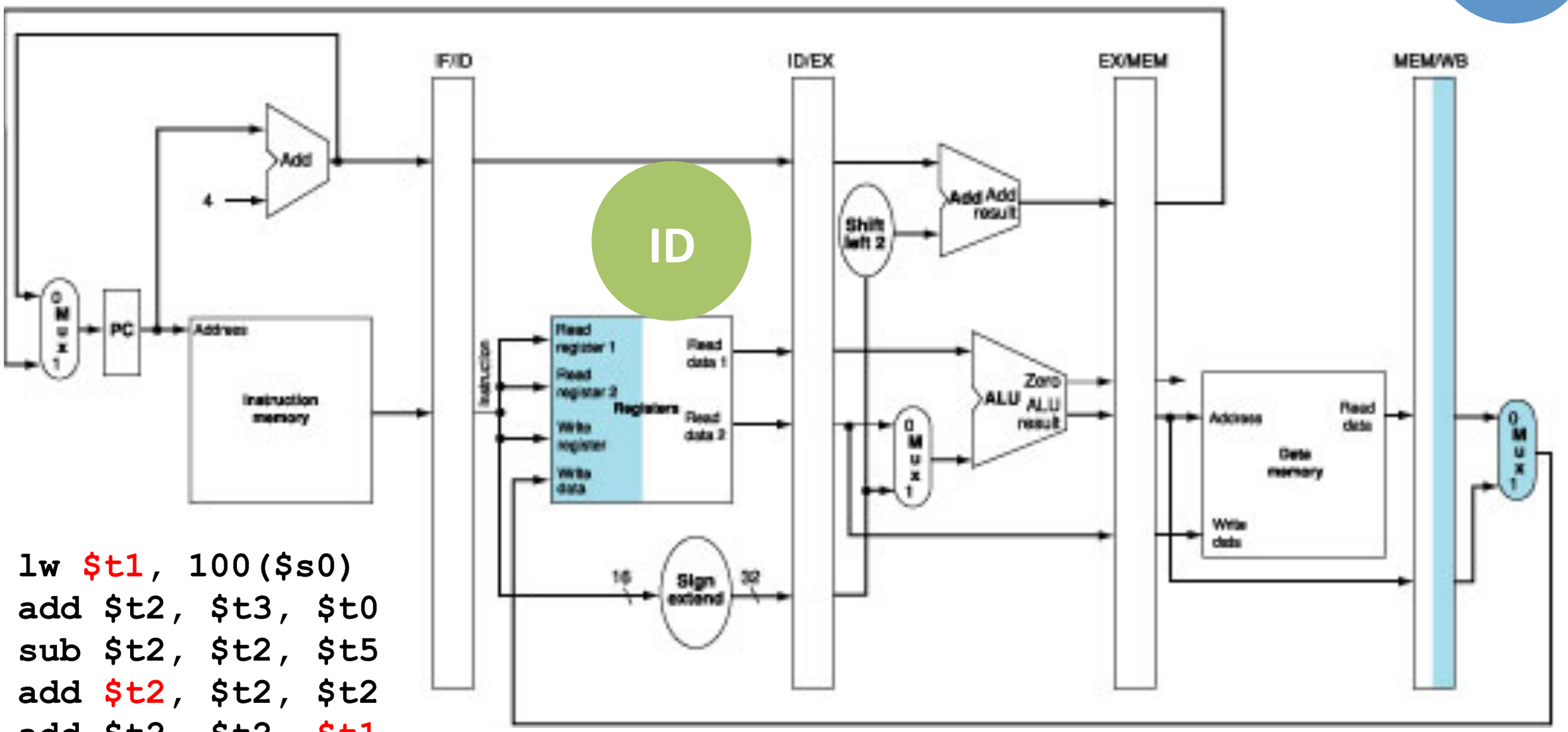




WB

写回时数据通路存在问题

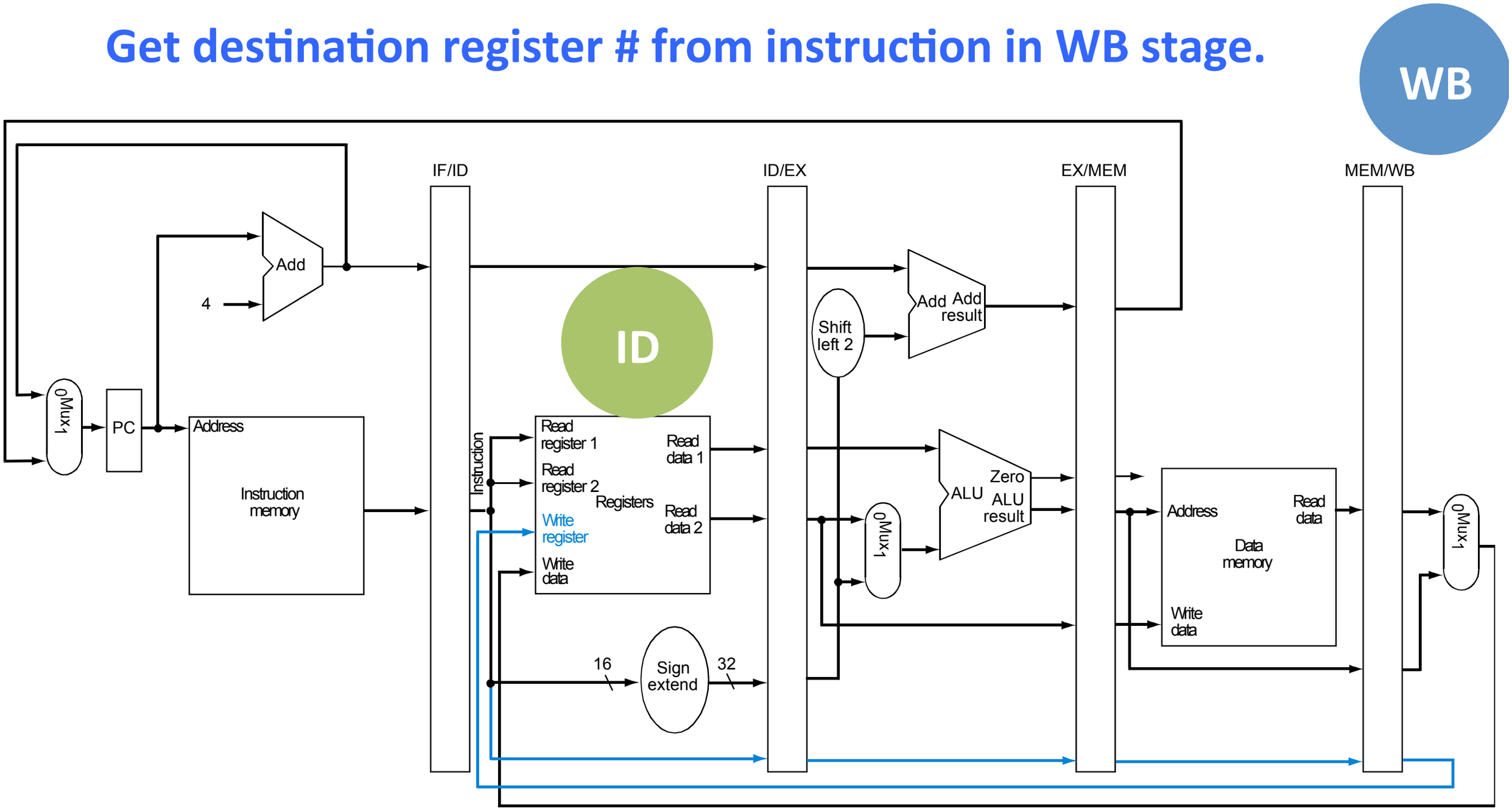
Which instruction's destination register # is used?




```
lw $t1, 100($s0)
add $t2, $t3, $t0
sub $t2, $t2, $t5
add $t2, $t2, $t2
add $t2, $t2, $t1
```

