中国科学院大学
University of Chinese Academy of Sciences

B0911006Y-01 2024-2025学年春季学期

计算机组成原理

Principles of Computer Organization

多周期处理器 II

控制单元设计、性能优化

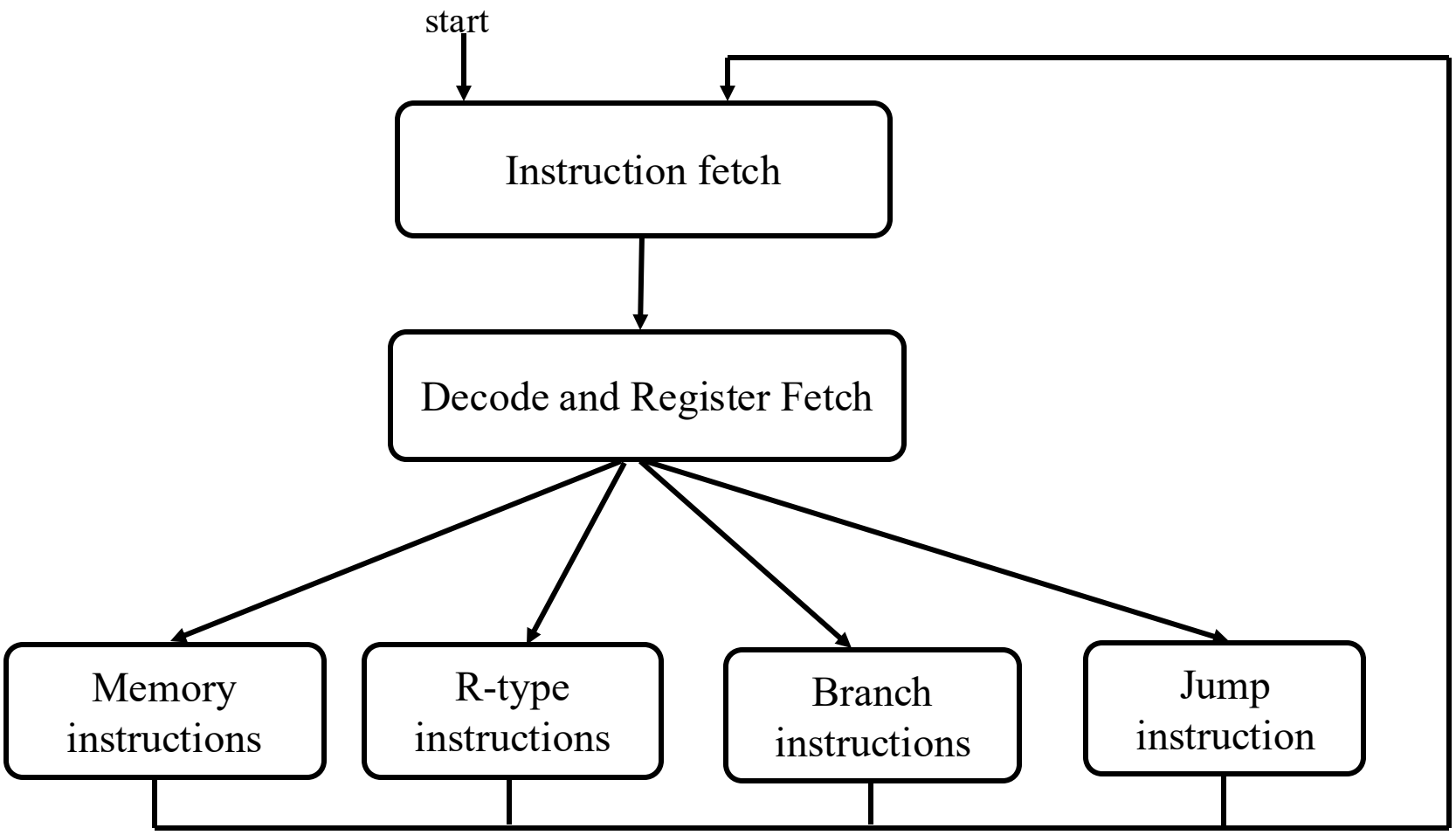
主讲教师：石 侃
shikan@ict.ac.cn

2025年5月20日

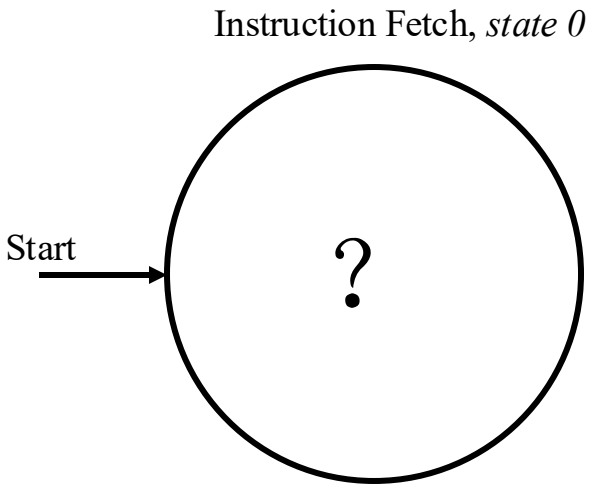
What About the Control?

- Single-cycle control used combinational logic
- How about Multi-cycle control?
  - FSM defines a succession of states, transitions between states (based on inputs), and outputs (based on state)
  - First two states are always idential for every instruction, next states depend on instruction opcode

