



清华大学  
Tsinghua University

# 第二单元 第三讲

## 单周期CPU控制器设计

刘卫东

计算机科学与技术系

# 本讲提要



## ❖ 指令执行方式和步骤

## ❖ 单周期CPU设计

- ❖ 指令集：7条典型指令

- ❖ 数据通路

- ❖ 控制器

## ❖ Project 3

- ❖ 存储器及串口访问

## 完成指令功能的主要部件

- ALU

## 指令功能

- 数据运算：算术、逻辑、移位

- 数据移动：

- 流程控制：转移、调用/返回、中断

- 其他：

## 如何实现？

- Datapath：实现数据的移动和运算

- 控制器：指挥数据的移动和运算

# 每条指令的执行过程



## ❖ 第一步

❖ 取指令 (IF)

## ❖ 第二步

❖ 指令译码 (ID)

## ❖ 第三步

❖ 执行指令 (EXE)

## ❖ 第四步

❖ 访问存储器 (MEM)

## ❖ 第五步

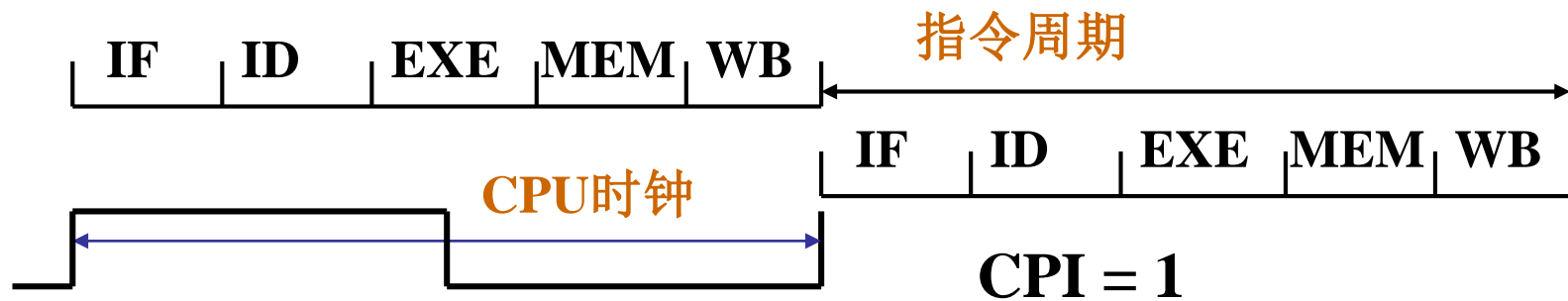
❖ 写回寄存器 (WB)

# 指令执行步骤-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一条指令。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**。