修正后的数据通路

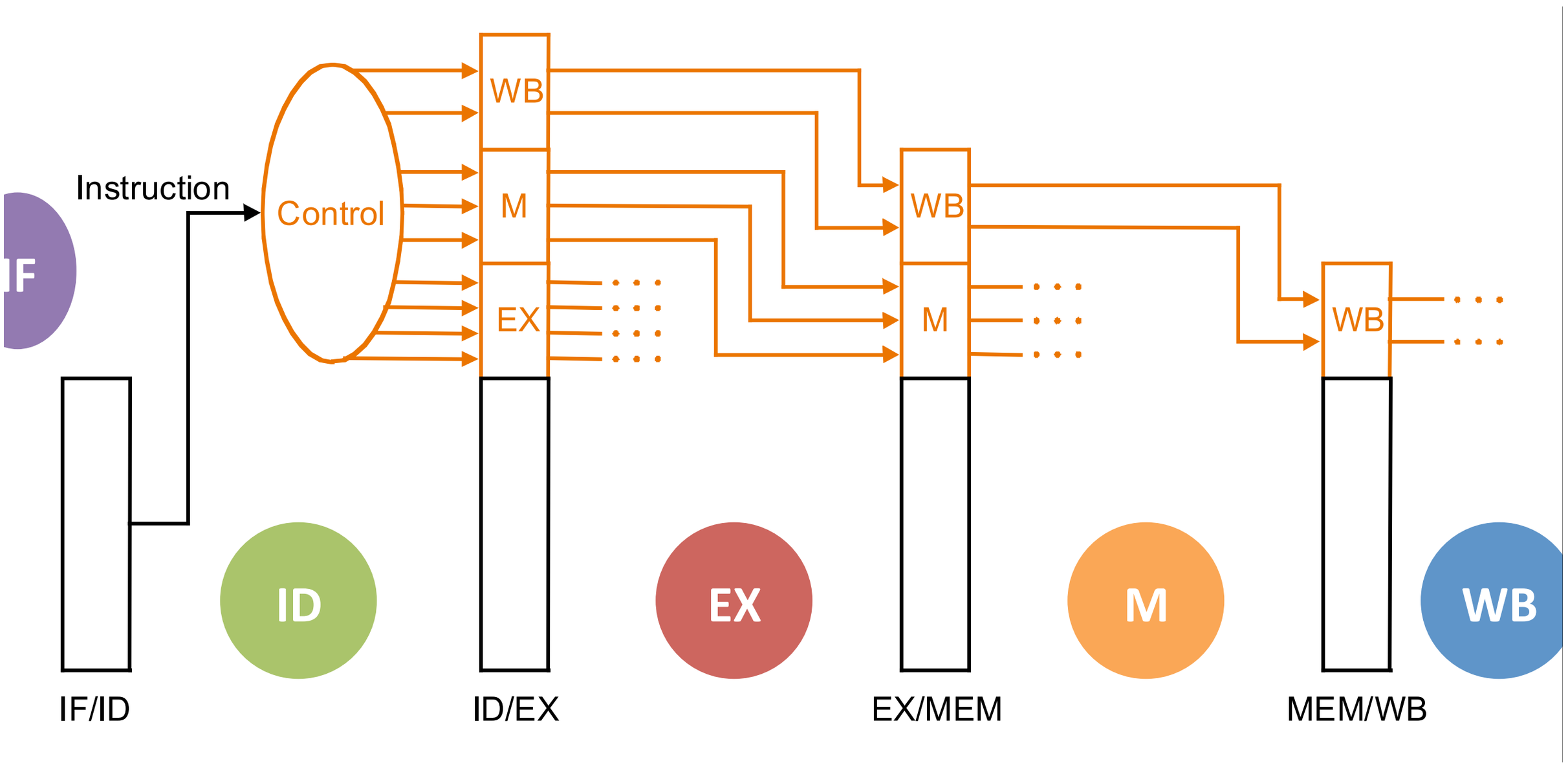
Get destination register # from instruction in WB stage.