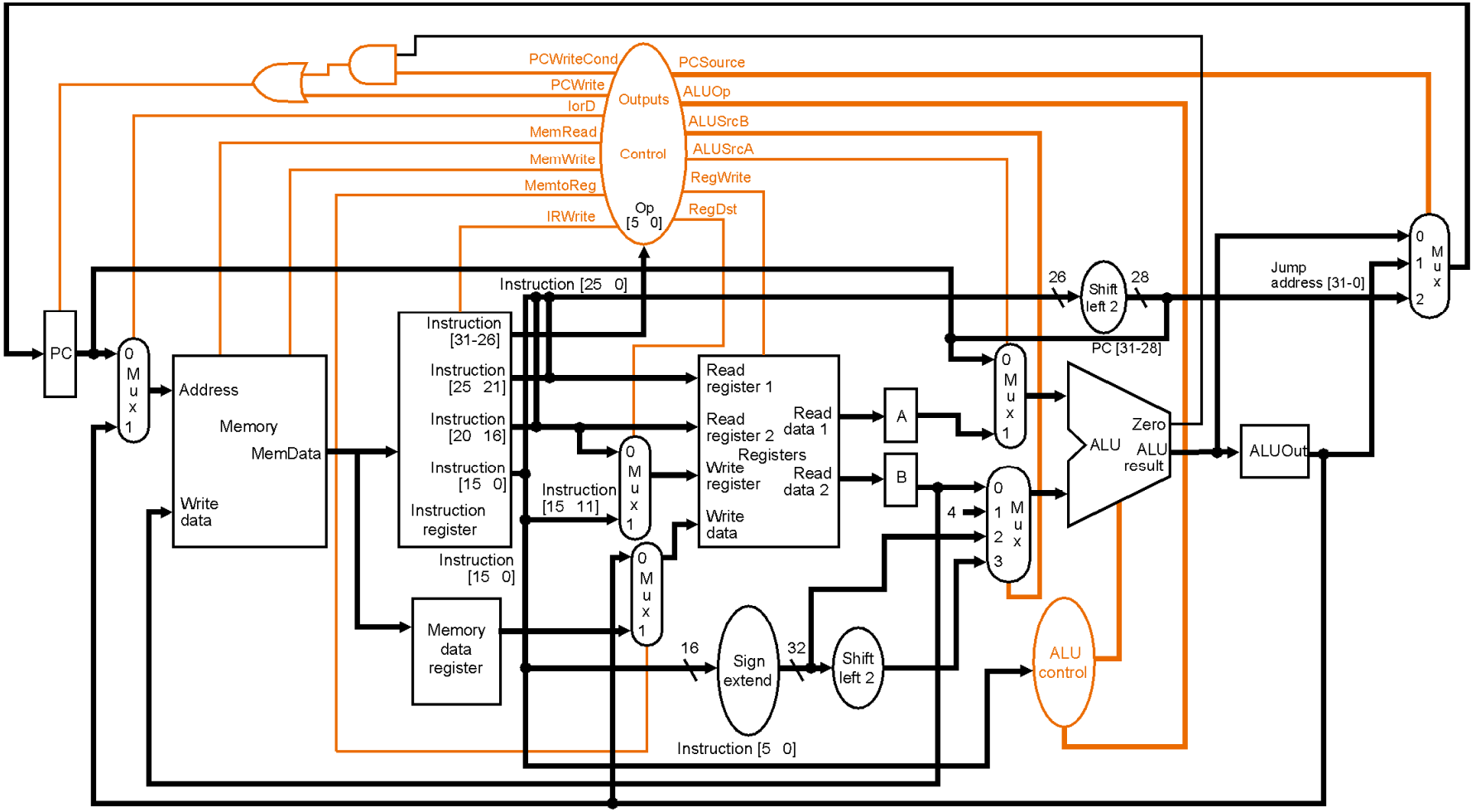
Multi-Cycle Control FSM



The first state of the FSM

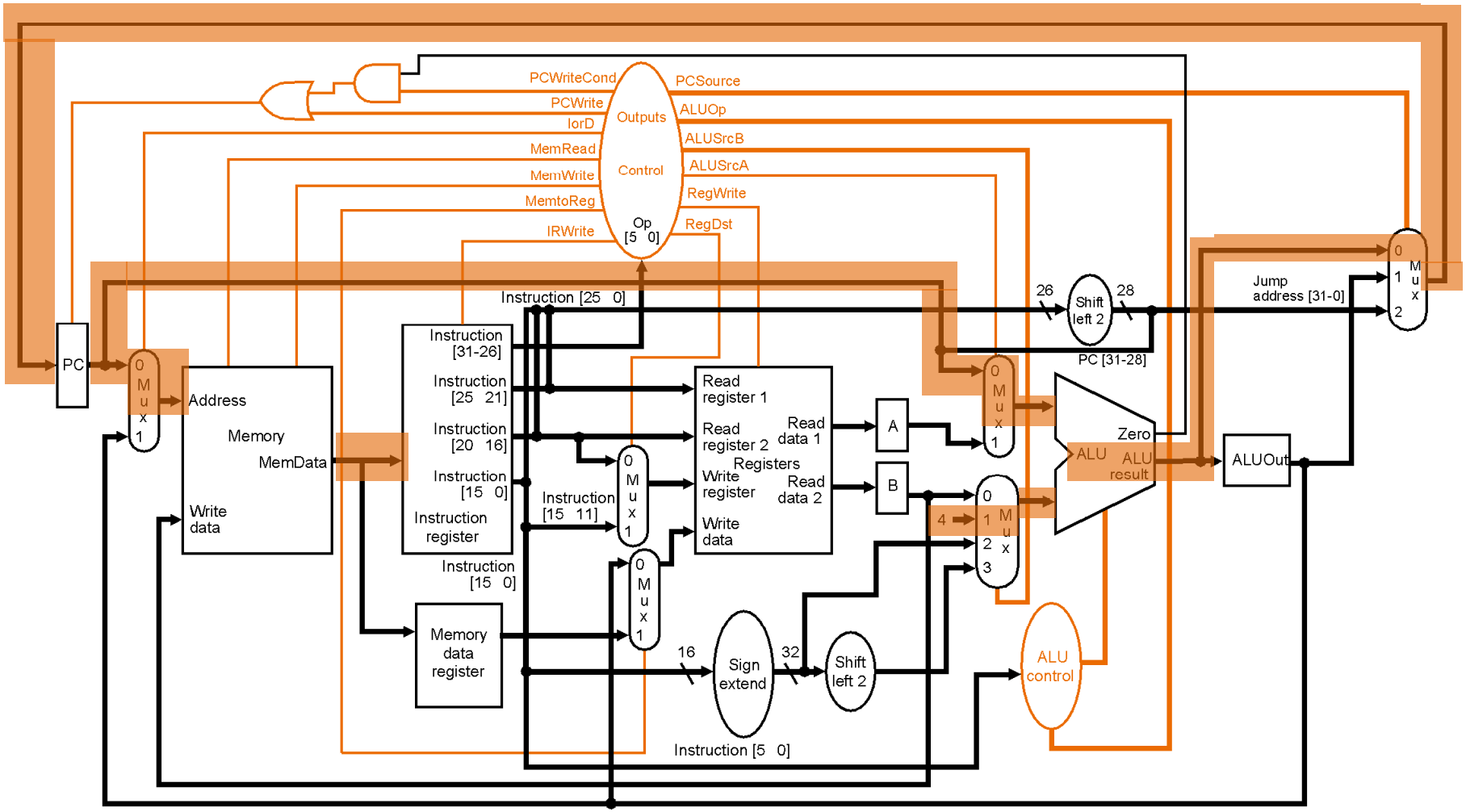


1. Instruction Fetch



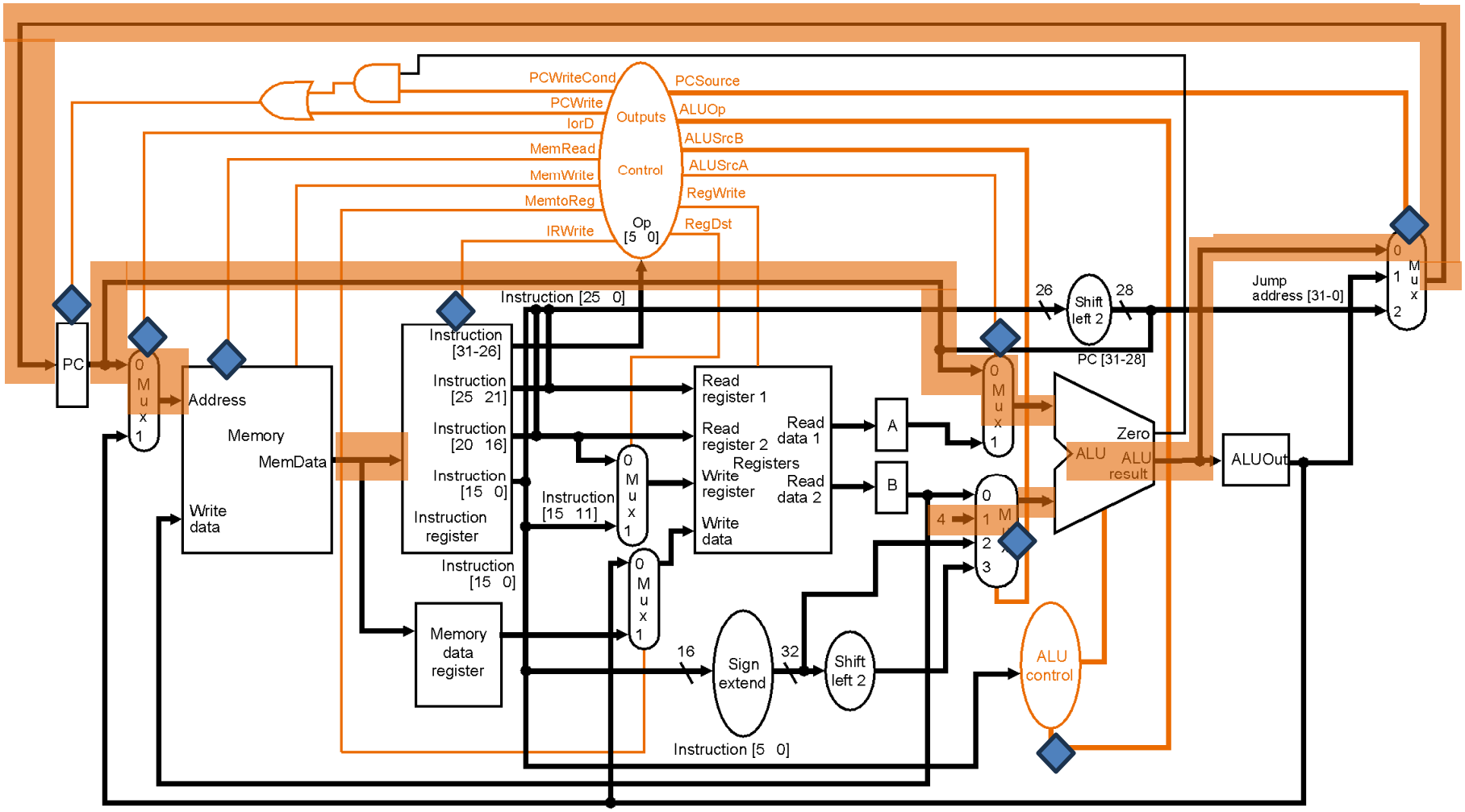
IR = Memory[PC]
PC = PC + 4

1. Instruction Fetch



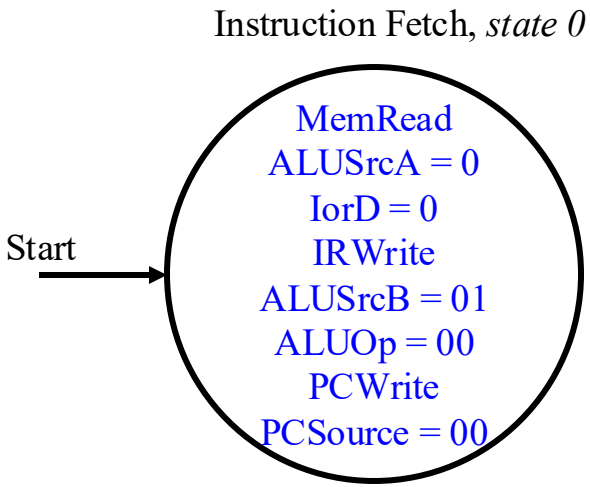
IR = Memory[PC]
PC = PC + 4

1. Instruction Fetch

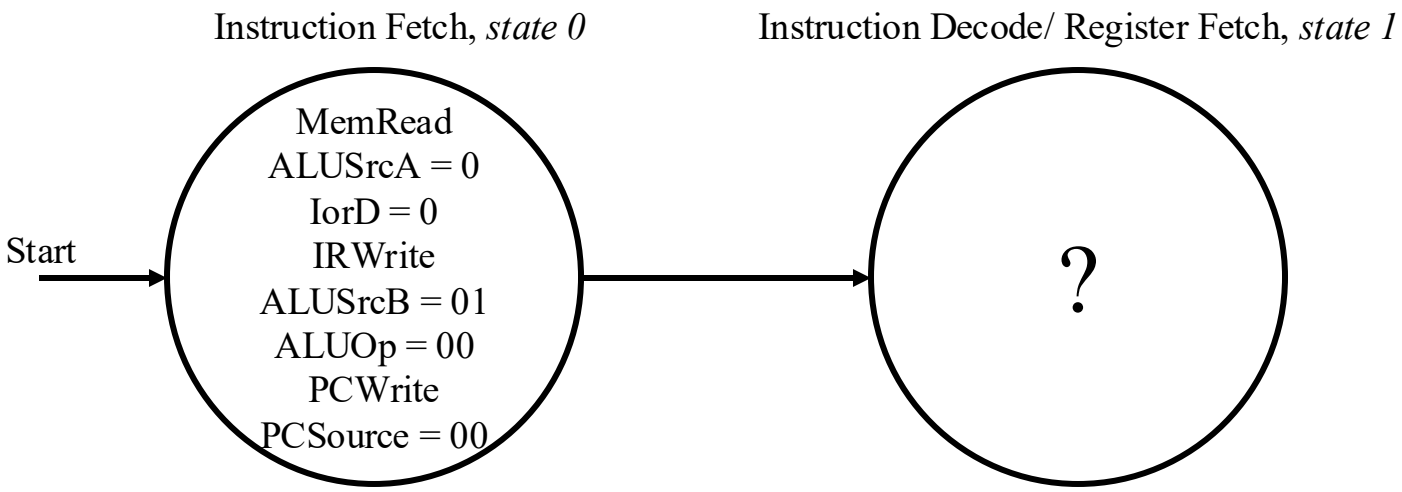


IR = Memory[PC]
PC = PC + 4

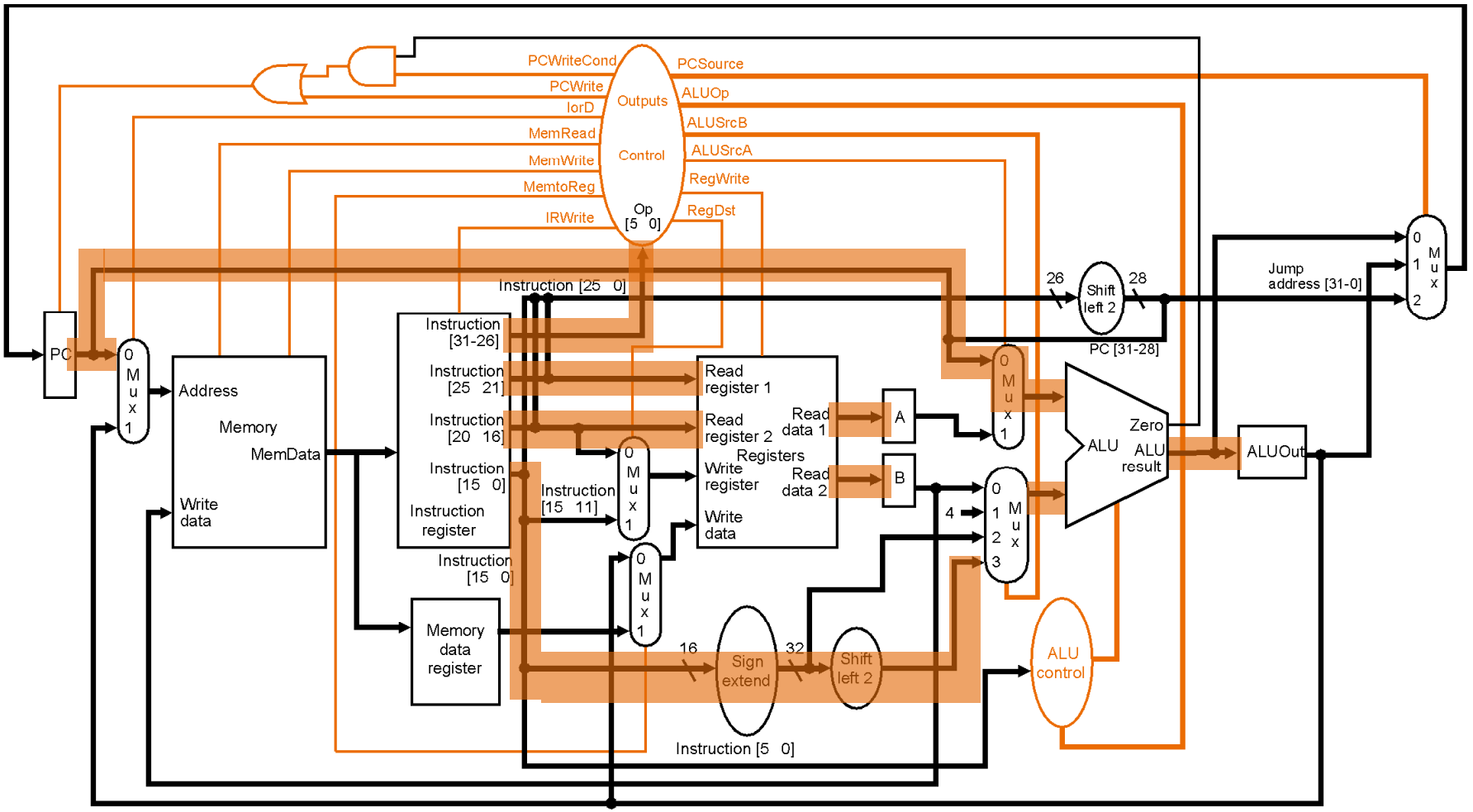
The first state of the FSM



The second state of the FSM

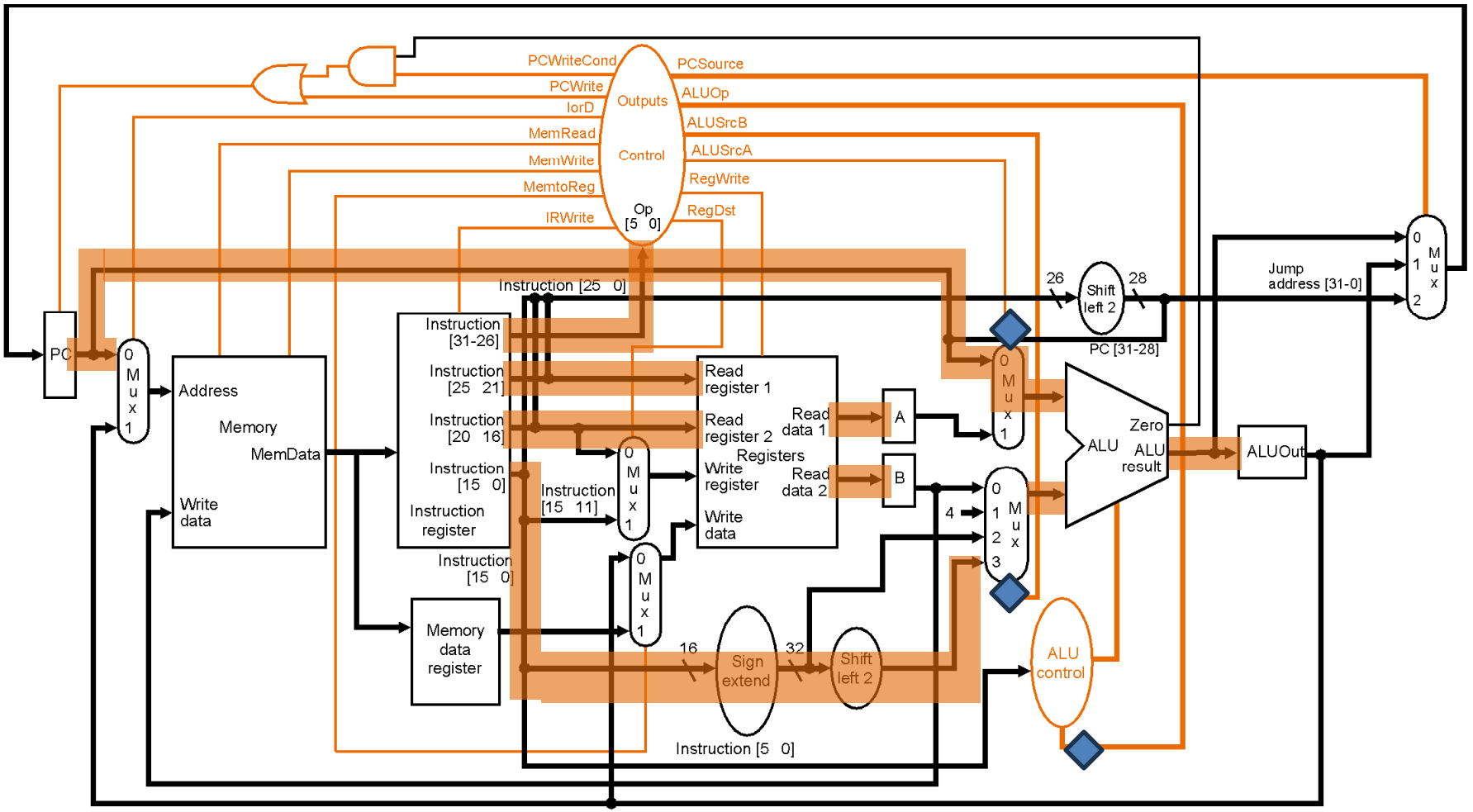


2. Instruction Decode and Register Fetch



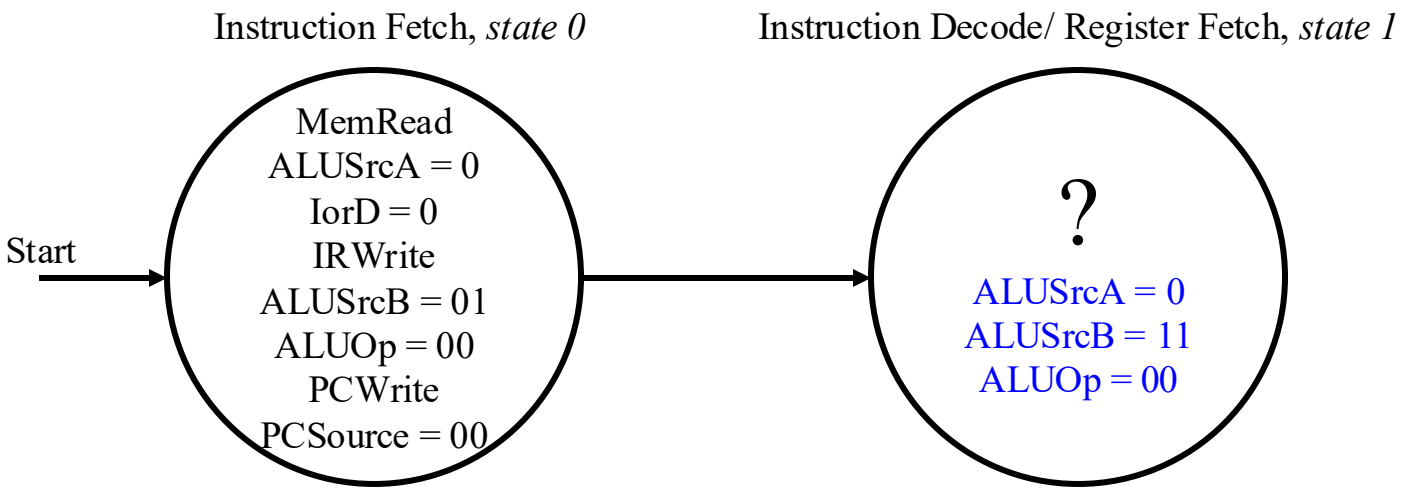
A = Register[IR[25-21]]
B = Register[IR[20-16]]
ALUOut = PC + (sign-extend (IR[15-0]) << 2)

2. Instruction Decode and Register Fetch

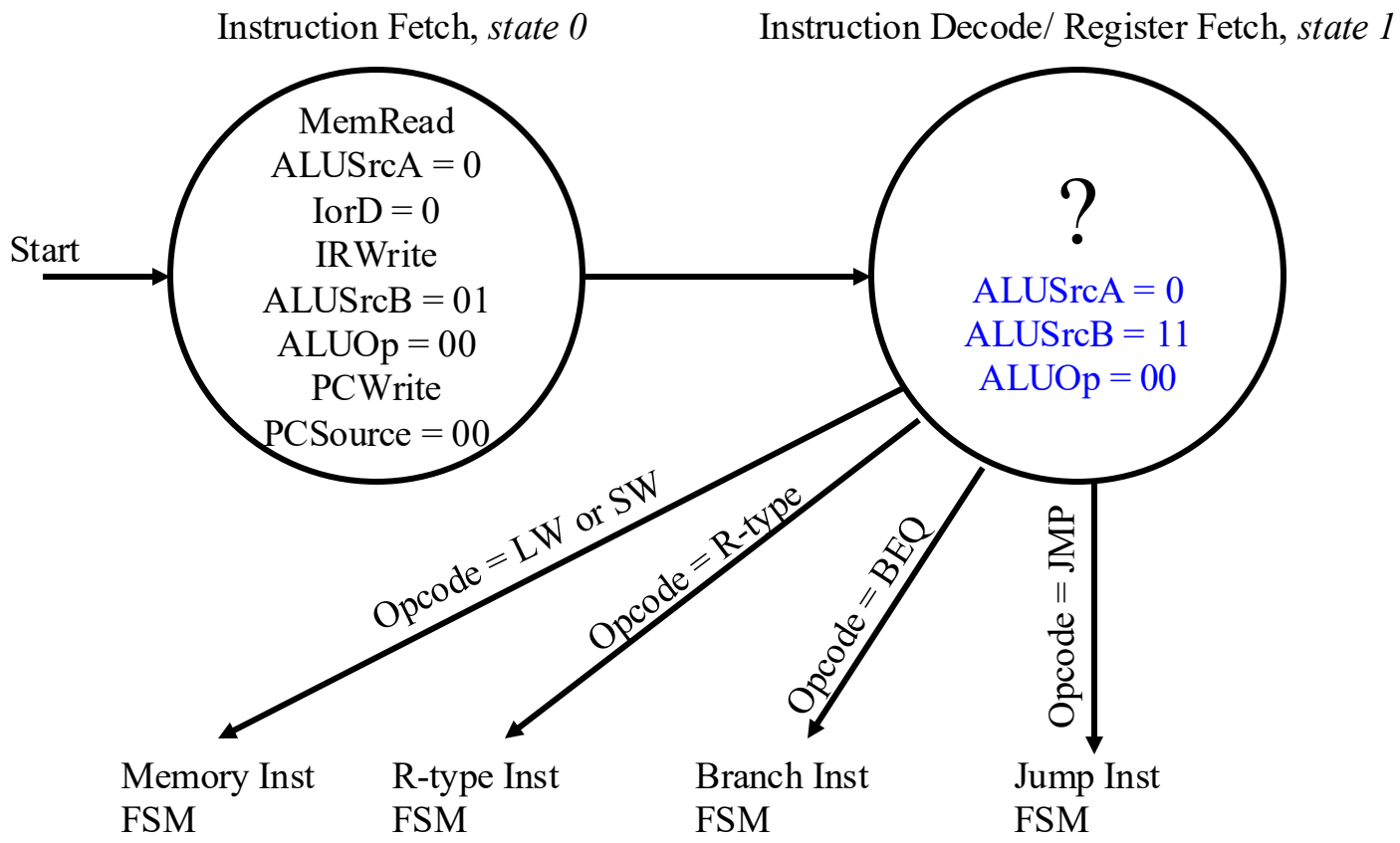


A = Register[IR[25-21]]
B = Register[IR[20-16]]
ALUOut = PC + (sign-extend (IR[15-0]) << 2)