# 实现的指令集



## ❖ 选取MIPS指令中7条典型指令组成的子集

- ❖ 访存指令：LW、SW

- ❖ 算逻运算指令：ADDU、SUBU、ORI

- ❖ 转移指令：BEQ、J

## ❖ 解决三个主要问题

- ❖ 数据通路设计

- ❖ 控制信号设计

- ❖ 执行时序设计

## ❖ 其他指令的实现原理可以从中体现

# 设计思路



## ❖ 指令的执行

- ❖ 显然要设计一个时序逻辑电路
- ❖ 一条指令用一个CPU周期完成

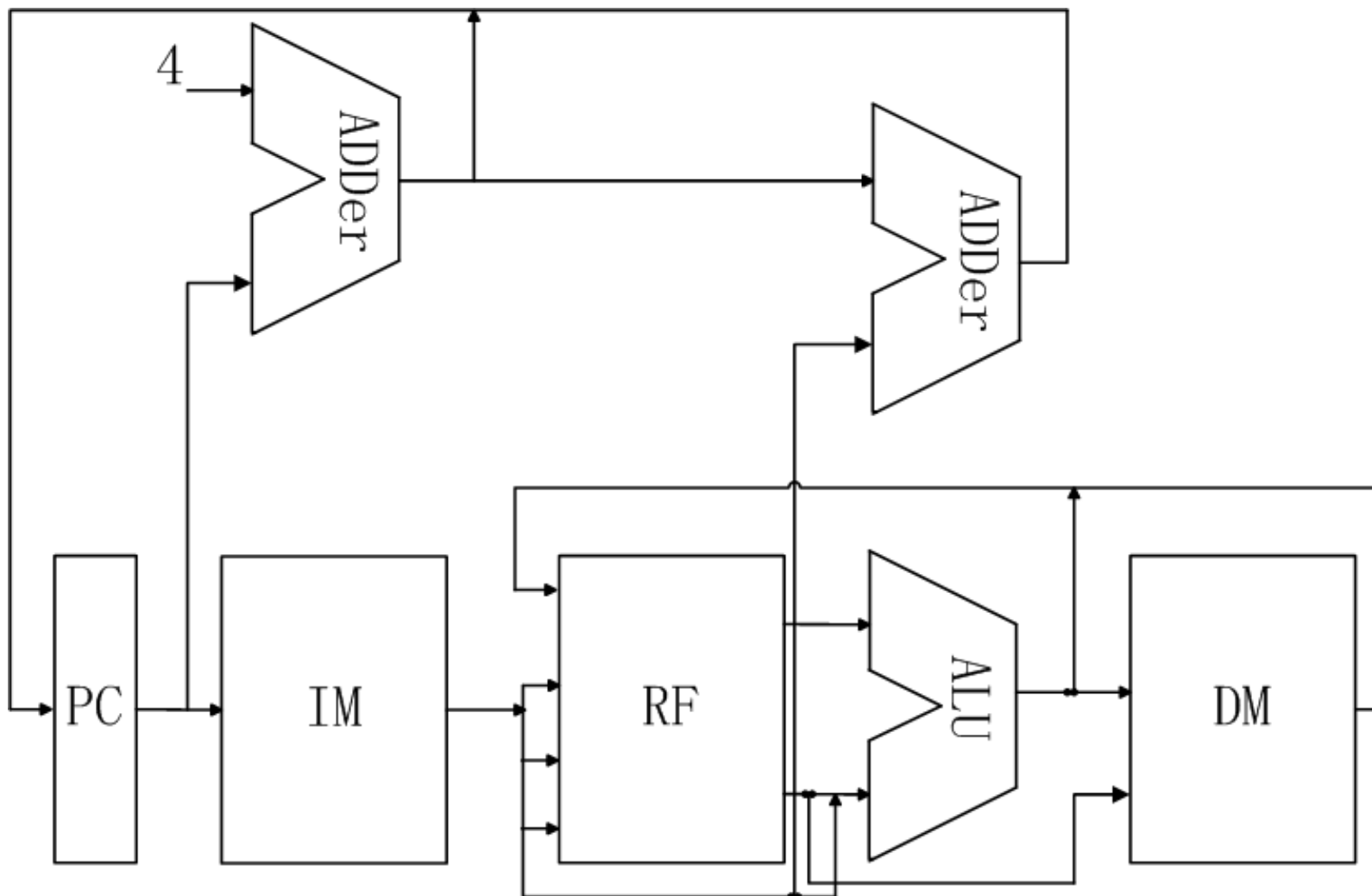