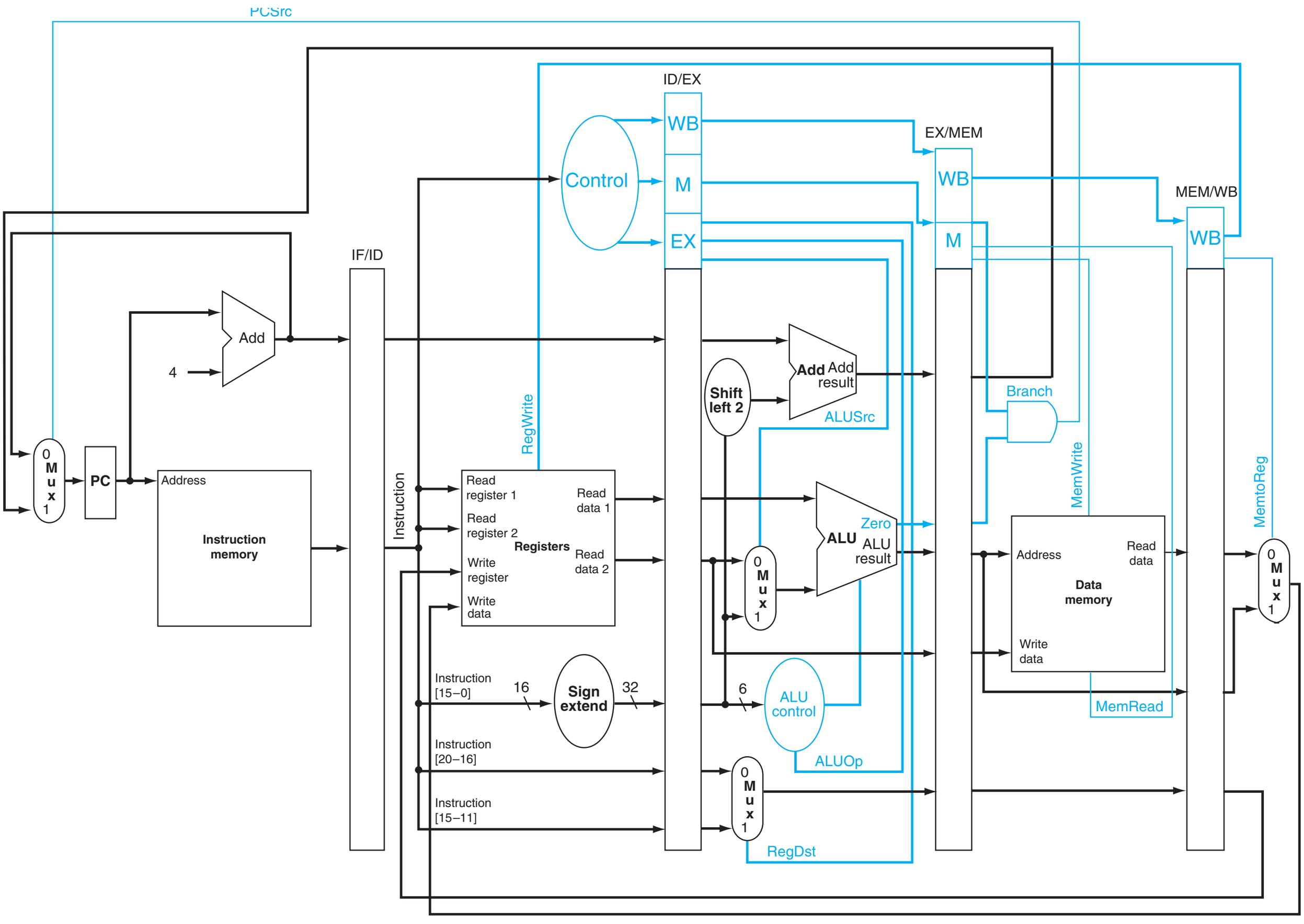


流水线控制

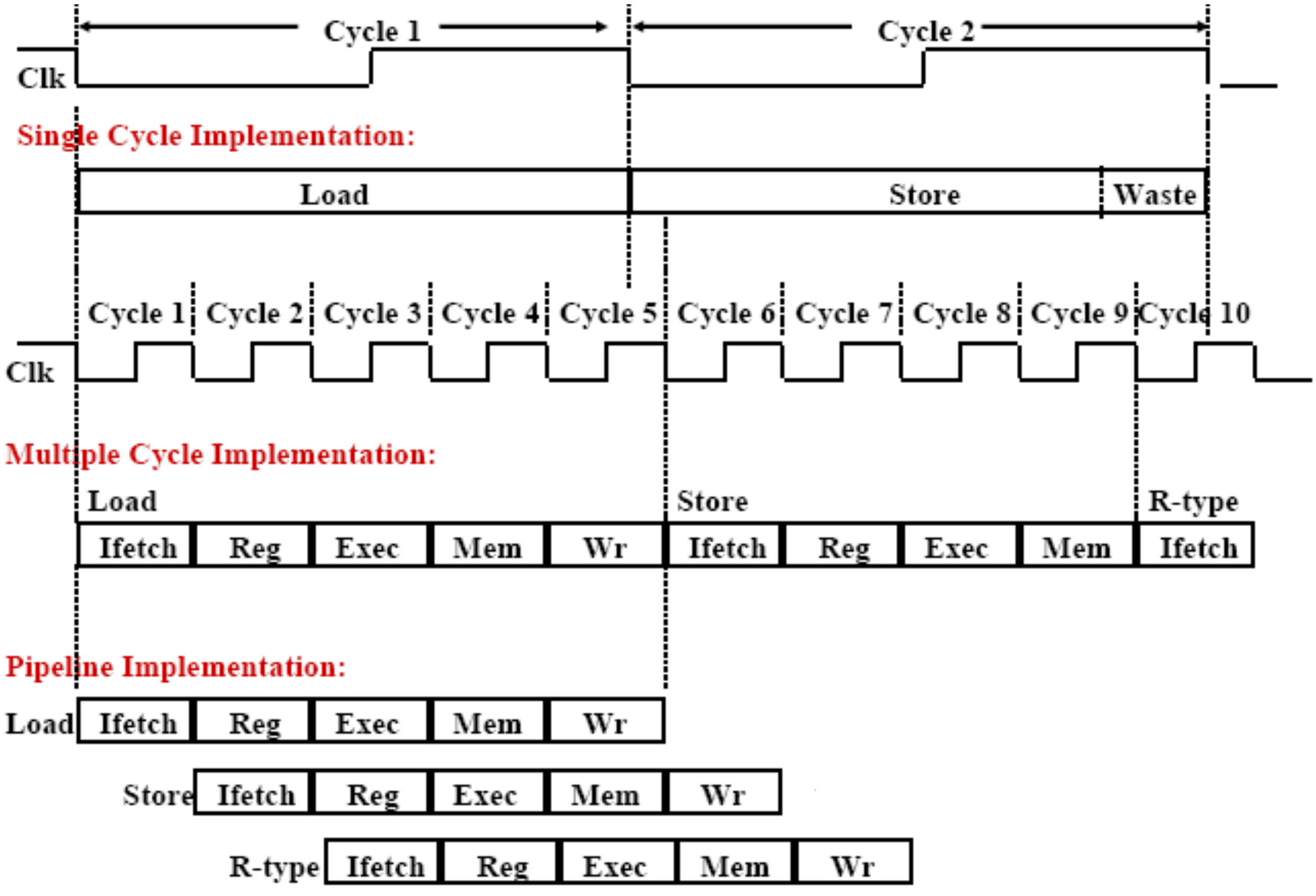
- 控制到每个功能部件
  - 但是,每个部件执行的不是同一条指令
- 解决方案
  - 把控制信号也和数据一样流动起来
  - 为区分起见,可以把控制信号前面加上标记,如 _IF等
- 每个时钟周期往下一步骤传递控制信号
  - 使正确的控制信号到达正确的位置