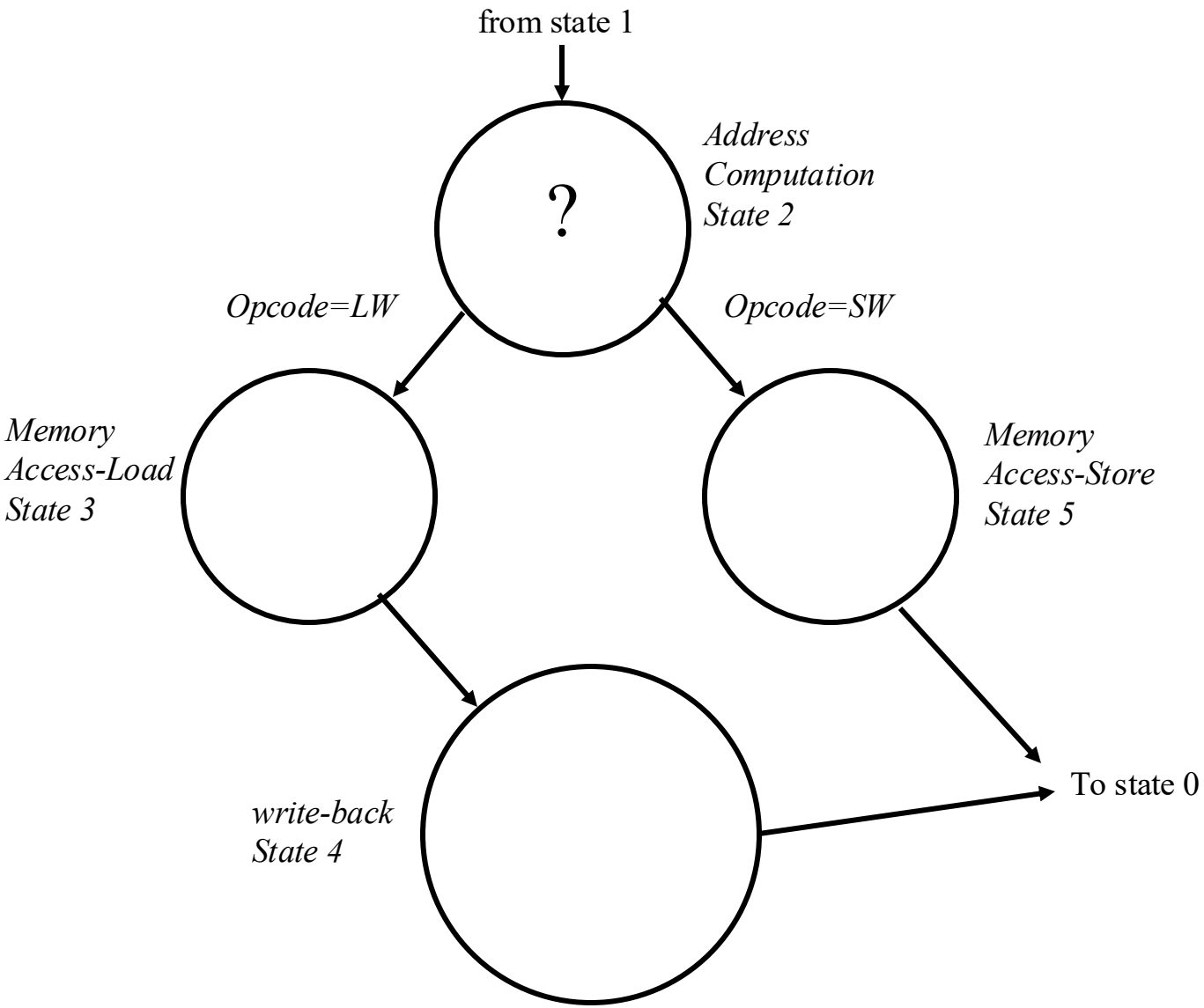
The second state of the FSM



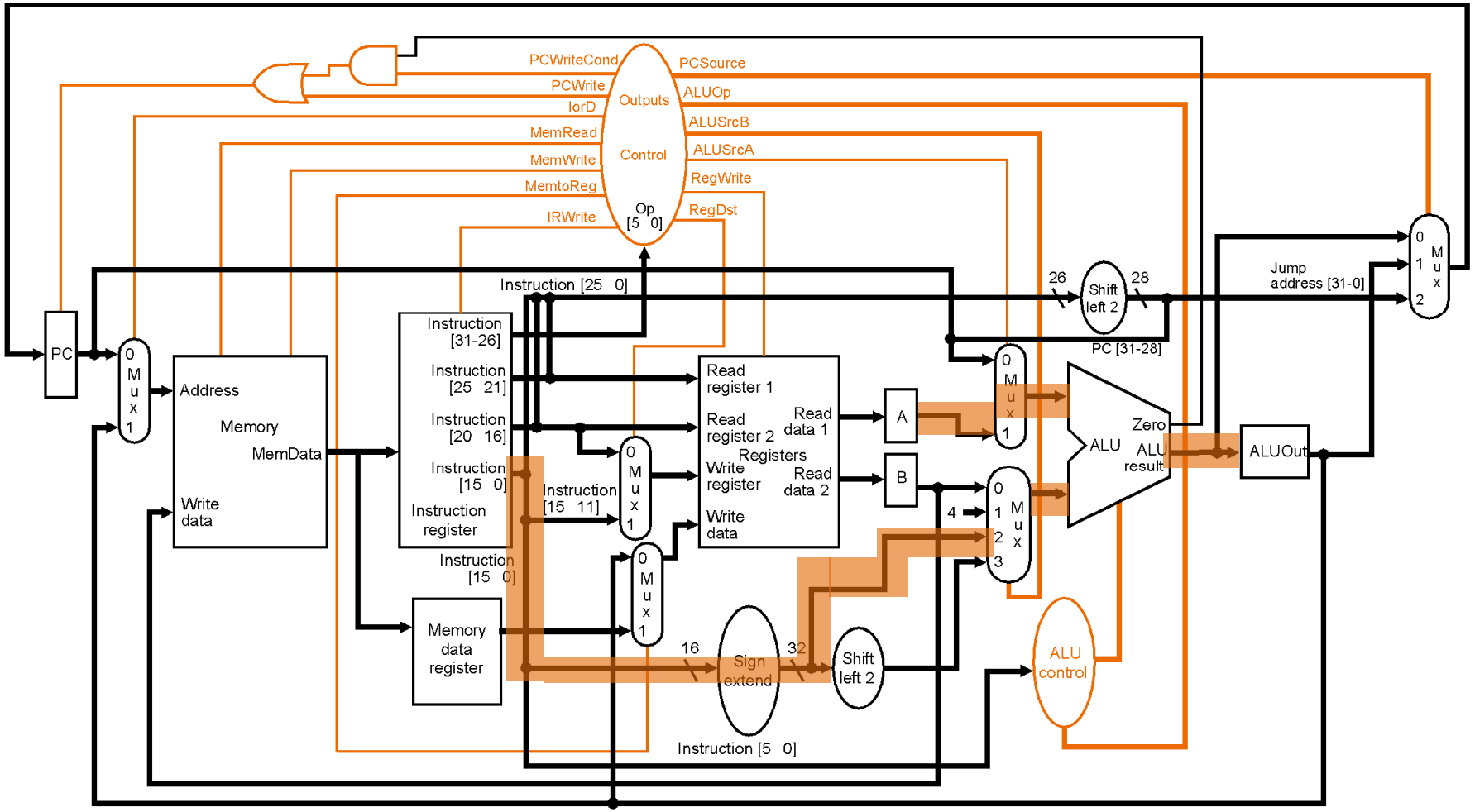
The second state of the FSM



Memory Instructions

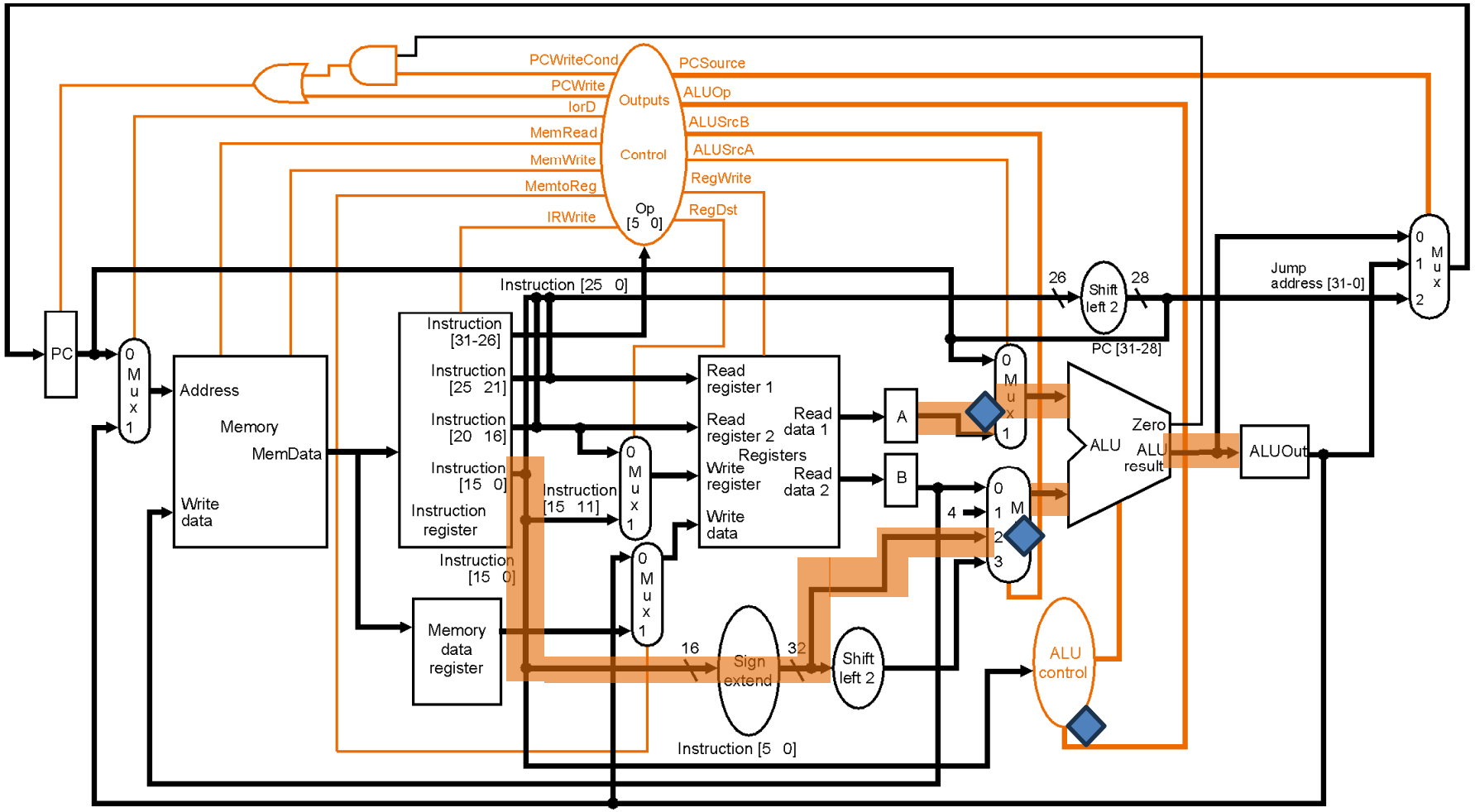


3. Memory Address Computation



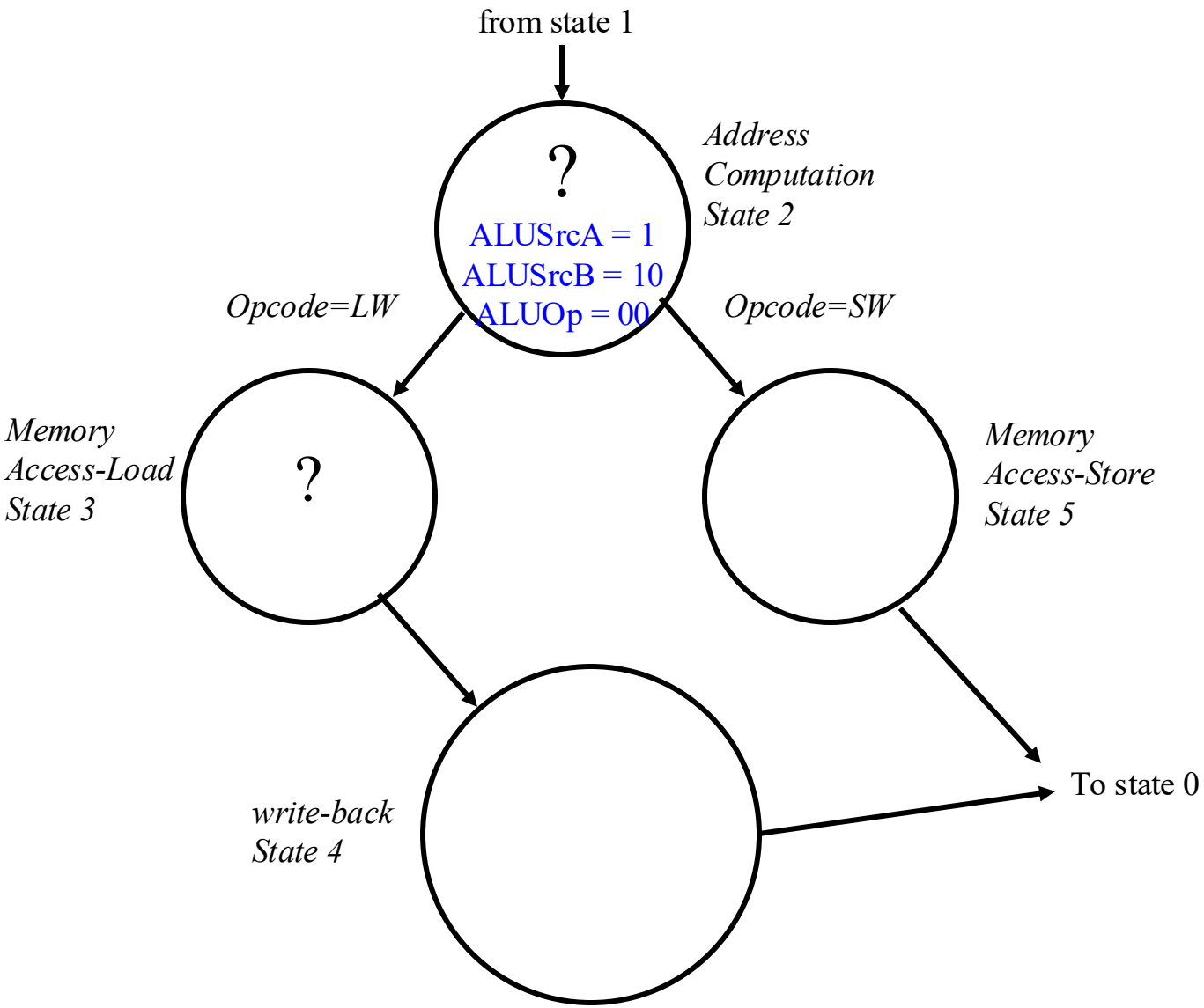
ALUout = A + sign-extend(IR[15-0])

3. Memory Address Computation

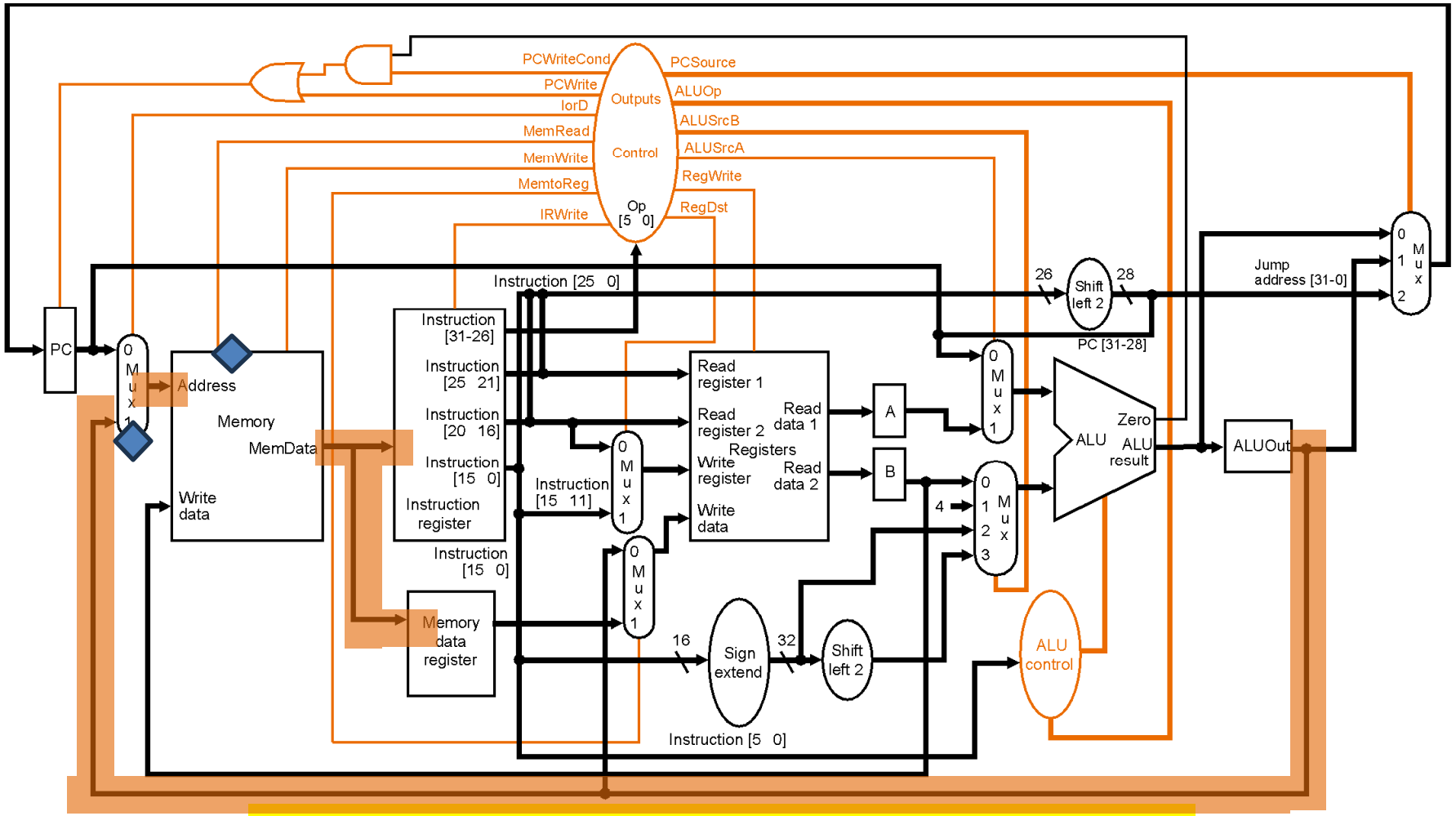


ALUout = A + sign-extend(IR[15-0])

Memory Instructions

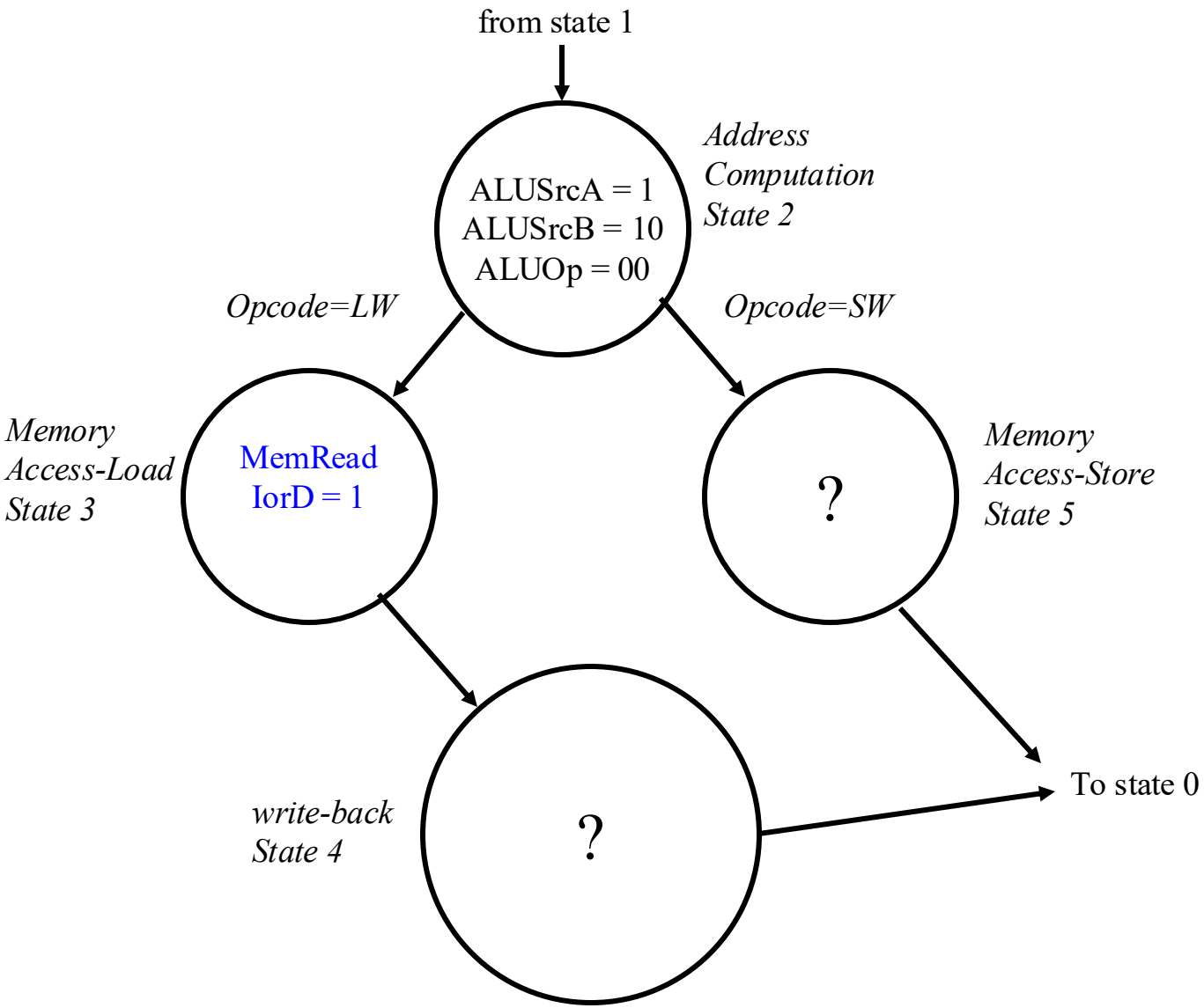


4. Memory Access Load

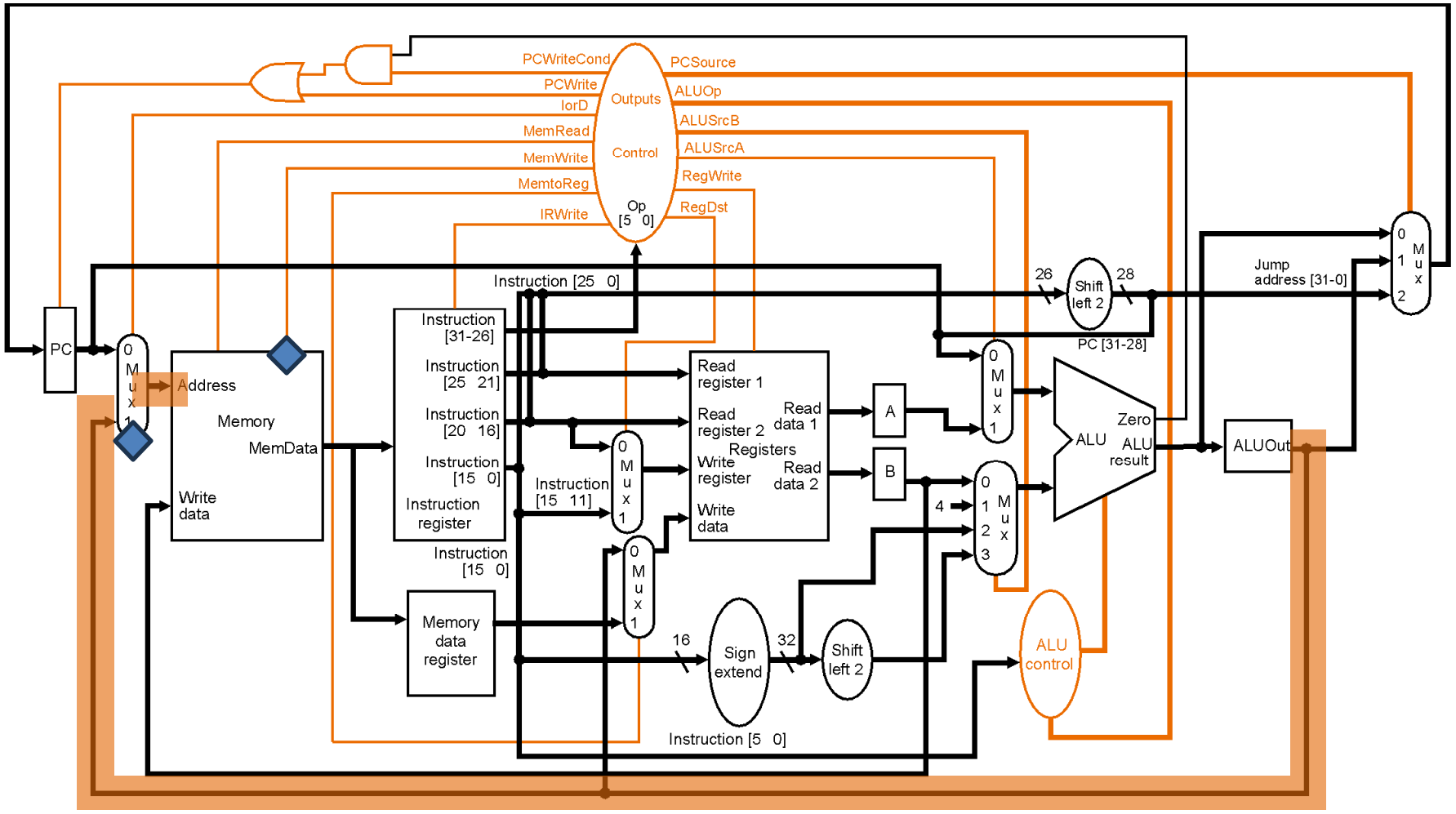


Memory-data-register = Memory[ALUout]