## ❖ 执行步骤的实现

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

## ❖ 需要保存的值

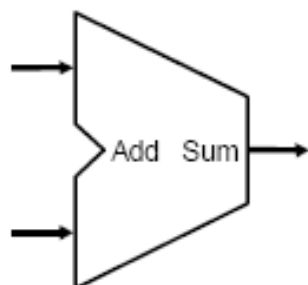
- ❖ PC、寄存器组、存储器

# 最初步的Datapath

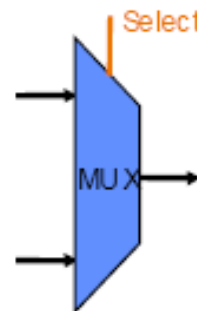




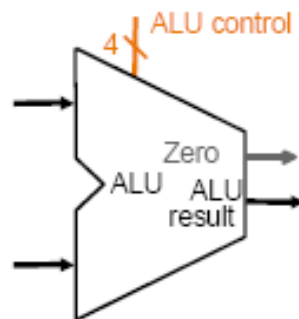
# 使用的组合逻辑部件



**Adder**



**MUX**



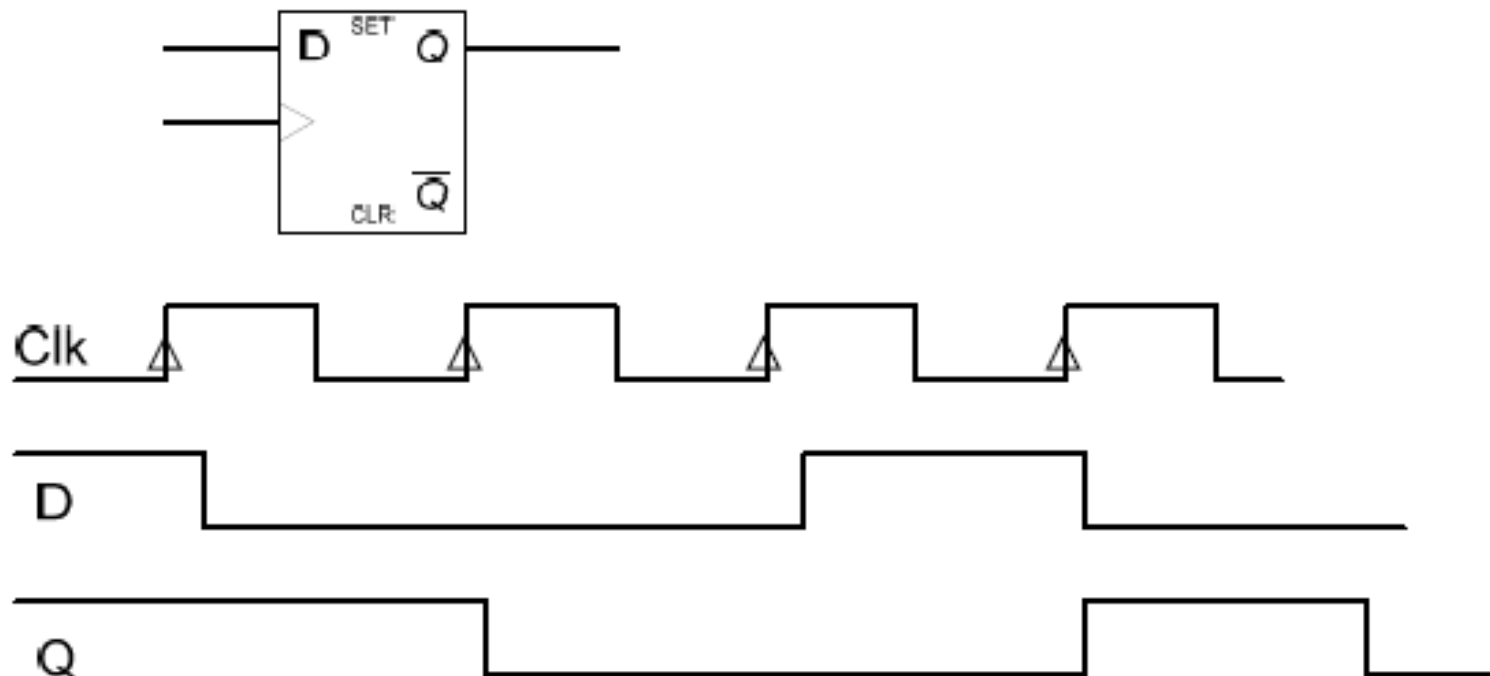