流水线控制


Instruction
Control
WB
M
EX
WB
M
WB
IF
ID
EX
M
WB
IF/ID
ID/EX
EX/MEM
MEM/WB


PCSrc
ID/EX
EX/MEM
MEM/WB
WB
M
EX
Control
IF/ID
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALUSrc
ALU
Zero
ALU result
Branch
MemWrite
MemtoReg
Address
Data memory
Read data
Write data
MemRead
Instruction [15–0]
16
Sign extend
32
6
ALU control
ALUOp
Instruction [20–16]
Instruction [15–11]
RegDst

不同类型CPU对比


Cycle 1
Cycle 2
Clk
Single Cycle Implementation:
Load
Store
Waste
Cycle 1
Cycle 2
Cycle 3
Cycle 4
Cycle 5
Cycle 6
Cycle 7
Cycle 8
Cycle 9
Cycle 10
Clk
Multiple Cycle Implementation:
Load
Store
R-type
Ifetch
Reg
Exec
Mem
Wr
Ifetch
Reg
Exec
Mem
Ifetch
Pipeline Implementation:
Load
Ifetch
Reg
Exec
Mem
Wr
Store
Ifetch
Reg
Exec
Mem
Wr
R-type
Ifetch
Reg
Exec
Mem
Wr

本讲提纲

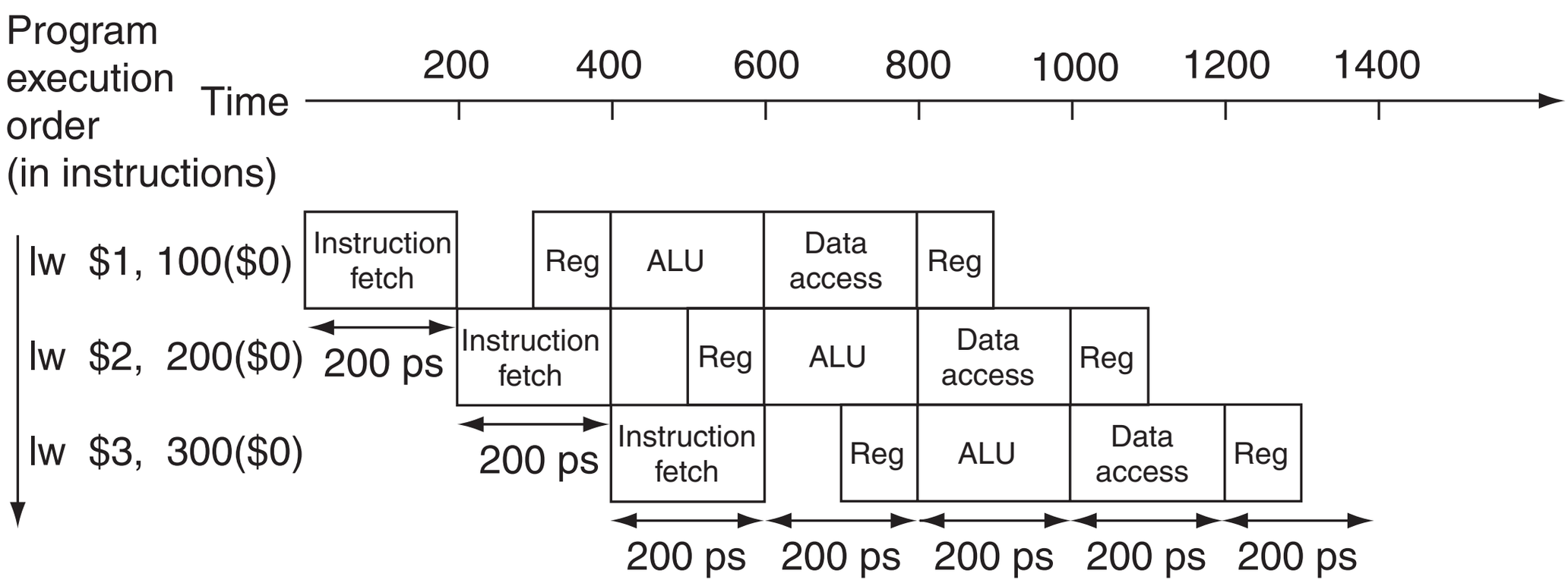
- 单周期CPU和多周期CPU
  - 单周期CPU特点
  - 多周期CPU特点
- 流水线概述
  - 流水线概念
  - 流水线实现原理
  - 流水线性能指标
  - MIPS流水线的实现思路
- 流水线冒险
  - 结构冒险、数据冒险、控制冒险

流水线冒险

- 结构冒险
  - 是指令在重叠执行的过程中,硬件资源满足不了指令重叠执行的要求而产生的冒险
- 数据冒险
  - 是指在同时重叠执行的几条指令中,一条指令依赖于前面指令执行结果数据,但是又得不到时发生的冒险。
- 控制冒险
  - 决策依赖于一条指令的结果，而其正在执行中

流水线冒险

- 结构冒险
  - 是指令在重叠执行的过程中,硬件资源满足不了指令重叠执行的要求而产生的冒险



如果只有一个寄存器，执行第四条指令时，第一条指令和第四条指令将同时访问寄存器，就会发生结构冒险

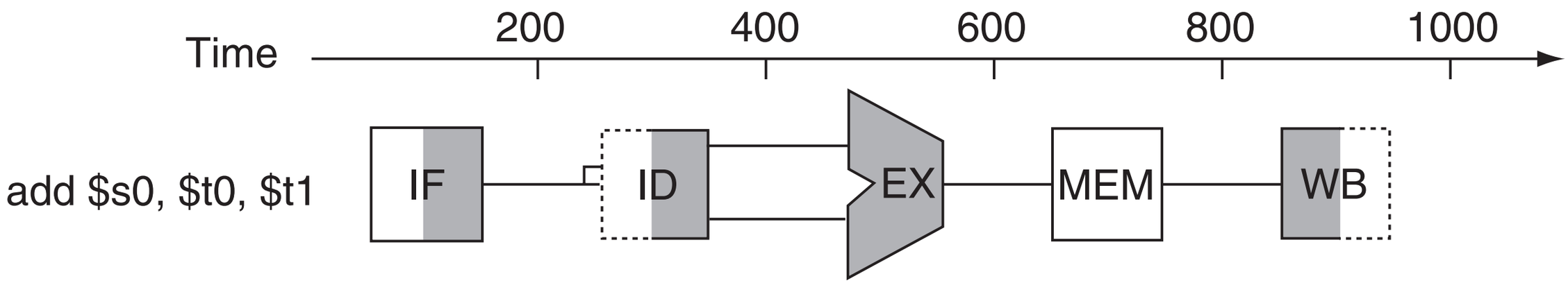
流水线冒险

- 结构冒险
  - 是指令在重叠执行的过程中,硬件资源满足不了指令重叠执行的要求而产生的冒险
  - 解决方案
    - 数据通路中的每一个功能单元都只能在一个流水级中使用