Memory Instructions

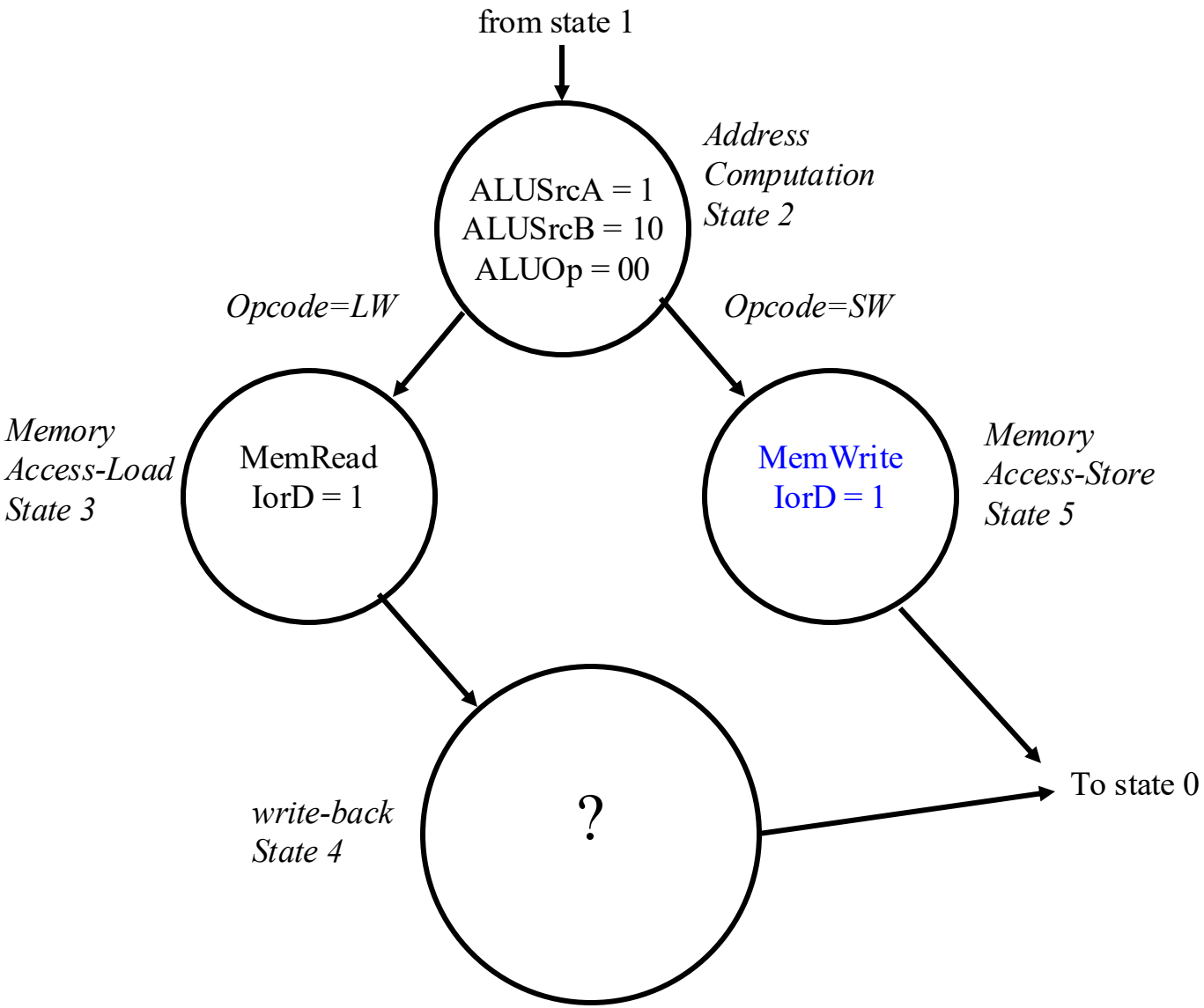


4. Memory Access Store

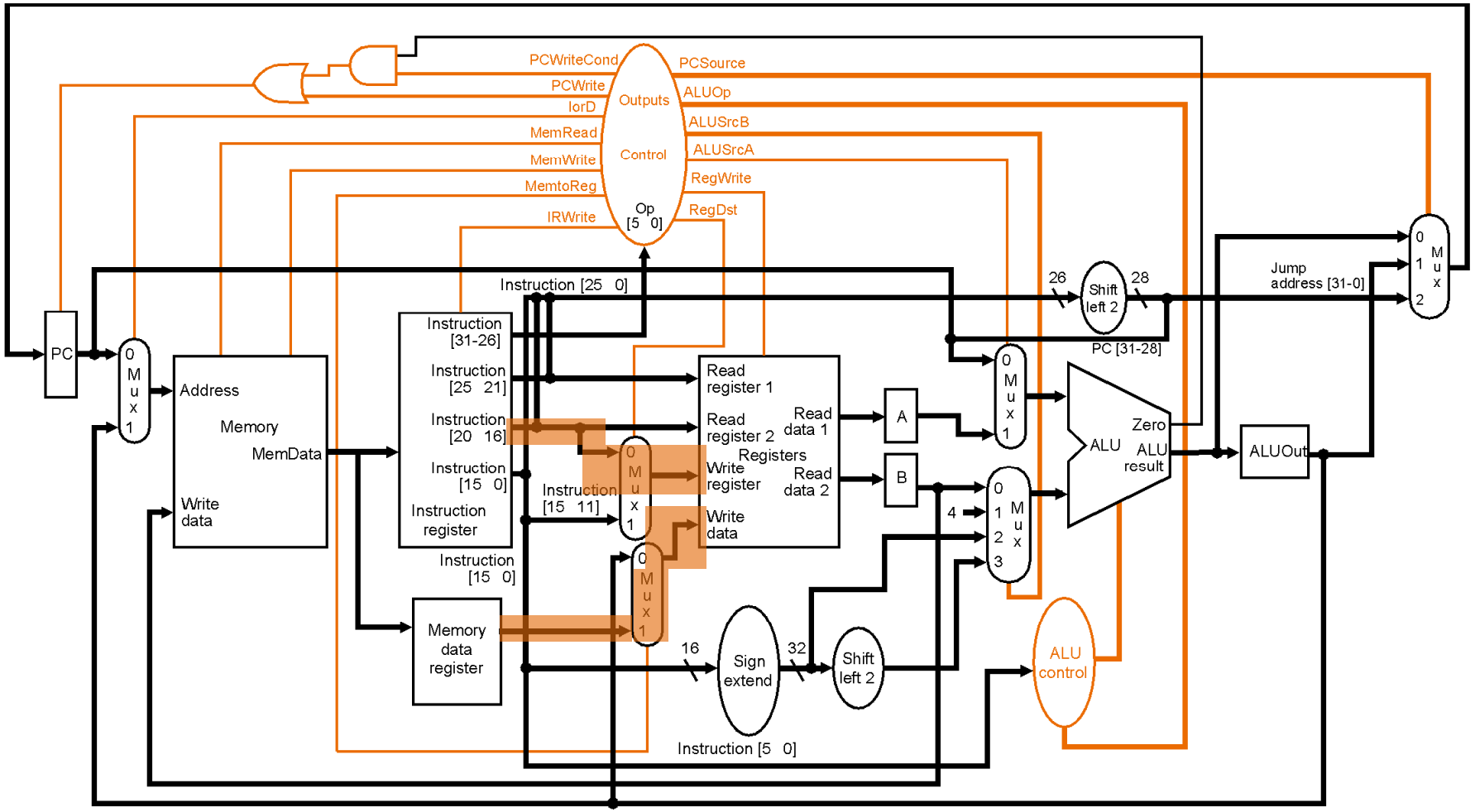


Memory[ALUout] = B

Memory Instructions

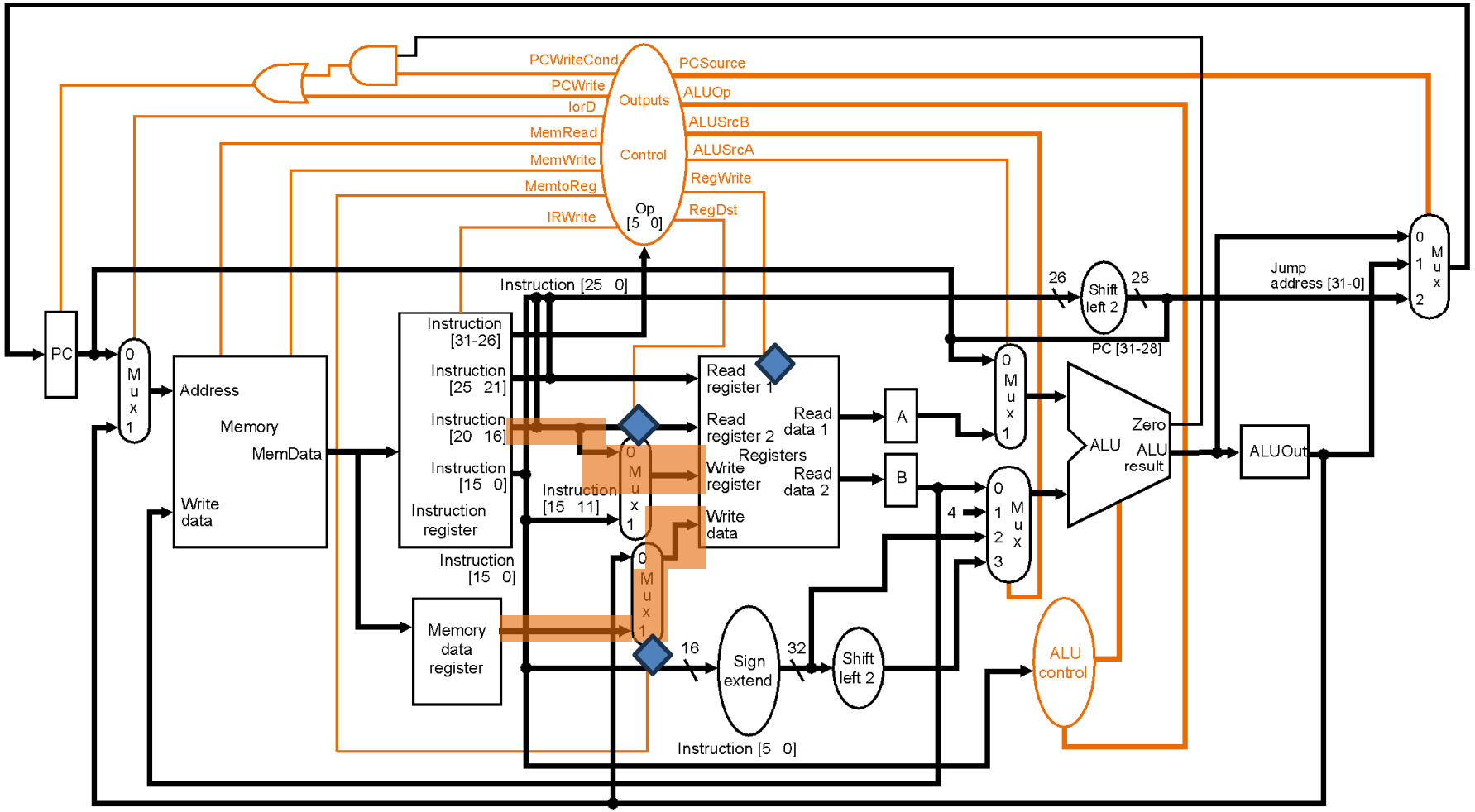


5. Load Write-Back



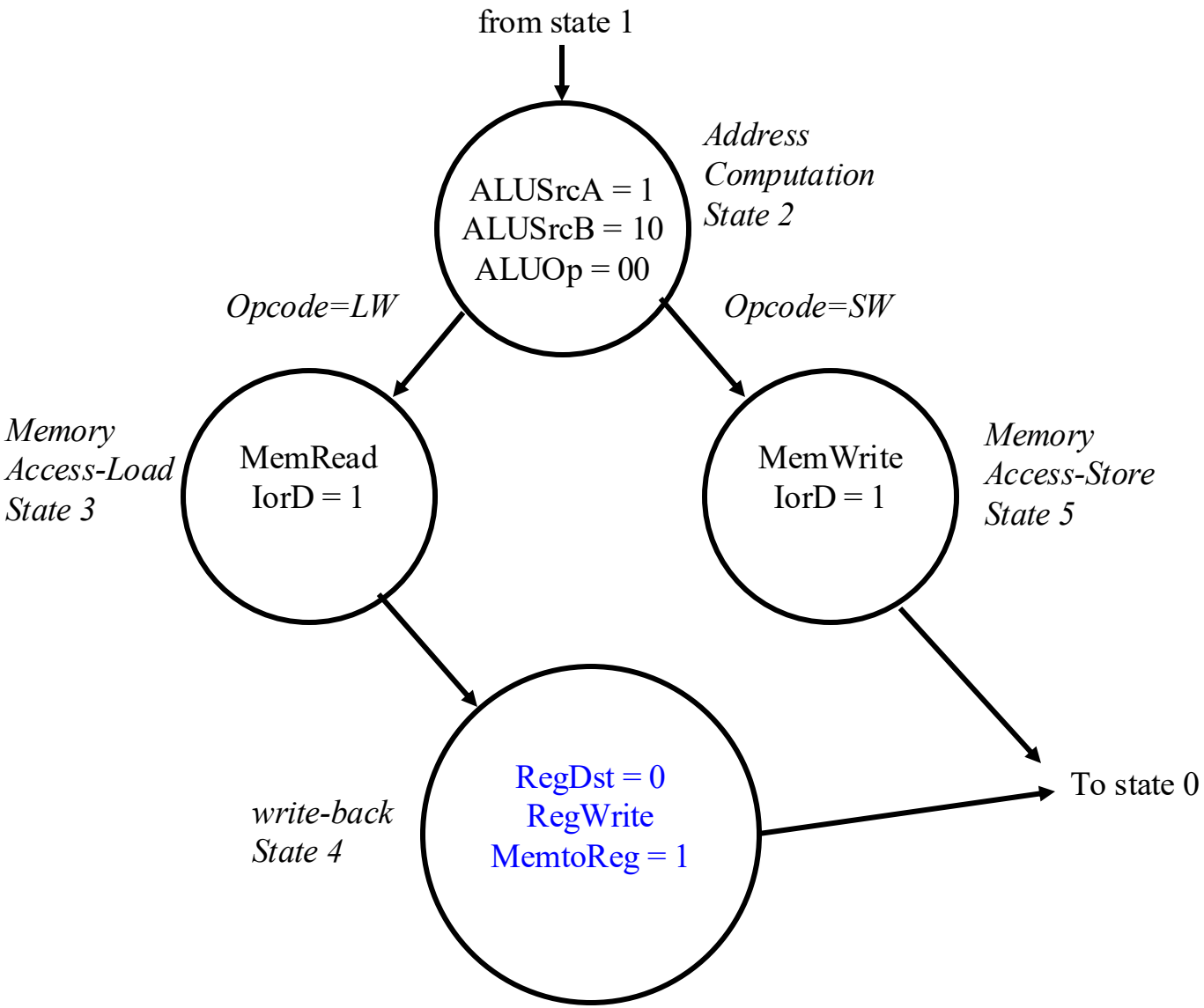
Reg[IR[20:16]] = Memory-data-register

5. Load Write-Back

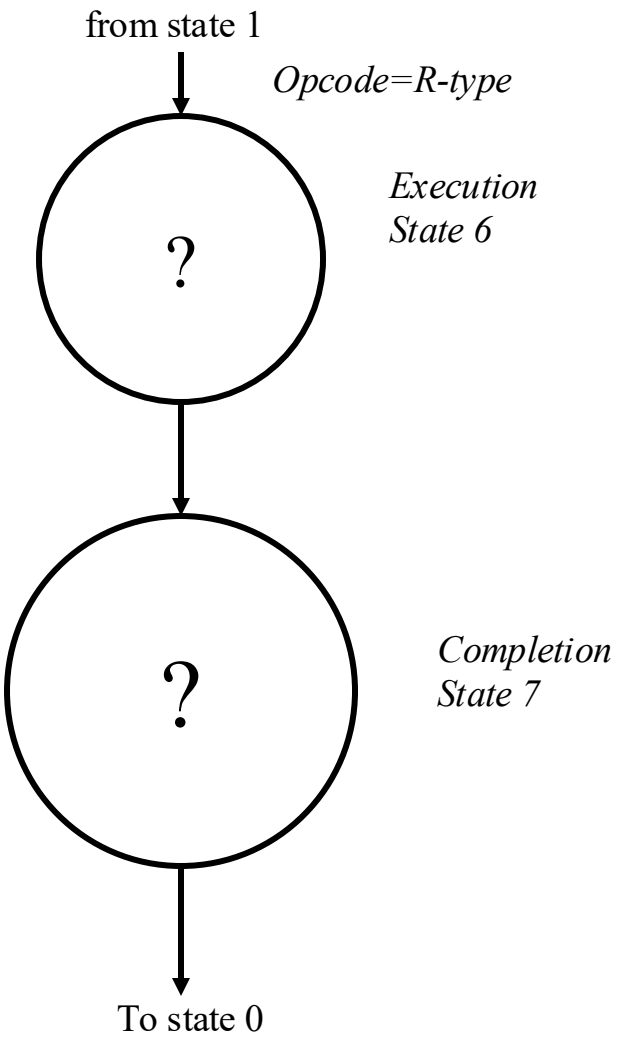


Reg[IR[20:16]] = Memory-data-register

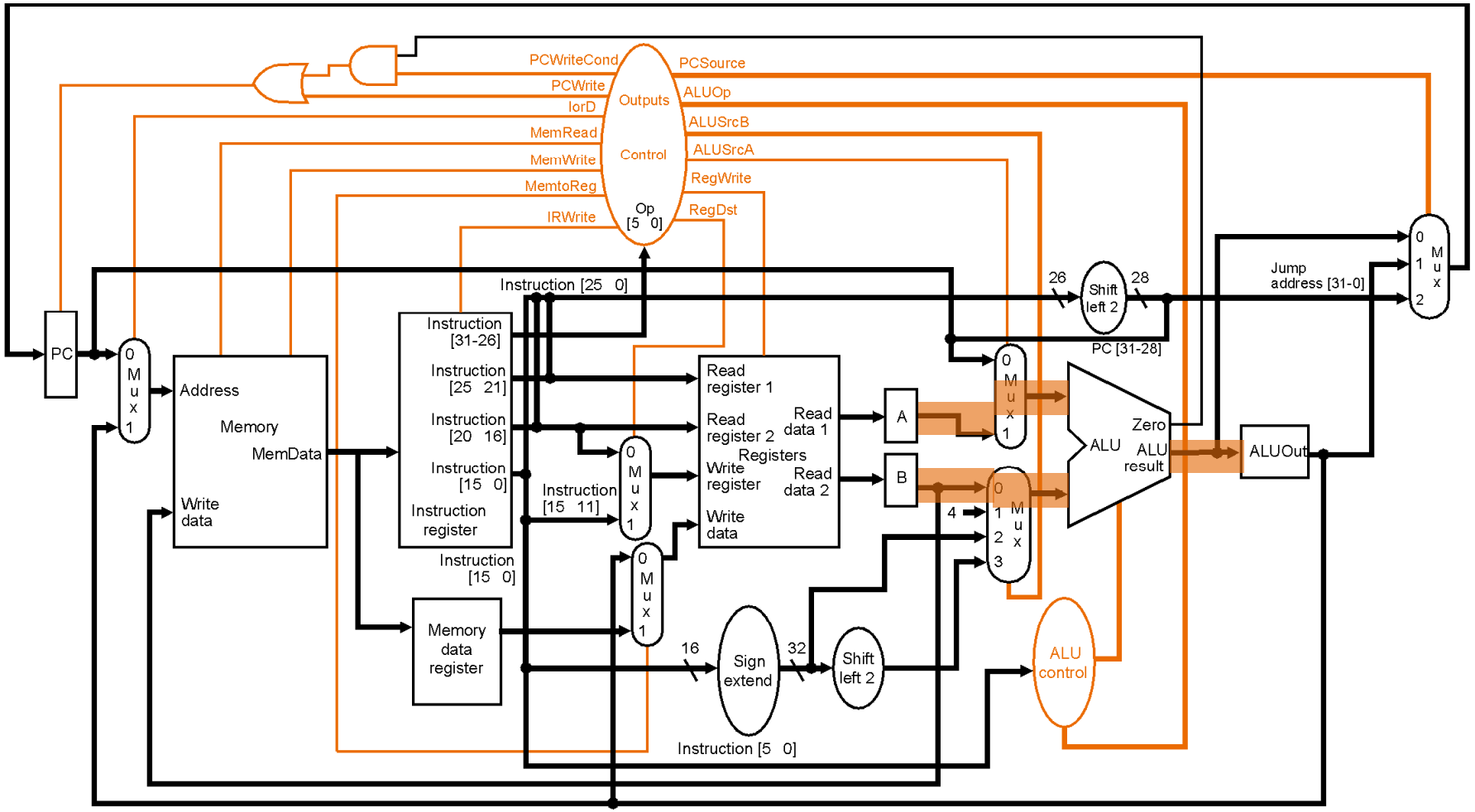
Memory Instructions



R-type Instructions

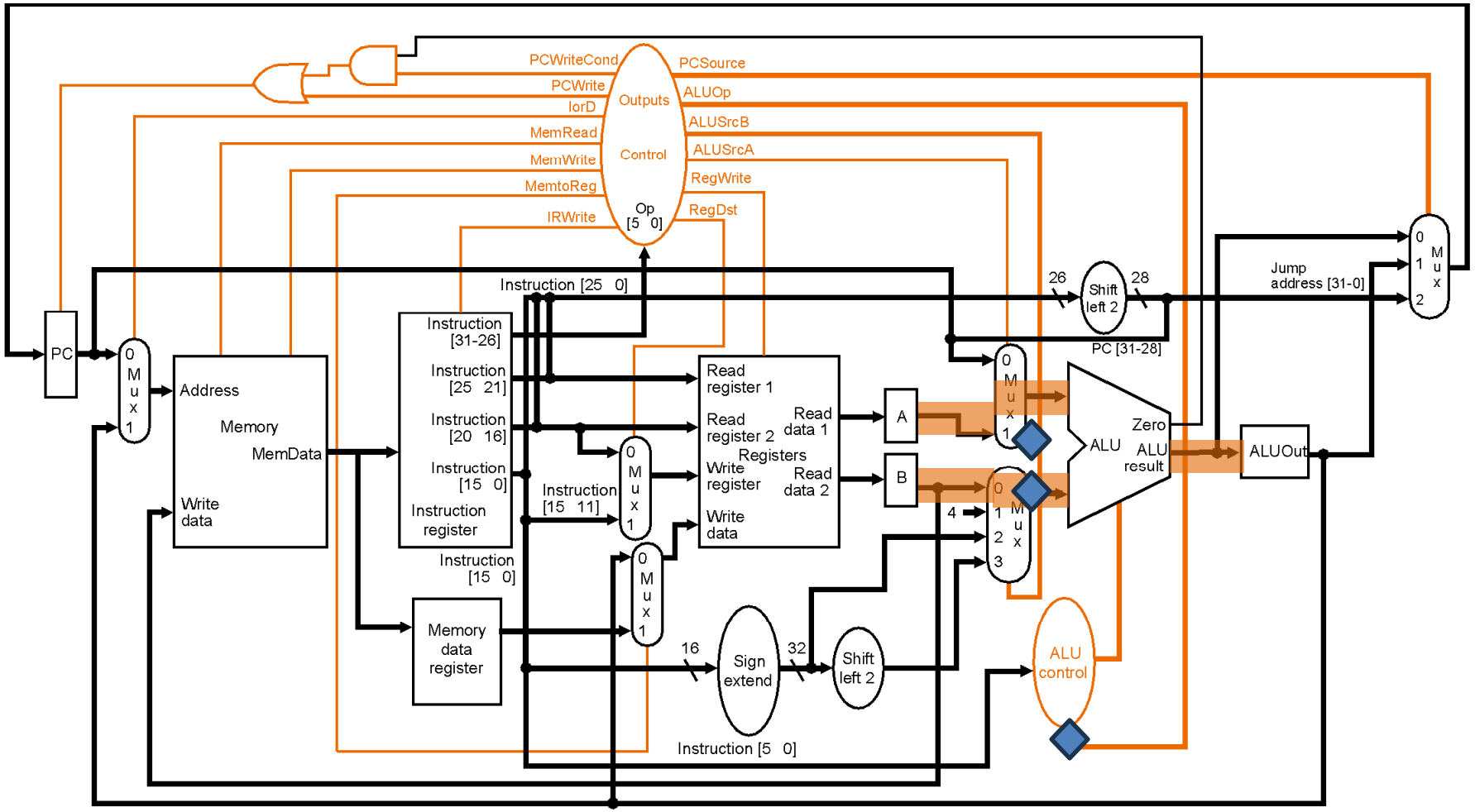


3. Execution (R-Type)



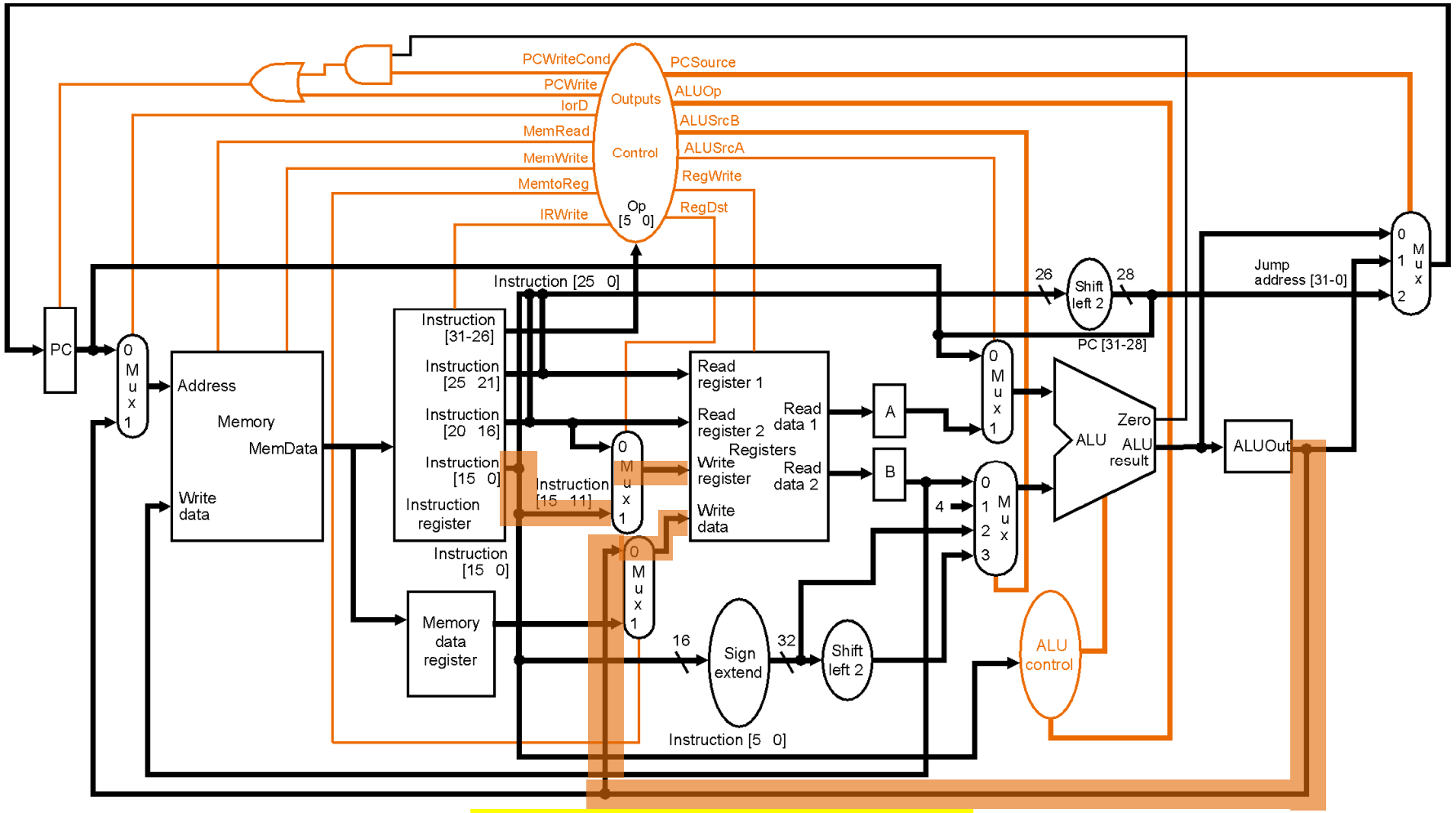
ALUout = A op B