流水线冒险

- 数据冒险
  - 是指在同时重叠执行的几条指令中,一条指令依赖于前面指令执行结果数据,但是又得不到时发生的冒险。

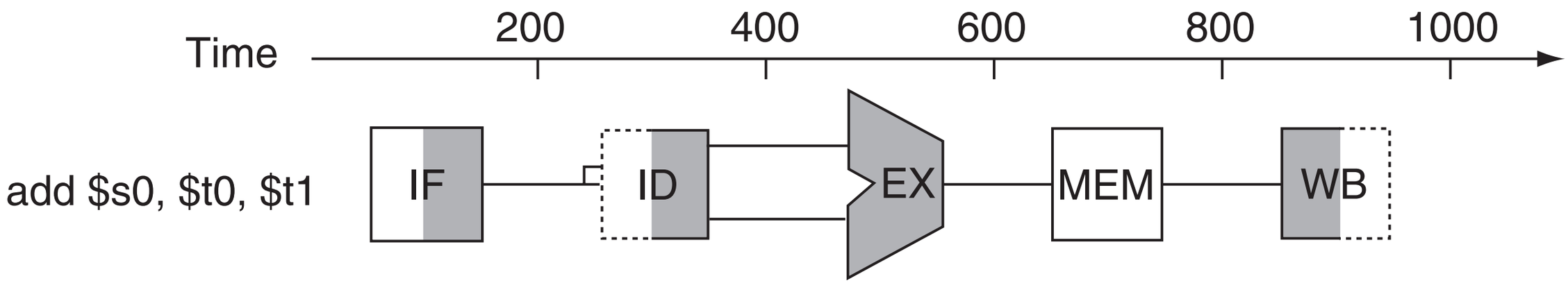
```
add     $s0, $t0, $t1
sub     $t2, $s0, $t3
```



流水线冒险

- 数据冒险
  - 是指在同时重叠执行的几条指令中,一条指令依赖于前面指令执行结果数据,但是又得不到时发生的冒险。

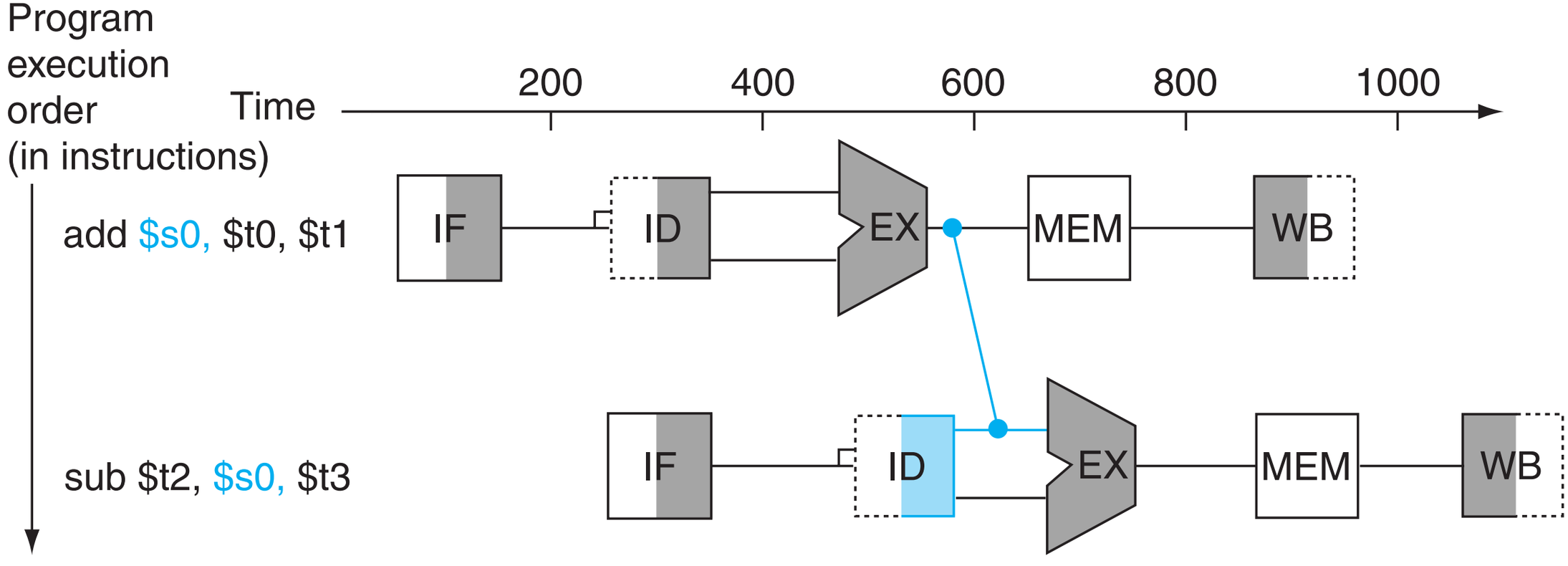
```
add     $s0, $t0, $t1
sub     $t2, $s0, $t3
```