3. Execution (R-Type)



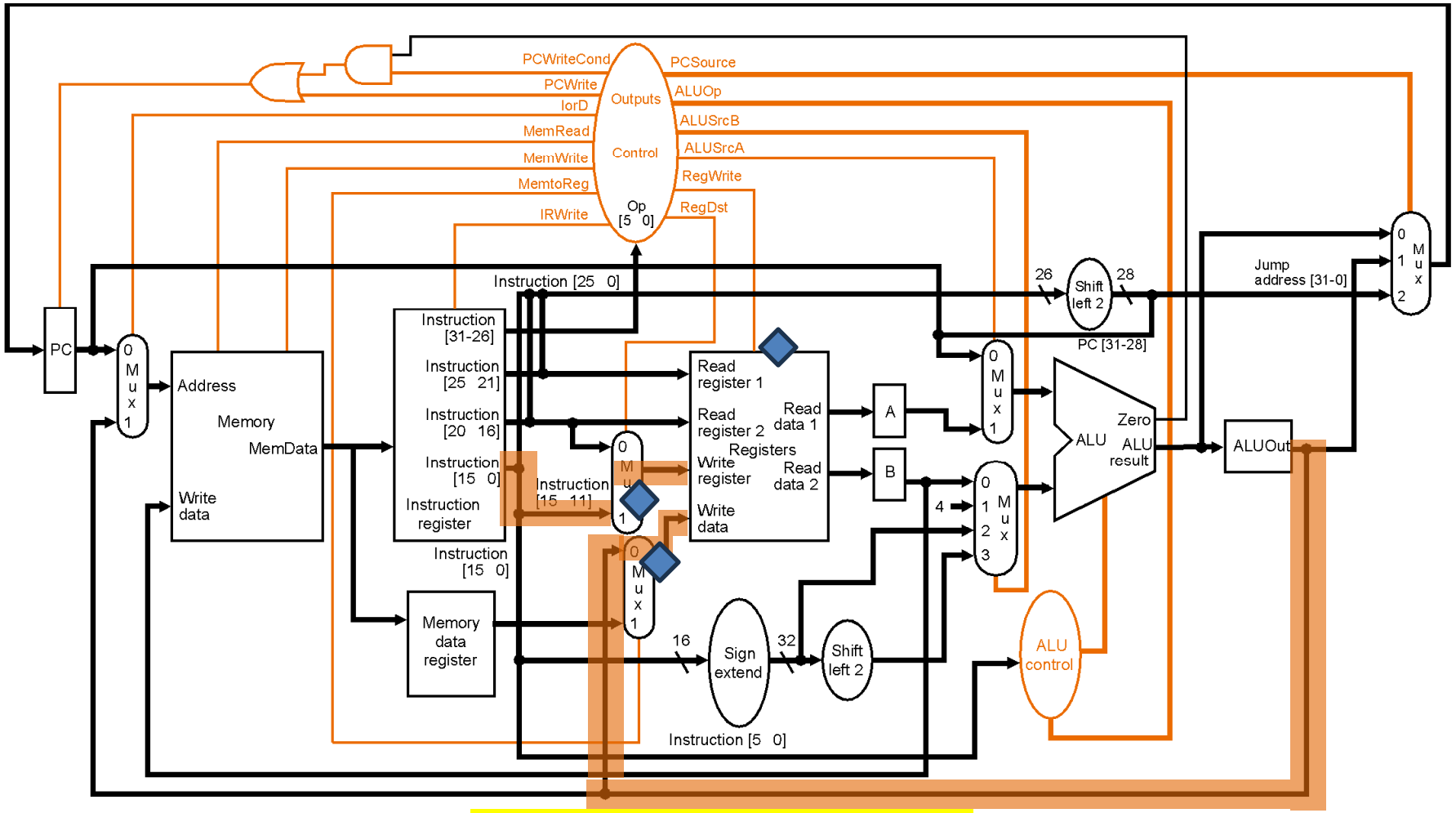
ALUout = A op B

4. R-Type Completion



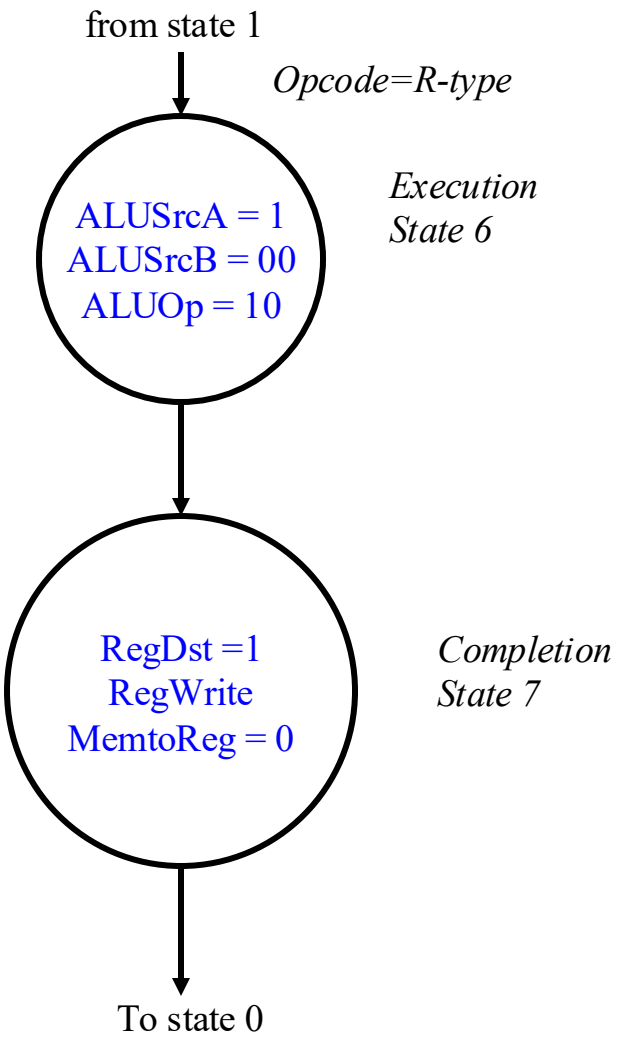
Reg[IR[15-11]] = ALUout

4. R-Type Completion

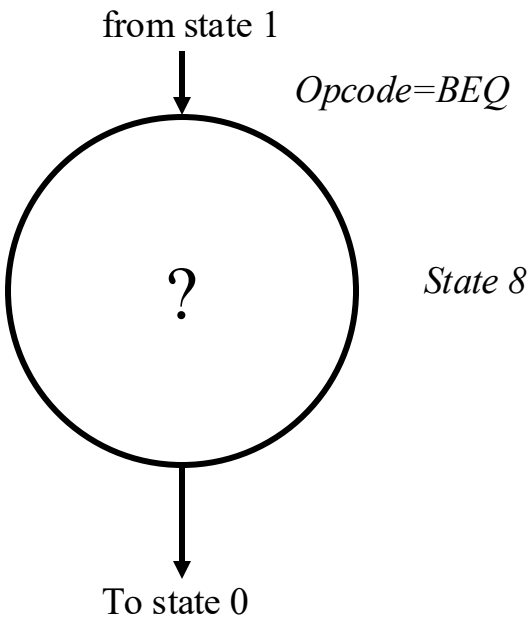


Reg[IR[15-11]] = ALUout

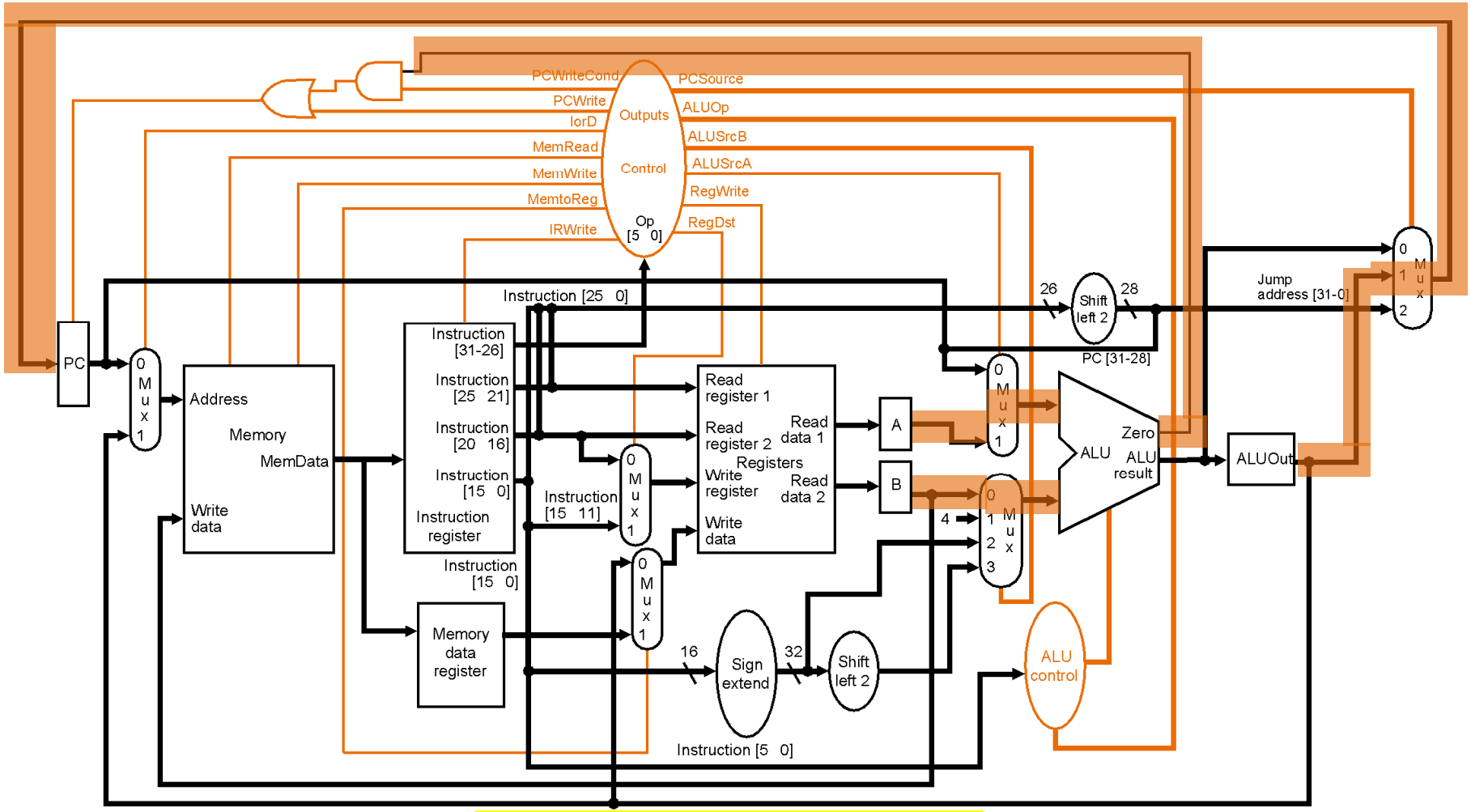
R-type Instructions



BEQ Instruction

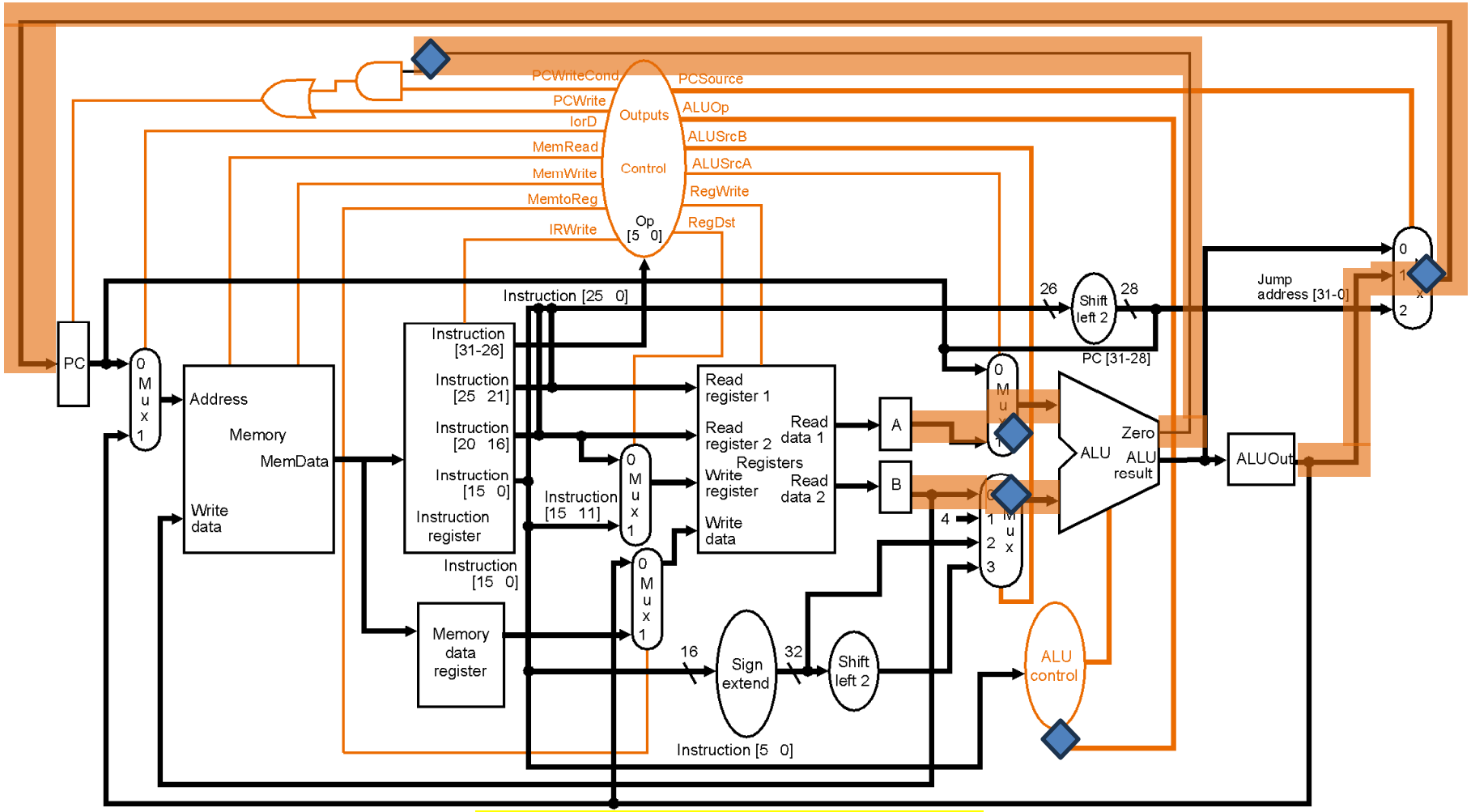


3. Branch Completion



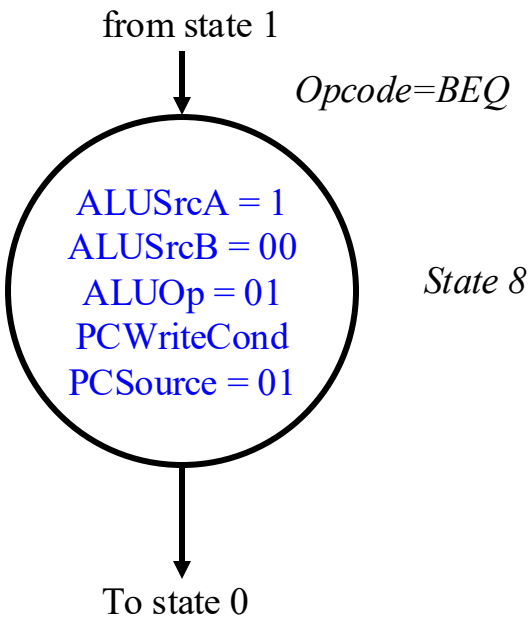
if (A == B) PC = ALUOut

3. Branch Completion

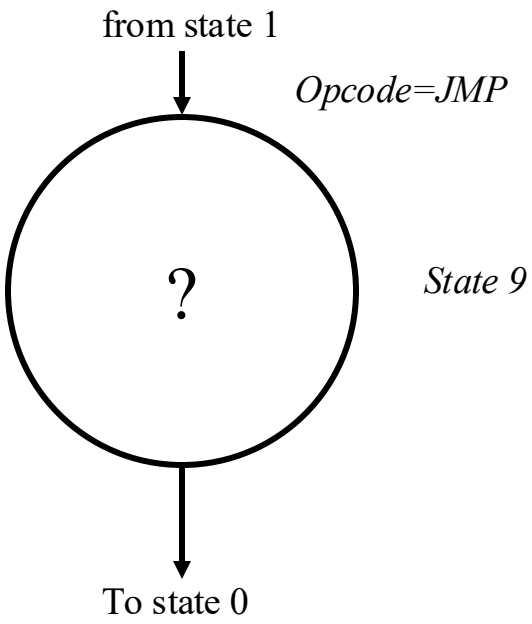


if (A == B) PC = ALUOut

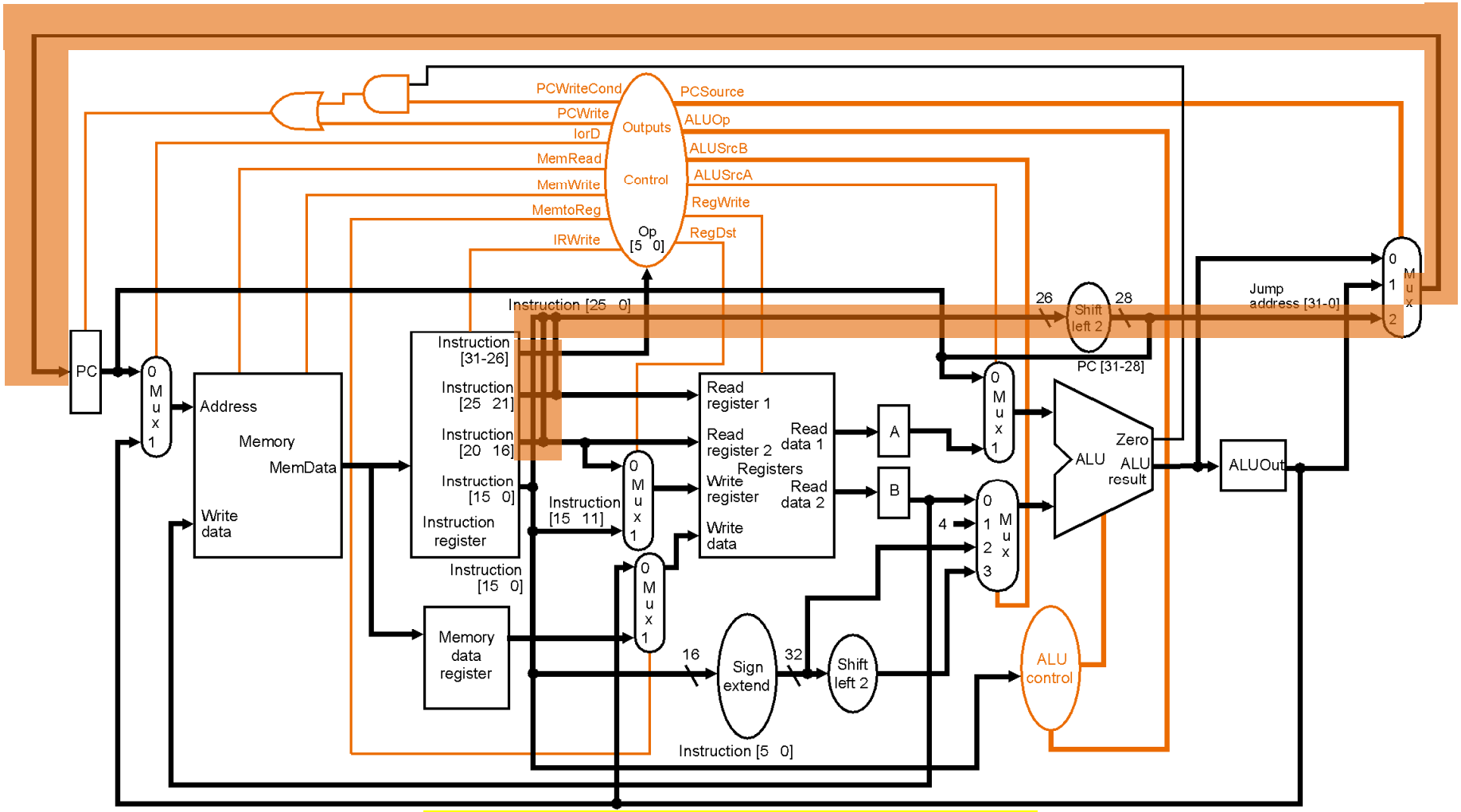
BEQ Instruction



JMP Instruction