加法指令执行到第五部才能返回结果

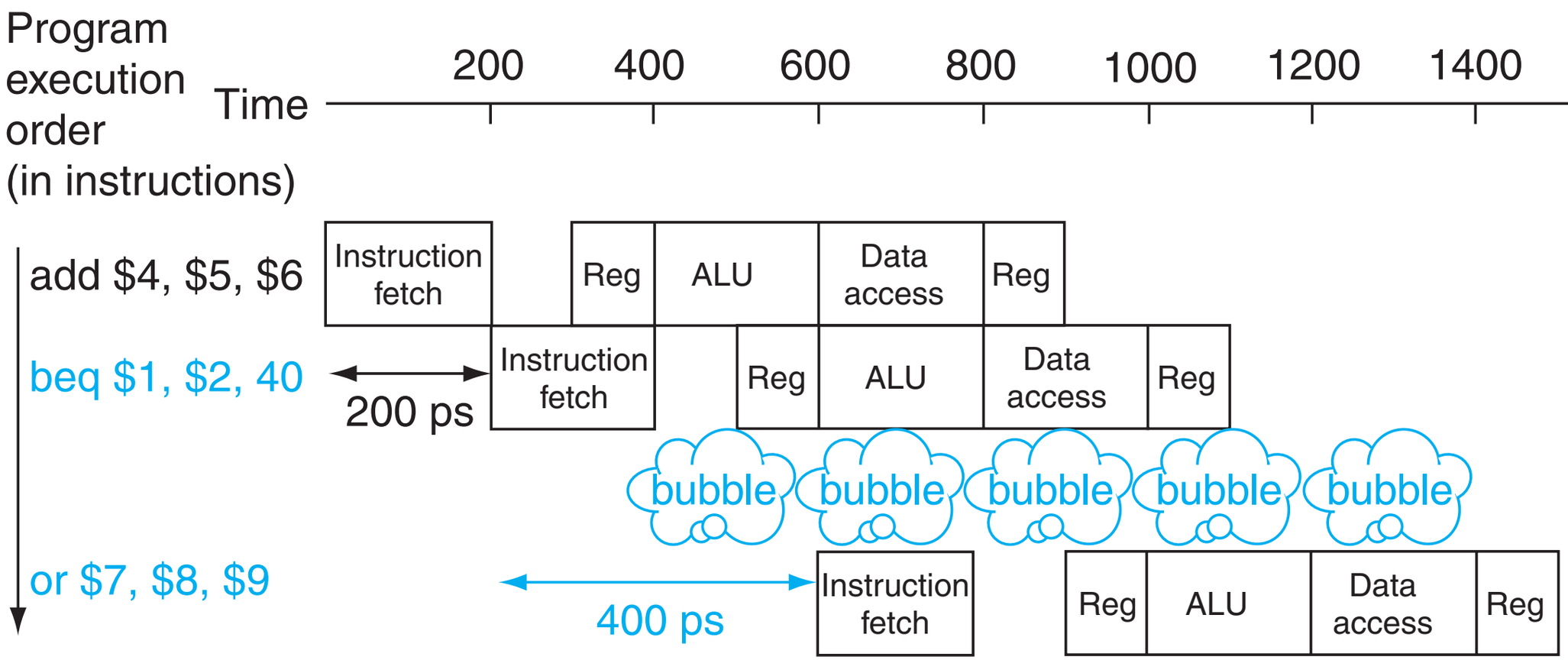
流水线冒险

- 数据冒险
  - 是指在同时重叠执行的几条指令中,一条指令依赖于前面指令执行结果数据,但是又得不到时发生的冒险。
  - 解决方案：转发



流水线冒险

- 控制冒险
  - 决策依赖于一条指令的结果，而其正在执行中
  - 解决方案：阻塞



本讲小结

- 单周期CPU和多周期CPU
- 流水线概述
  - 流水线概念
  - 流水线实现原理
  - 流水线性能指标
  - MIPS流水线的实现思路
- 流水线冒险
  - 结构冒险、数据冒险、控制冒险