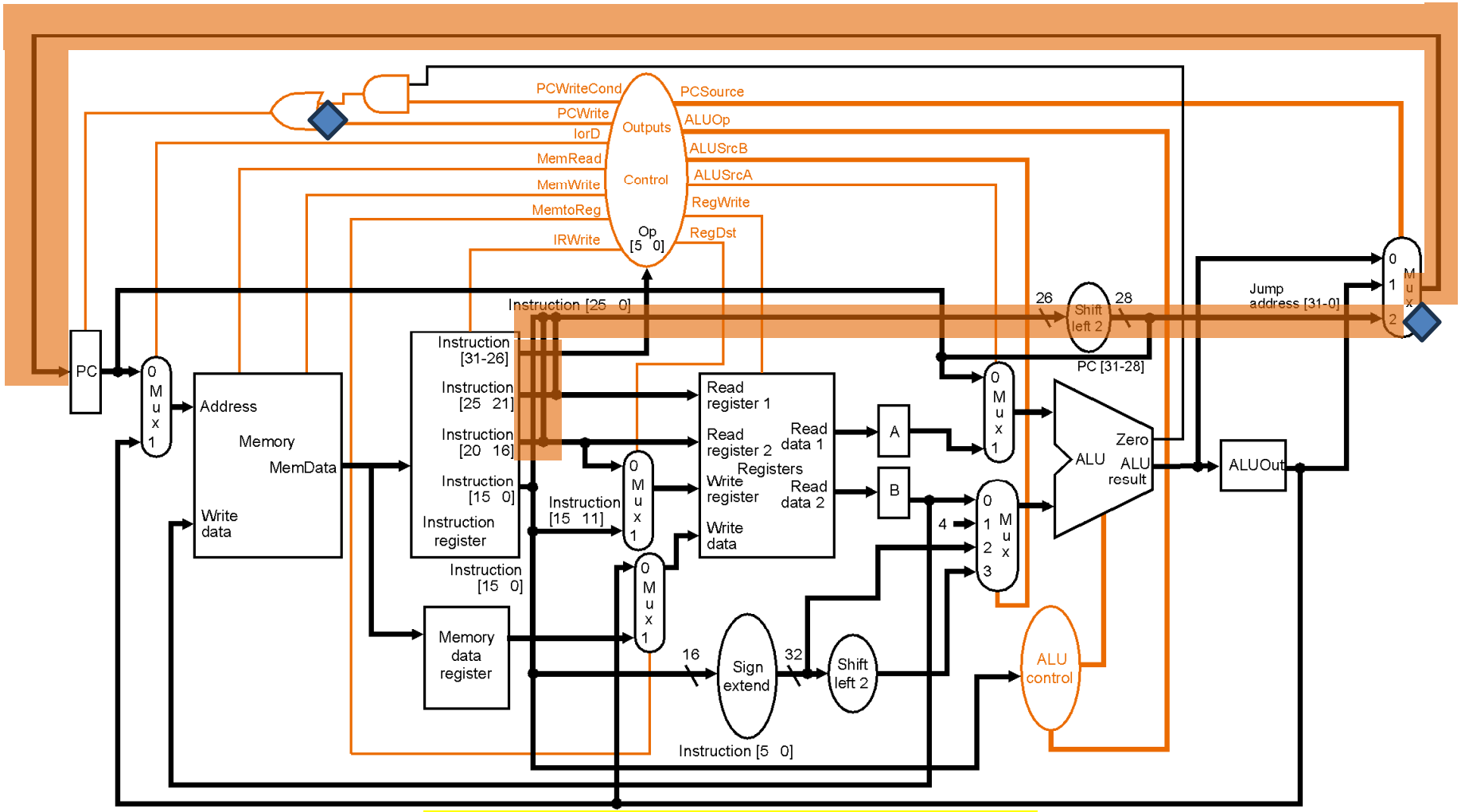


3. Jump Completion



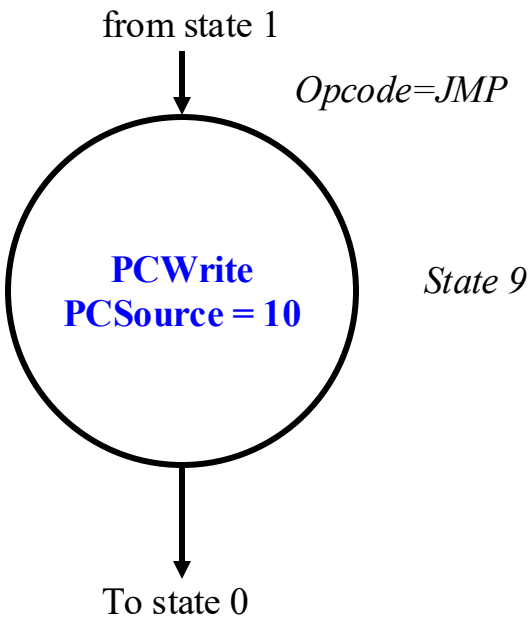
PC = {PC[31:28], (IR[25:0]<<2)}

3. Jump Completion

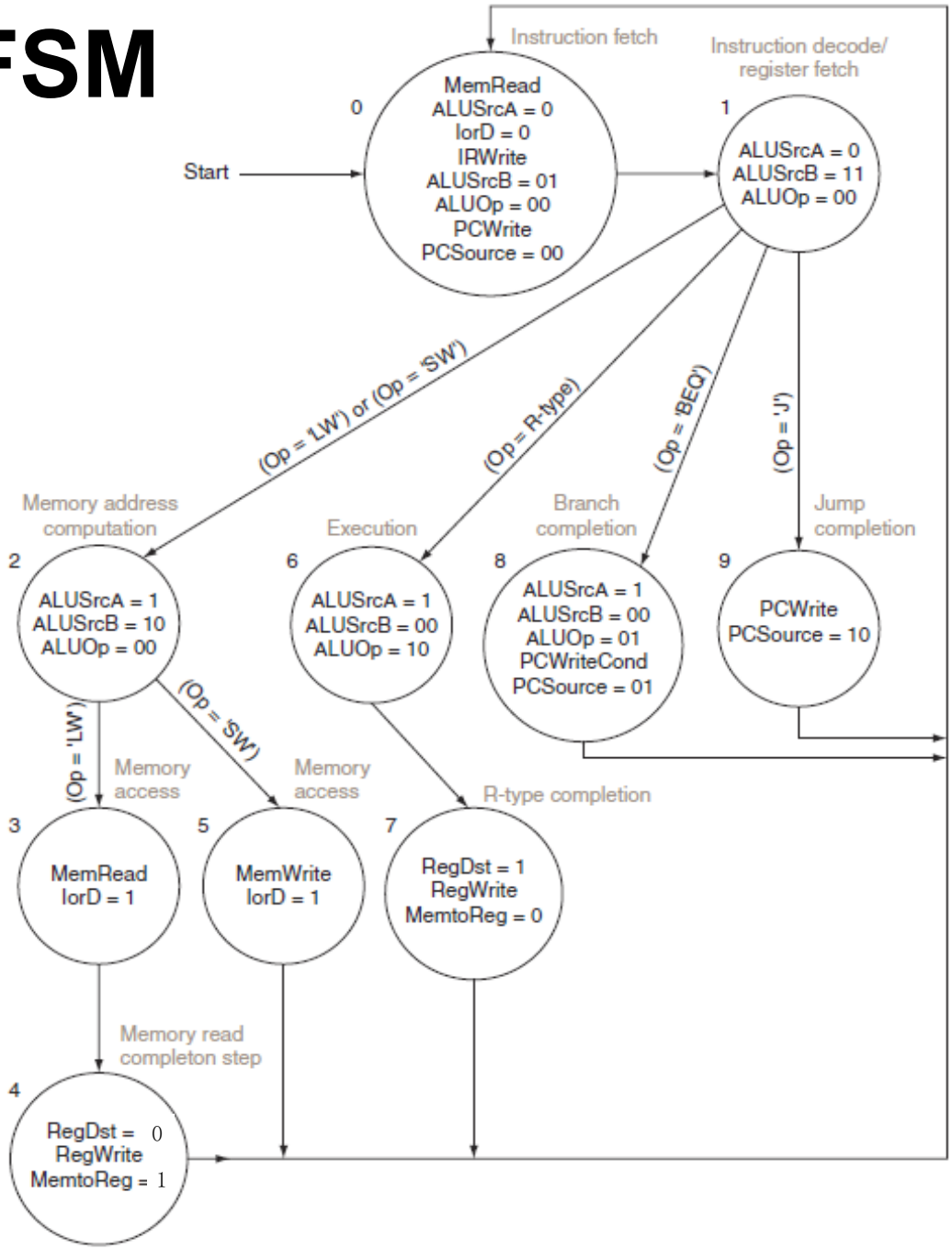


PC = {PC[31:28], (IR[25:0]<<2)}

JMP Instruction

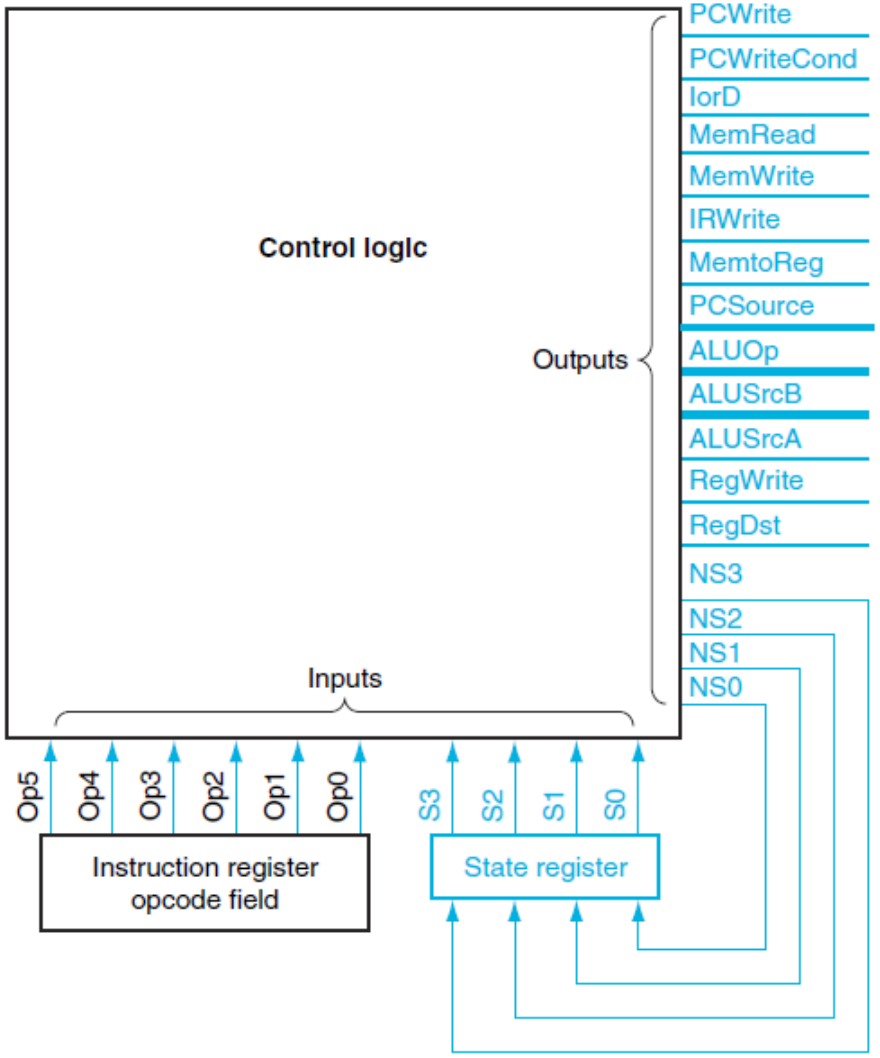


The Whole FSM



Implementing FSM Controller

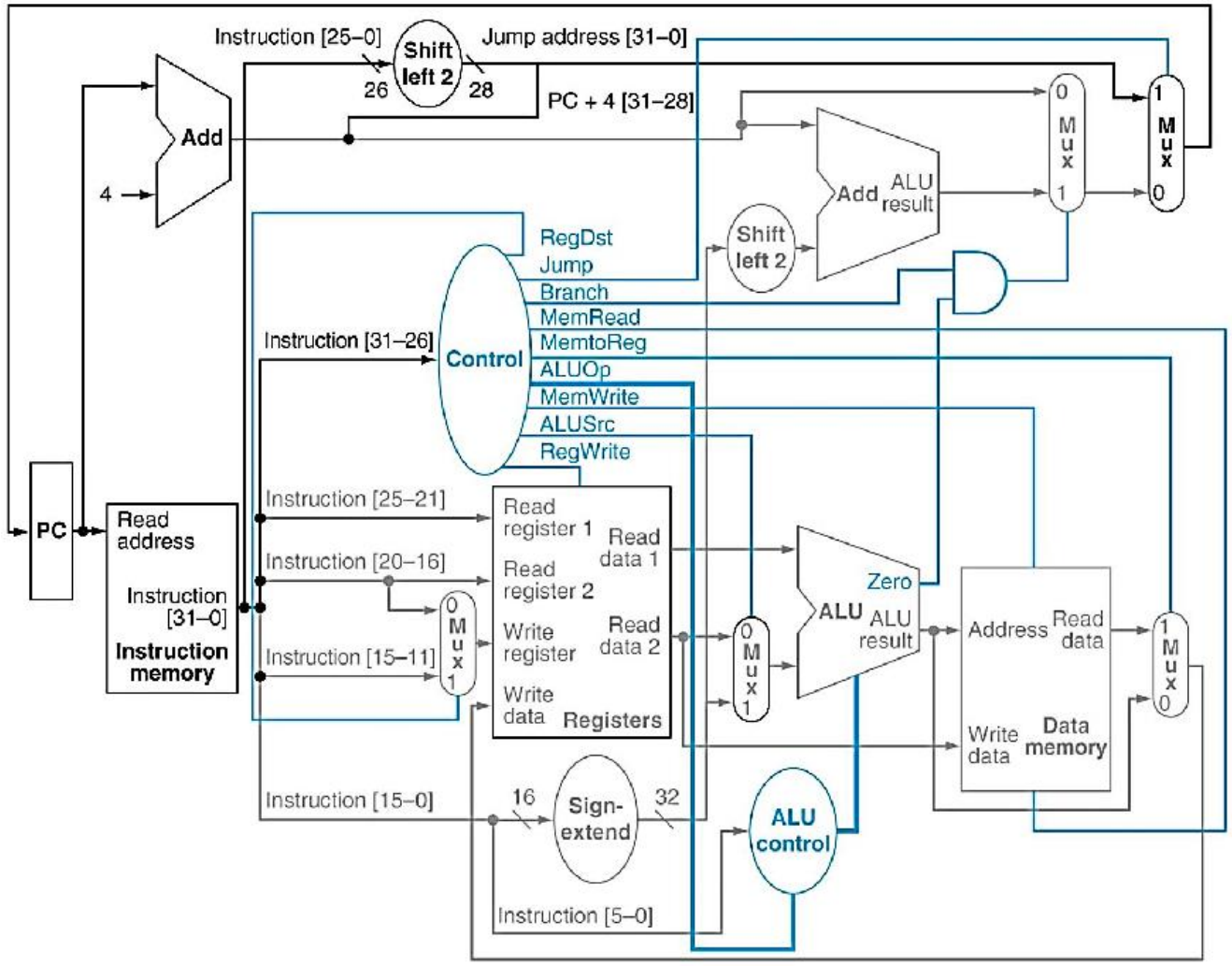
- State register that holds the current state
  - S3, S2, S1, and S0 (4-bit reg)
  - Or one-hot code (?-bit reg)
- Combinational logic block to compute the next state and outputs.
- Control outputs depend only on the current state, instead of the inputs (Moore FSM)



补充知识：one-hot encoding

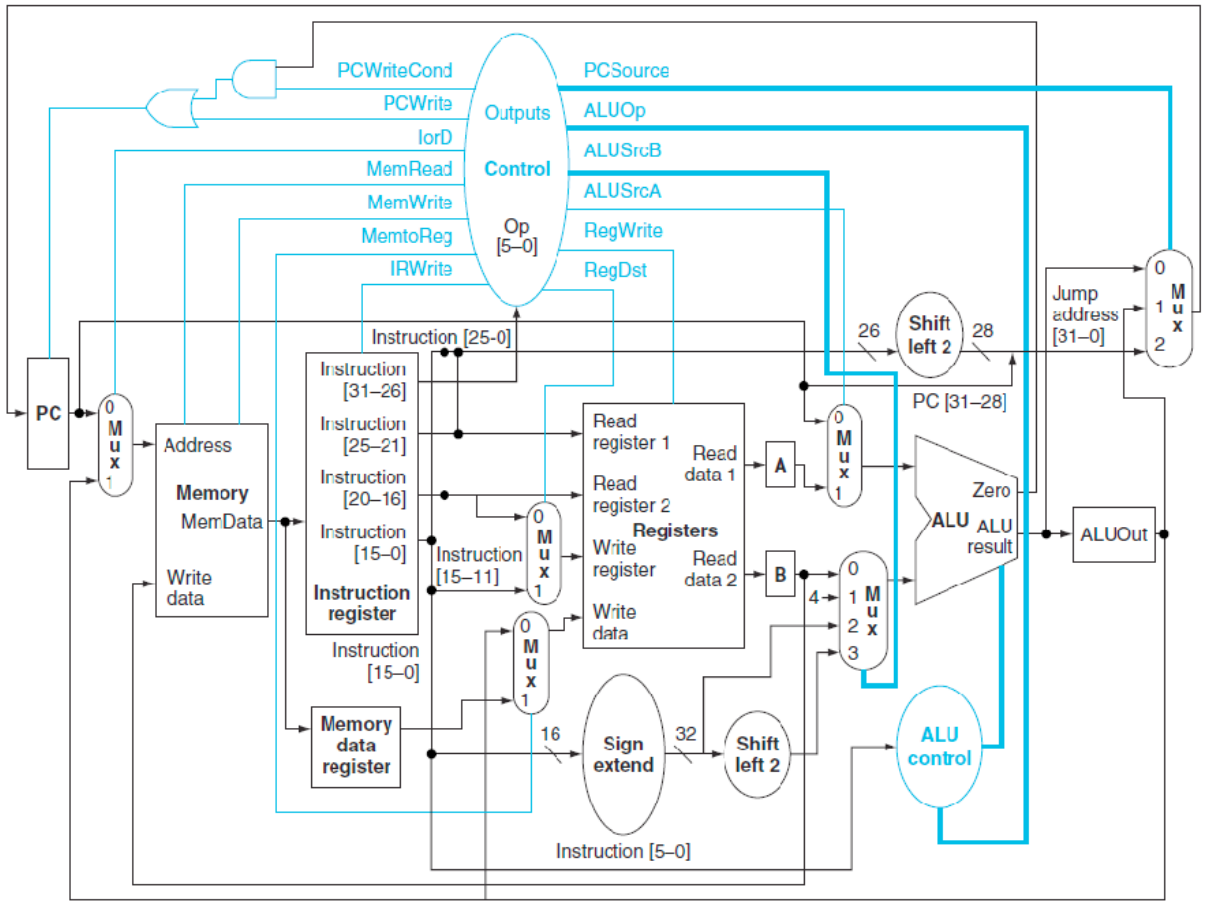
- **One-hot is a group of bits among which the legal combinations of values are only those with a single high (1) bit and all the others low (0)**
- **Advantages**
  - Easy to design and modify
    - Takes advantage of an FPGA's abundant flip-flops (FFs)
    - Determining the state at a low and constant cost of accessing one FF, no decoder
    - Changing the state has the constant cost of accessing two FFs
  - Easy to detect illegal states
- **Disadvantages**
  - Requires more flip-flops than other encodings
  - Many of the states are illegal
- **Using a one-hot implementation typically allows a state machine to run at a faster clock rate than any other encoding of that state machine**

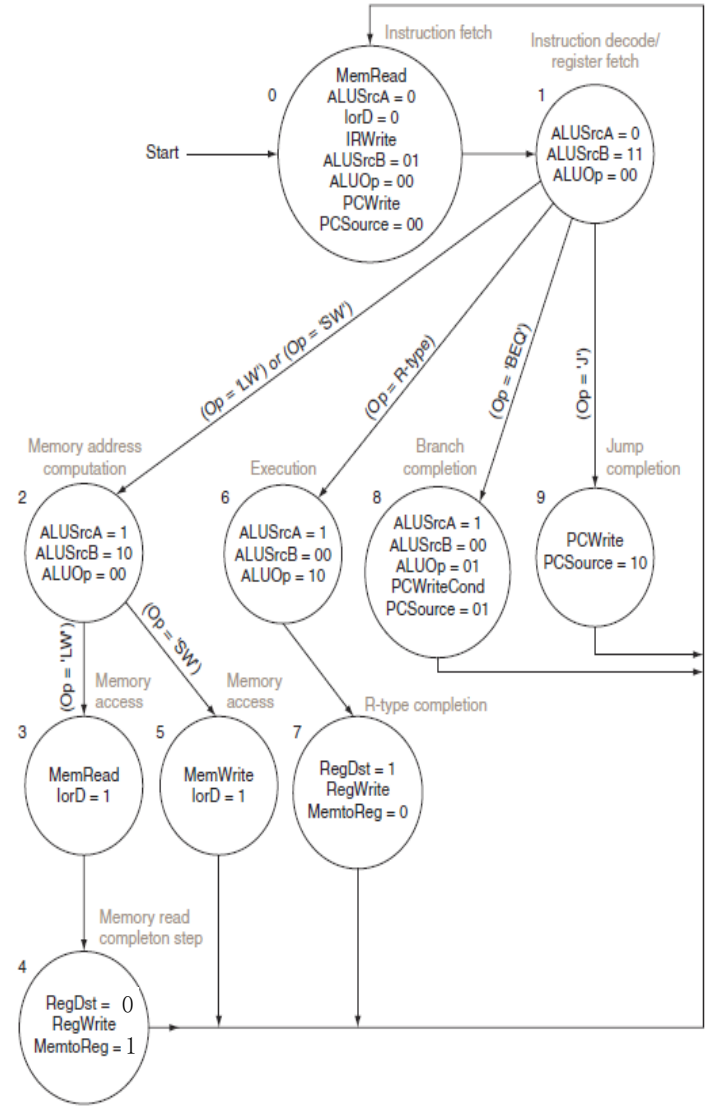
回顾：单周期处理器及其特点



- 每条指令都在一个时钟周期内完成
- 时钟周期为最长的Load指令所花的时间
- 无临时寄存器存放指令执行的中间结果
- 数据通路上的功能部件不能重复使用
- 控制信号在一条指令的执行过程中不变，因此控制器设计简单

回顾：多周期处理器DP&Ctrl





Control Truth Table

Output	Current states	Op
PCWrite	state0 + state9	
PCWriteCond	state8	
IorD	state3 + state5	
MemRead	state0 + state3	
MemWrite	state5	
IRWrite	state0	
MemtoReg	state4	
PCSource1	state9	
PCSource0	state8	
ALUOp1	state6	
ALUOp0	state8	
ALUSrcB1	state1 +state2	
ALUSrcB0	state0 + state1	
ALUSrcA	state2 + state6 + state8	
RegWrite	state4 + state7	
RegDst	state7	
NextState0	state4 + state5 + state7 + state8 + state9	
NextState1	state0	
NextState2	state1	(Op = 'lw') + (Op = 'sw')
NextState3	state2	(Op = 'lw')
NextState4	state3	
NextState5	state2	(Op = 'sw')
NextState6	state1	(Op = 'R-type')
NextState7	state6	
NextState8	state1	(Op = 'beq')
NextState9	state1	(Op = 'jmp')

Key Points of Multi-Cycle CPU

- Performance gain achieved from variable-length instructions
- ET = IC * CPI * cycle time
- Required very few new state elements
- More, and more complex, control signals
- Control requires FSM
- 单条指令的执行时间：单周期CPU vs. 多周期CPU 哪个长？

单周期CPU vs 多周期CPU

- 成本比较：
  - 单周期下功能部件不能重复使用；而多周期下可重复使用，比单周期省
  - 单周期指令执行结果直接保存在PC、Regfile和Memory；而多周期下需加一些临时寄存器保存中间结果，比单周期费
- 性能比较：
  - 单周期CPU的CPI为1，但时钟周期为最长的load指令执行时间
  - 多周期CPU的CPI是多少？时钟周期多长？

    假定程序中22%为Load，11%为Store，49%为R-Type，16%为Branch，2%为Jump。每个状态需要一个时钟周期，CPI为多少?

    分析如下：每种指令所需的时钟周期数为：

    Load：5；Store：4；R-Type：4；Branch：3；Jump：3

    CPI计算如下：

    CPI=CPU时钟周期数 / 指令数 = Σ（指令数$_i$ x CPI$_i$）/ 指令数

    = Σ（指令数$_i$ / 指令数）x CPI$_i$

    CPI = 0.22x5+0.11x4+0.49x4+0.16x3+0.02x3 = 4.04

    假设单周期时钟宽度为1，则多周期时钟周期约为单周期的1/5，所以，多周期的总体时间：4.04x1/5=0.808；而单周期总体时间为：1x1=1

    由此看出：多周期比单周期效率高!

Quiz

- How many cycles will it take to execute this code? **21**

```
5 lw $t2, 0($t3)
5 lw $t3, 4($t3)
3 beq $t2, $t3, Label  #assume not taken
4 add $t5, $t2, $t3
4 sw $t5, 8($t3)
Label:    ...
```

- What is going on during the 8th cycle of execution? **lw**
- In what cycle does the actual addition of $t2 and $t3 take place? **16th**

- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?

.2*(5) + .1*(4) + .5*(4) + .2*(3) = 4

总结

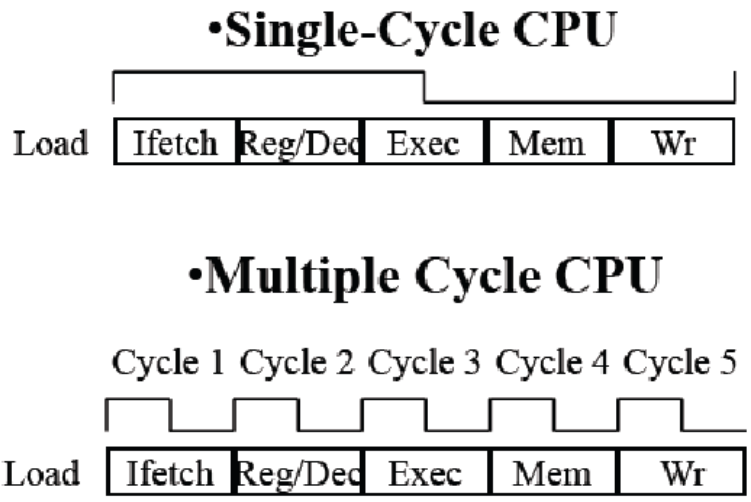
- 单周期处理器的设计
  - 每条指令都在一个时钟周期内完成
  - 时钟周期以最长的Load指令所花时间为准
  - 无需加临时寄存器存放指令执行的中间结果
  - 同一个功能部件不能重复使用
  - 控制信号在整个指令执行过程中不变，所以控制器设计简单，只要写出指令和控制信号之间的真值表，就可以设计出控制器
- 多周期处理器的设计
  - 每条指令分成多个阶段，每个阶段在一个时钟内完成
  - 不同指令包含的时钟个数不同
  - 阶段的划分要均衡，每个阶段只能完成一个独立、简单的功能，如：
    - 一次ALU操作
    - 一次存储器访问
    - 一次寄存器存取
  - 需加临时寄存器存放指令执行的中间结果
  - 同一个功能部件能在不同的时钟中被重复使用
  - 可用有限状态机来表示指令执行流程，并以此设计控制器

总结

- 控制单元实现方式
  - 有限状态机描述方式
    - 每个时钟周期包含的控制信号的值的组合看成一个状态，每来一个时钟，控制信号会有一组新的取值，也就是一个新的状态
    - 所有指令的执行过程可用一个有限状态转换图来描述
    - 用一个组合逻辑电路来生成控制信号，用一个状态寄存器实现状态之间的转换
    - 实现的控制器称为硬布线控制器
  - 微程序描述方式 （详见唐朔飞教材第10.2章节）
    - 每个时钟周期所包含的控制信号的值的组合看成是一个0/1序列，每个控制信号对应一个微命令，控制信号取不同的值，就发出不同的微命令
    - 若干微命令组合成一个微指令，每条指令所包含的动作就由若干条微指令来完成，每来一个时钟，执行一条微指令
    - 每条指令对应一个微程序（需多个时钟完成），执行时，先找到对应的第一条微指令，然后按照特定的顺序取出后续的微指令执行
    - 实现的控制器称为微程序控制器

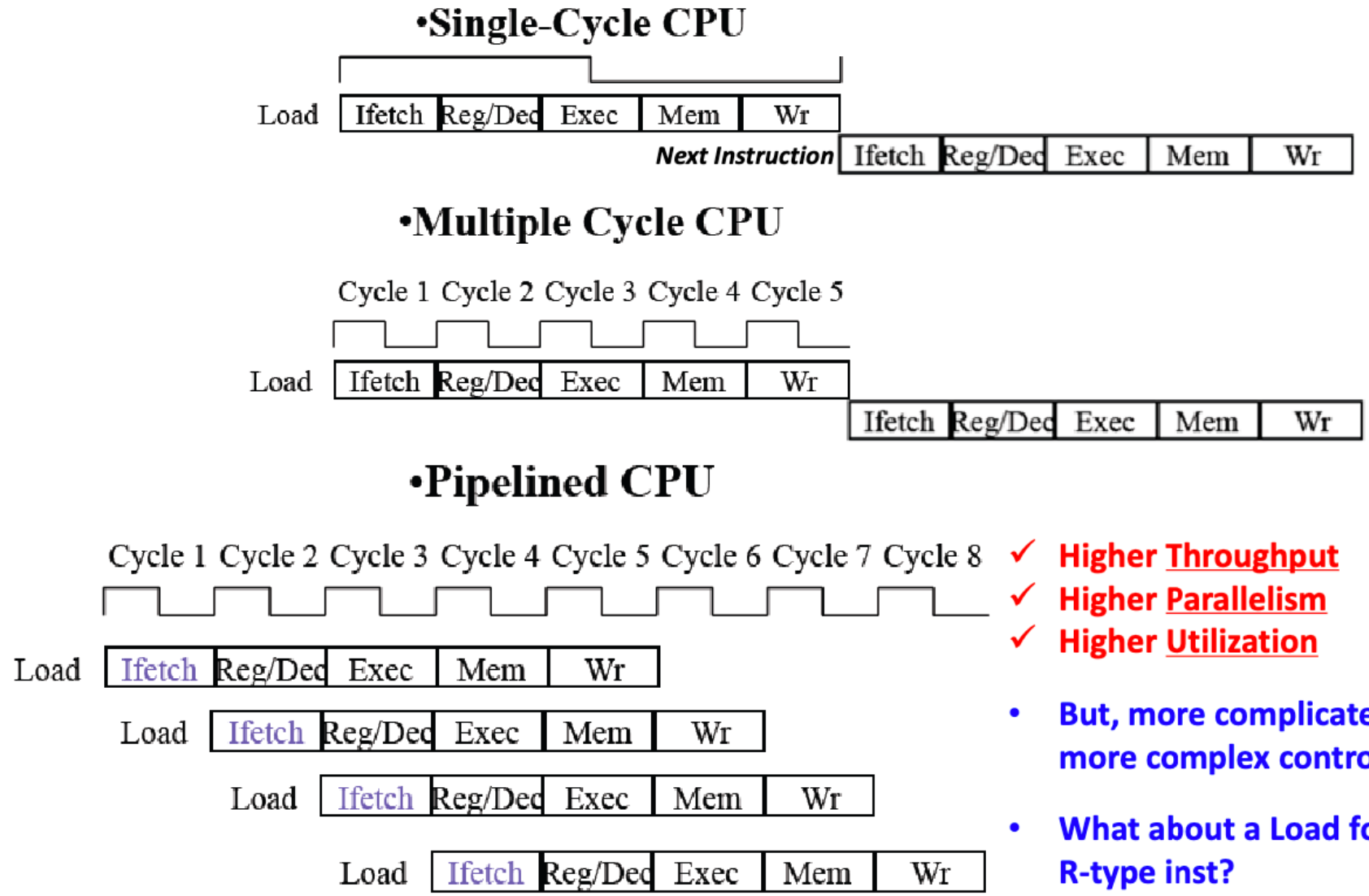
从指令执行看处理器内部资源利用率

- 单周期或多周期CPU中，Load指令执行时间最长



- 单周期CPU若执行其他短指令，在时钟周期后半阶段整体处于闲置
- 多周期CPU内部的所有部件没有在每一拍都处于忙碌状态
- 能否尽可能多地让CPU内部各个部件都持续地、充分地使用

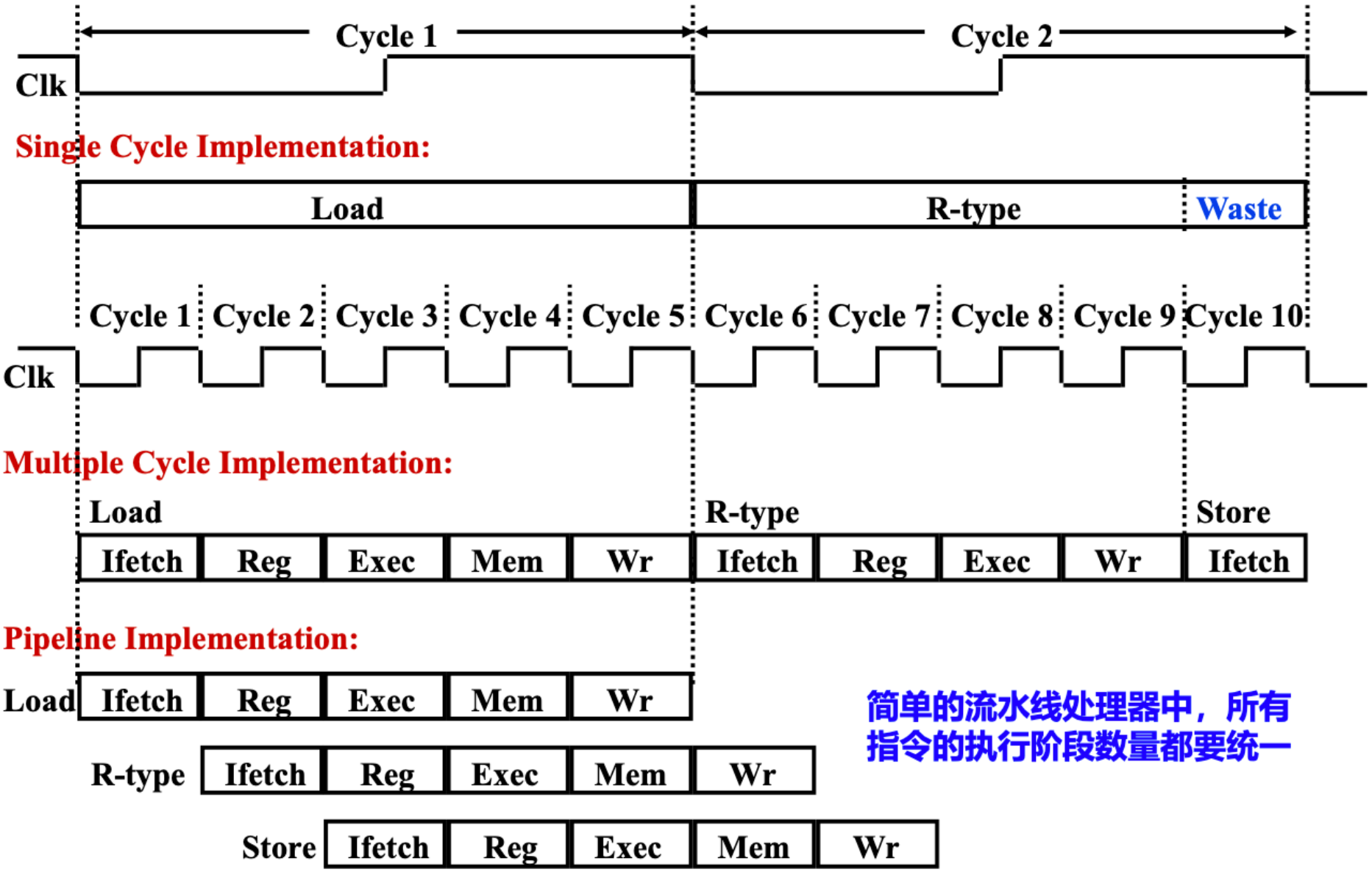
流水线处理器（Pipelined CPU）



- ✓ Higher Throughput
- ✓ Higher Parallelism
- ✓ Higher Utilization

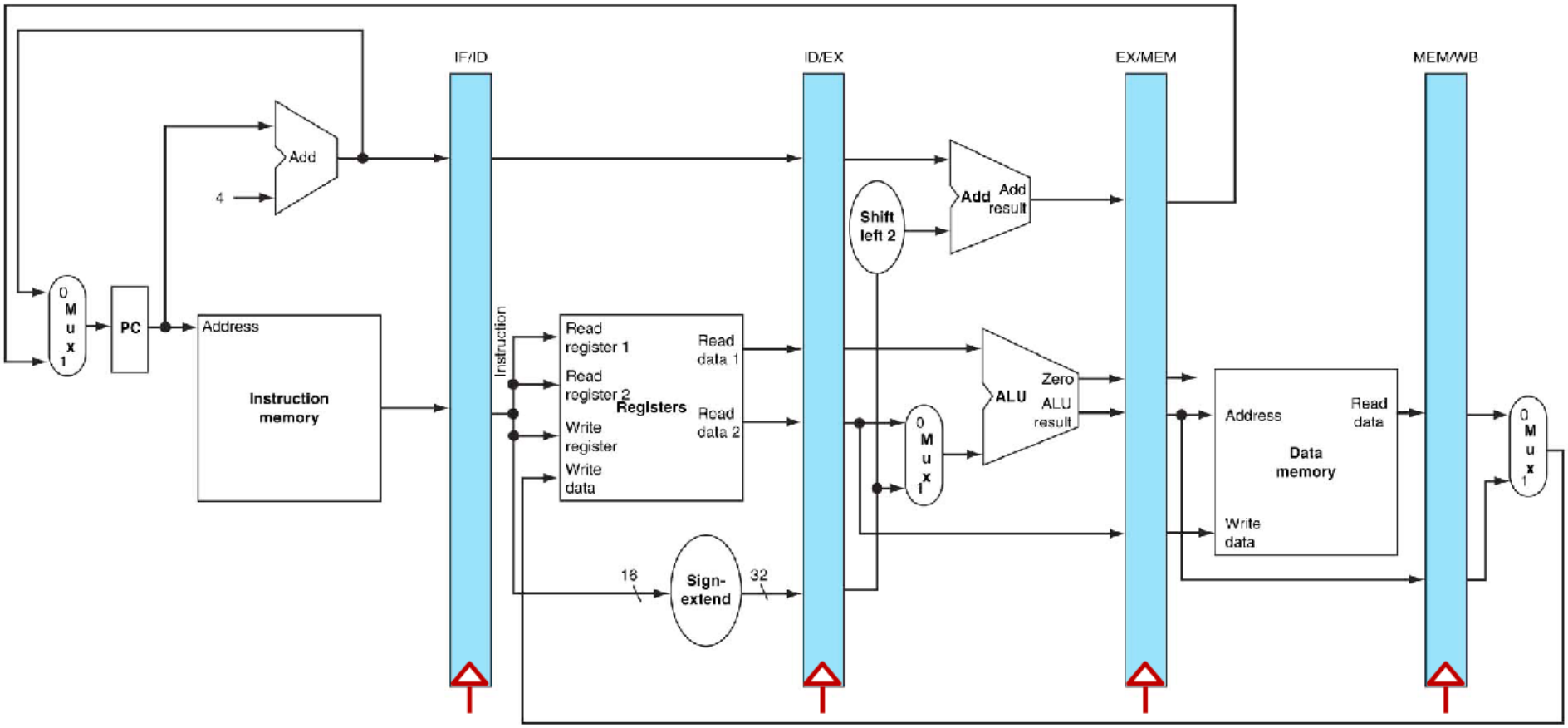
- But, more complicated datapath, more complex control
- What about a Load followed by a R-type inst?

单周期 vs 多周期 vs 流水线



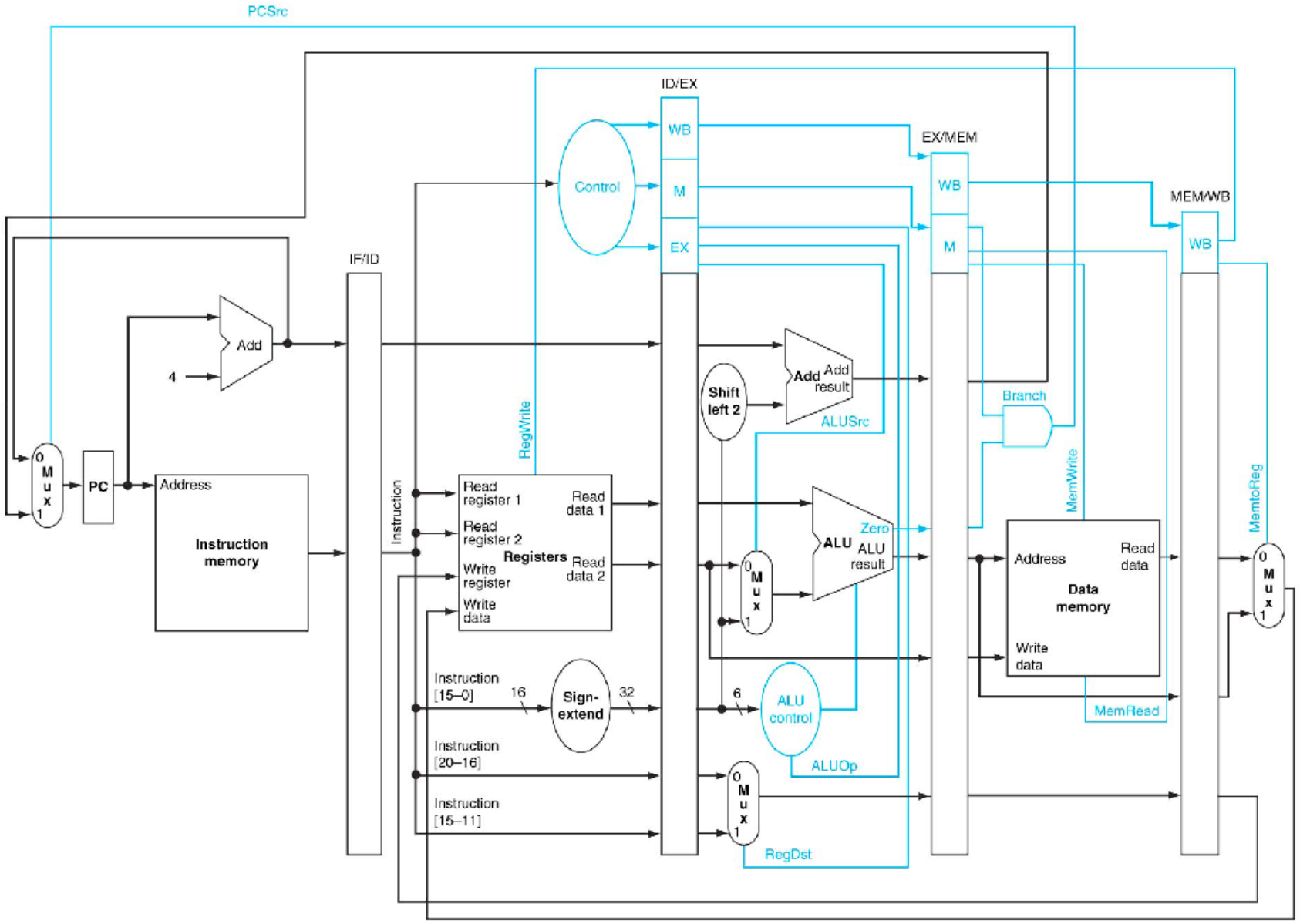
简单的流水线处理器中，所有指令的执行阶段数量都要统一

A Quick Look at the Pipelined DP



- Registers between stages to hold information produced in previous cycle
- How many bits do these registers contain for each stage ?

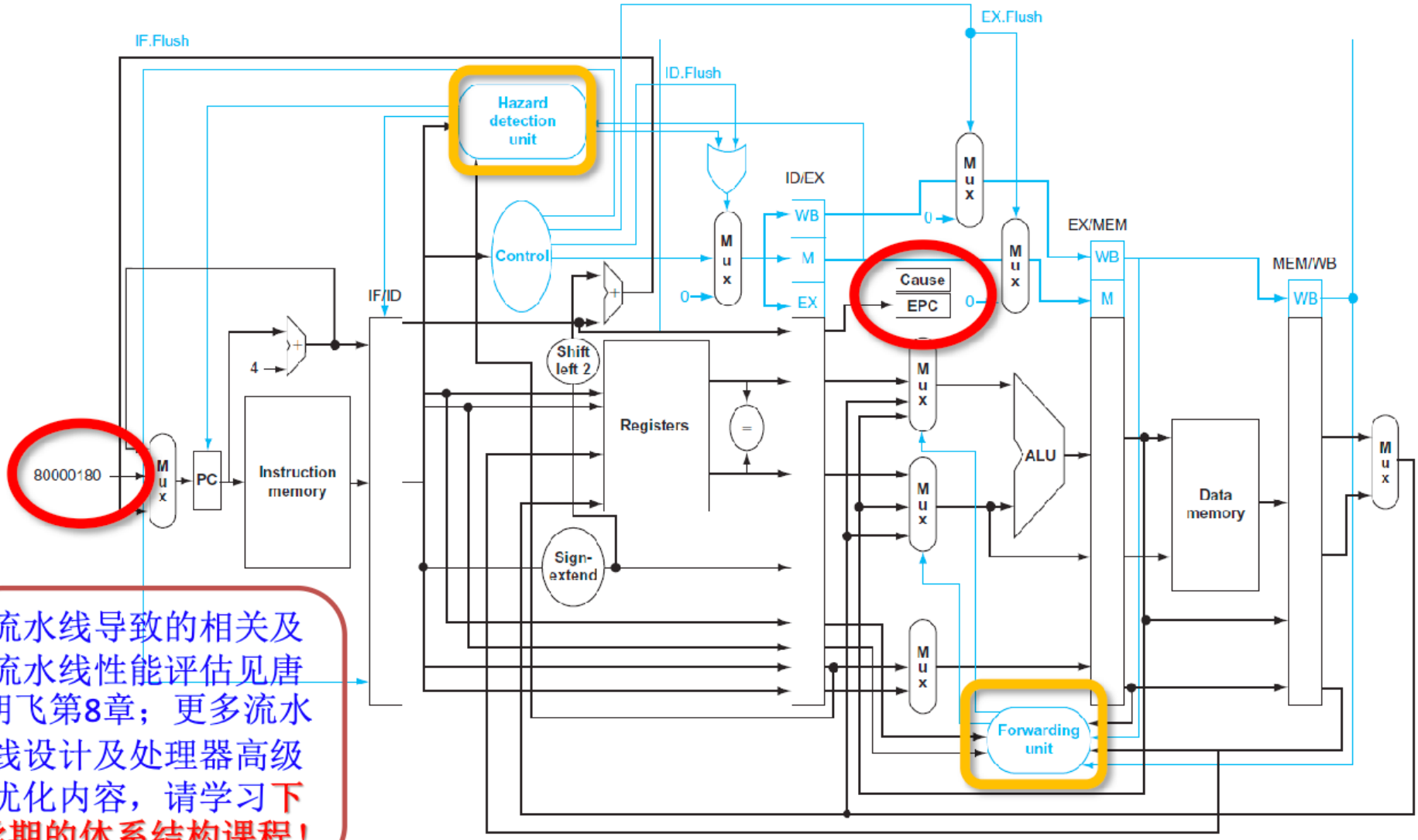
Pipelined DP with control signals



Design Principles of Pipelining

- **All instructions that share a pipeline should have the same stages in the same order**
  - E.g., add instruction does nothing during MEM stage
  - E.g., sw instruction does nothing during WB stage
- **There is no functional block reuse**
- **All intermediate values must be registered each cycle**
- **Pipeline control signals generated once, but follow instruction through the pipeline**
  - Pipelining uses combinational logic to generate (and registers to propagate) control signals
- **Pipelining creates potential hazards (翻译为冒险或相关，唐朔飞教材第8章)**

Pipelined datapath with controls to handle hazards & exceptions



流水线导致的相关及流水线性能评估见唐朔飞第8章；更多流水线设计及处理器高级优化内容，请学习下学期的体系结构课程！

异常（Exception）和中断（Interrupt）

- 程序执行过程中CPU会遇到一些特殊情况，使正在执行的程序被“打断”
  - 此时，CPU中止原来正在执行的程序，转到处理异常情况或特殊事件的程序去执行，执行后再返回到原被中止的程序处（断点）继续执行
- 程序执行被 “打断” 的事件有两类
  - 内部“异常”：在CPU内部发生的意外事件或特殊事件

    按发生原因分为硬故障中断和程序性中断两类

    硬故障中断：如电源掉电、硬件线路故障等

    程序性中断：执行某条指令时发生的“例外”，如算术溢出、缺页、越界、越权、非法指令、除数为0、堆栈溢出、访问超时、断点设置、单步、系统调用等
  - 外部“中断”：在CPU外部发生的特殊事件

    通过“中断请求”信号向CPU请求处理。如实时钟、控制台、打印机缺纸、外设准备好、采样计时到、DMA传输结束等
- 通常，异常指控制流中任何意外改变，中断特指来自外部事件
- 中断处理(Interrupt handling，或称中断服务程序)用来处理异常和中断
- CPU周而复始执行指令
  - 执行指令过程中，若发现异常，则立即转异常处理（或作出检测标记，记录原因，并到指令最后阶段再处理）
  - 每个指令结束，查询有没有中断请求，有则响应中断

作业

- 理论课无作业！
- 请大家开展：计算技术调研作业&报告
- 预习唐朔飞教材第八章

调研作业：计算技术调研报告

- 结合自己的兴趣和方向，利用学到的计算机组成原理知识，写一份**关于计算领域内某个技术或新趋势的调研报告，并自愿报名进行课堂演讲**

- 参考技术方向（包括但不限于）：
  - 处理器架构、ASIC设计及验证、SoC、FPGA、计算机系统设计、硬件电路设计、嵌入式开发、OS及系统软件开发、应用编程、性能优化、高性能计算/超级计算机、大数据、云计算、数据中心、物联网IoT、存储、网络、安全、人工智能/智能信息、机器学习/神经网络、无线/有线通信、数据分析、生物计算、脑机接口

- 或者继续深入调研课堂上补充介绍过的新技术或相关历史

- 聚焦，不要泛泛而谈

调研作业：计算技术调研报告

- 形式要求：提交中文或英文文档(建议，但不要大段的直接拷贝)，并自愿做演讲报告(不超过10分钟)
  - 内容：背景、特征、工作原理、结构等，请务必标注引用的参考文献
- 可独立或分组完成
  - 一组不超过3人，报告中写清作者列表及分工情况
- 文档(Doc)：必做
- 演讲(PPT)：选做并加分
  - 可以在演讲时进行实物、视频、图片等动态展示
  - 注意：PPT屏幕尺寸比例需为4:3
  - 助教统计报名信息
- PPT/Doc文档截至时间（暂定）：
  - 最迟本学期最后一天23:55之前上传“完整版本终稿”至SEP“课堂作业”
  - HARD Deadline！错过此时间节点将影响课程的最终成绩发布
  - 多人合作时，仅需第一作者在SEP的“作业”中提交
- **评分标准：有趣、有人文关怀、信息密度高、AI含量低**

如何做好调研报告与演讲？

- 我们课程的调研报告可以偏学术研究方向，也可以偏工业界热点
- 为何要做：
  - 老石谈芯：这十个思维方式如何改变我的人生
    - https://www.bilibili.com/video/BV1A5411E7zG?spm_id_from=333.788.recommend_more_video.3&vd_source=97f81424e6d8a747839d4cdd338c8419
- 如何撰写技术文档
  - Google Technical Writing Course, https://developers.google.com/tech-writing
- 如何做幻灯片PPT、如何进行演讲
  - “17 Simple Strategies To Survive Your PhD”，No. 9-Give Presentations
  - 《求生之路：博士生涯的17条简单生存法则》，第9条-发表演讲
    - https://www.jiqizhixin.com/articles/2018-06-19-5
- 白岩松西湖大学演讲：科研人要学会讲故事
  - https://mp.weixin.qq.com/s/xjcajIeZG_Tm1-fmb05BDg
- 博士学位真的那么重要吗？上交大博士亲述科研心路
  - https://www.zhihu.com/question/366627317/answer/1151278214
- 华盛顿大学2006年秋季《计算历史》课学生们的课程报告，供大家参考
  - http://courses.cs.washington.edu/courses/csep590/06au/course-projects.html

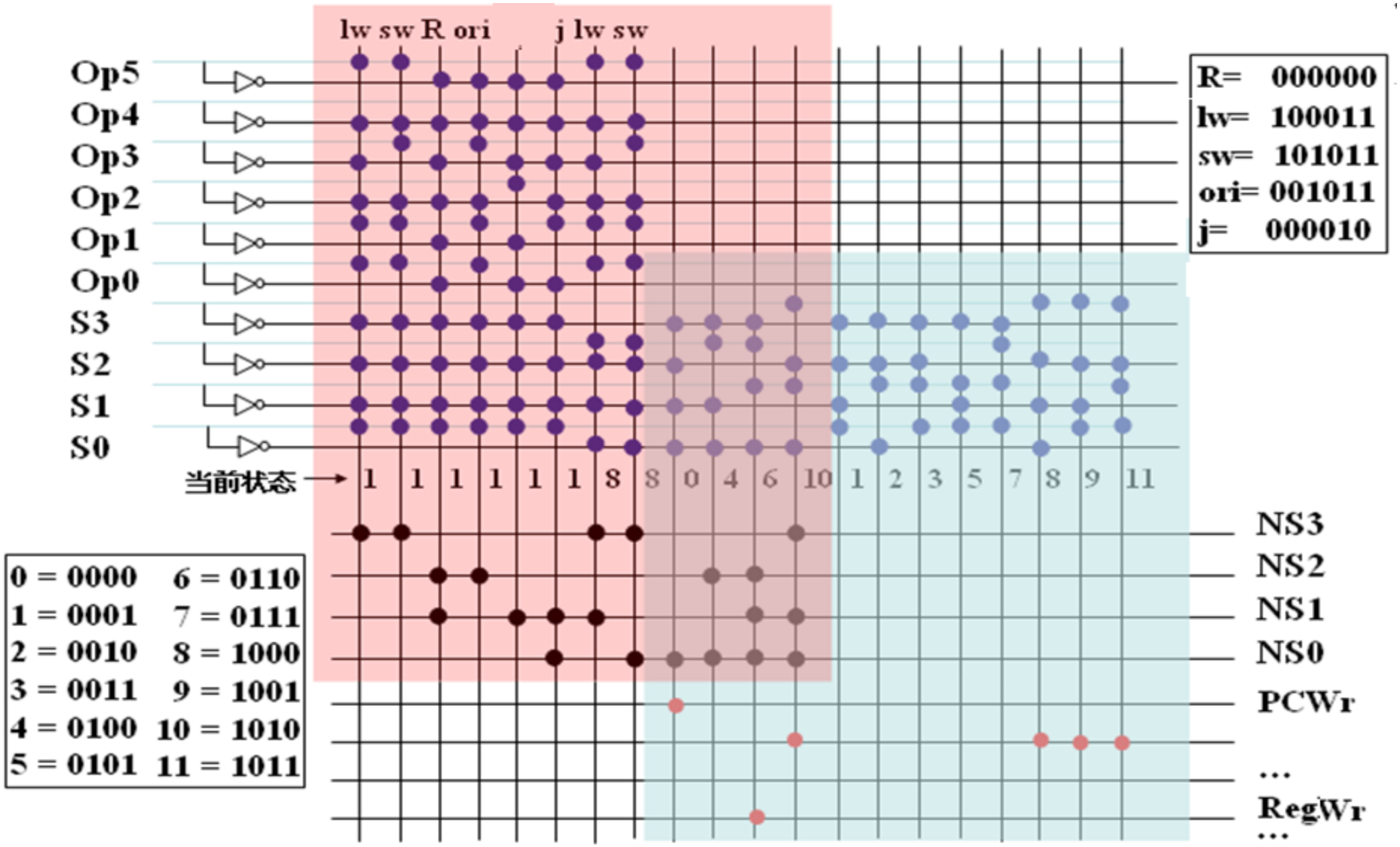
往届的调研汇报主题

冬奥会与计算机
会整活的StyleGAN
人工智能与芯片技术
虚拟化技术
“芯片胶水”——Chiplet技术
AI Image Matting
分布式深度学习系统
三维场景重建
复杂神经系统中的人脑建模
雷电技术简介
图像识别技术——卷积神经网络
图像超分辨率技术
树莓派计算机的发展历史
容器化技术
脑机接口
基于点云检测的3D目标识别在自
视频编码
网络附属存储NAS

量化投资
Chisel背景、特点、应用、学习
Computer Vision (CNN for Visual Recognition) History, Theory and Hardware-Implementation
Meltdown & Spectre处理器漏洞报
AI芯片
/[A-Z]PU/ Collection and Analysis
云游戏
硬盘挖矿
“香山”RISC-V处理器
深度学习
类脑计算
DNA计算
游戏引擎
碰撞检测算法
OLED技术
LoongArch
Chiplet
人机交互HCI
QPU

嵌入式系统简介(上/下)——历史、现状与未来展望
从VGA到DVI到HDMI再到DP——视频接口的发展历程（上/下）
Generative Adversarial Networks (GANs) & Model Compression
NVIDIA GPU图灵架构
真伪随机数发生器的软硬件实现
量子计算与量子传感
声卡和计算机音乐简介
精灵宝可梦中的计算机科学
USB的发展历史
分布式系统简介
物联网简介
神经网络简介
人机交互
CV的历史发展趋势与硬件需求
EDA的前世今生
显卡历史简介及深度学习方向
NAND闪存芯片颗粒介绍
现代高级cache技术简介
量子计算机简介
图像渲染技术简介

示意：用PLA电路实现控制单元（硬布线方式）



实现的有限状态机称为“摩尔机（Moore Machine）”

左上角：由操作码和当前状态确定下一状态的电路

右下角：由当前状态确定控制信号的电路