**ALU**

# D触发器

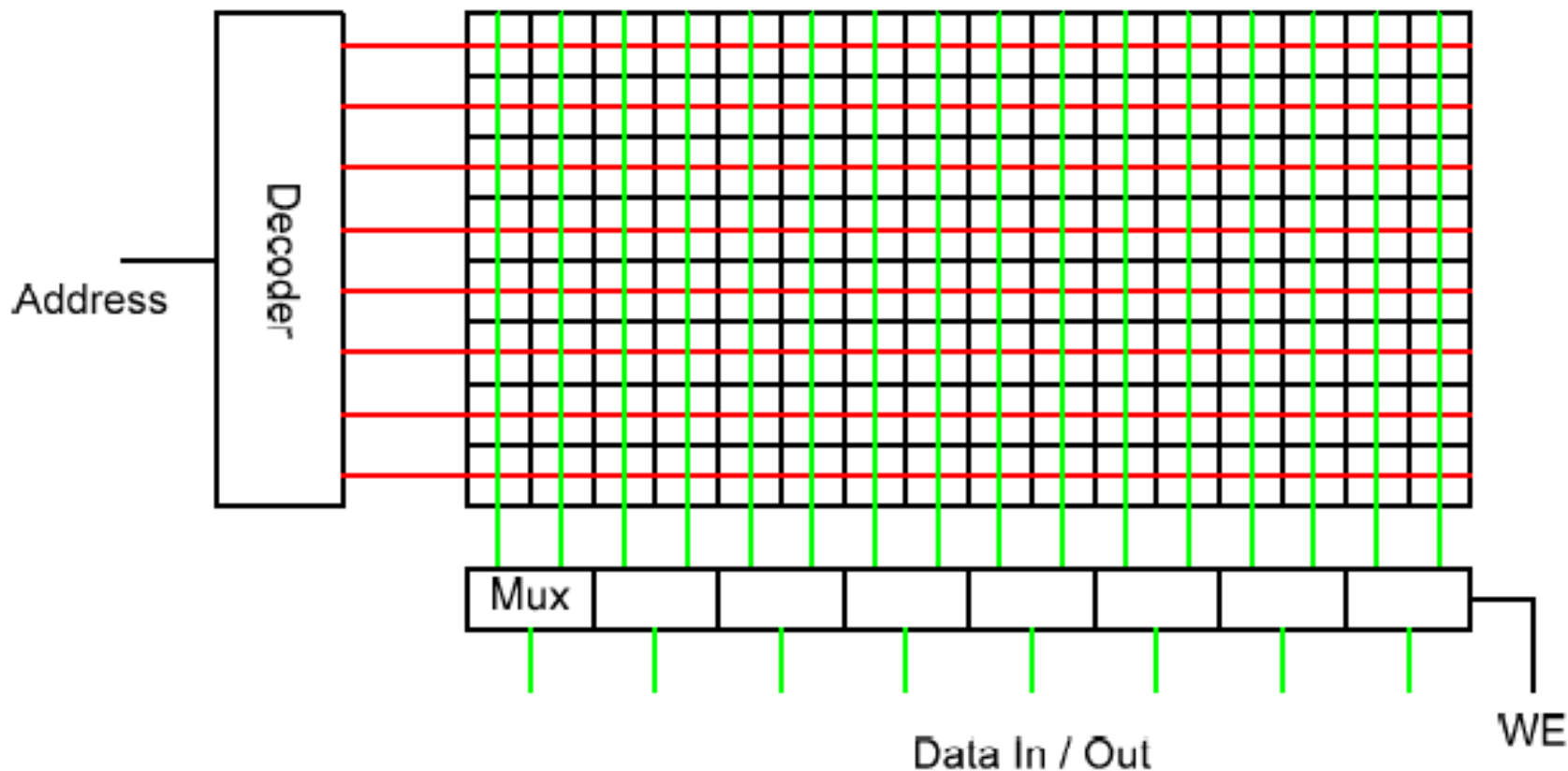


在时钟上升沿写入输入数据

一直保持到下一个上升沿



# 存储器



# 寄存器组



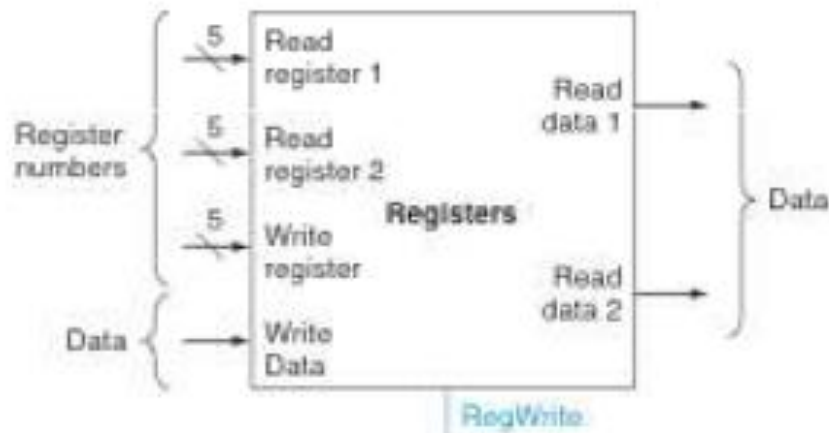
## 接口简单

### 输入

- ◆ 地址
- ◆ 写入数据
- ◆ 写信号

### 输出

- ◆ 读出的数据



3个地址端口，2个用来读，1个用来写

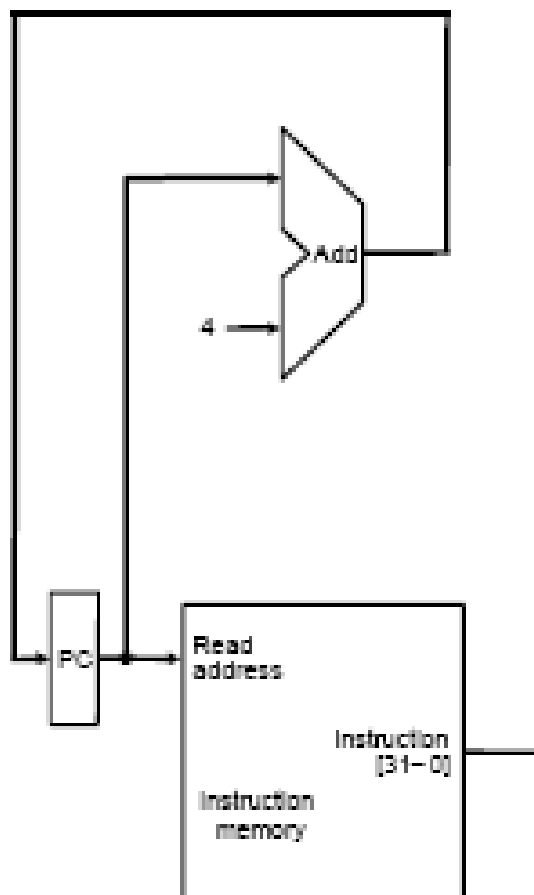
每个时钟周期可以完成3次访问

# 第一步：取指

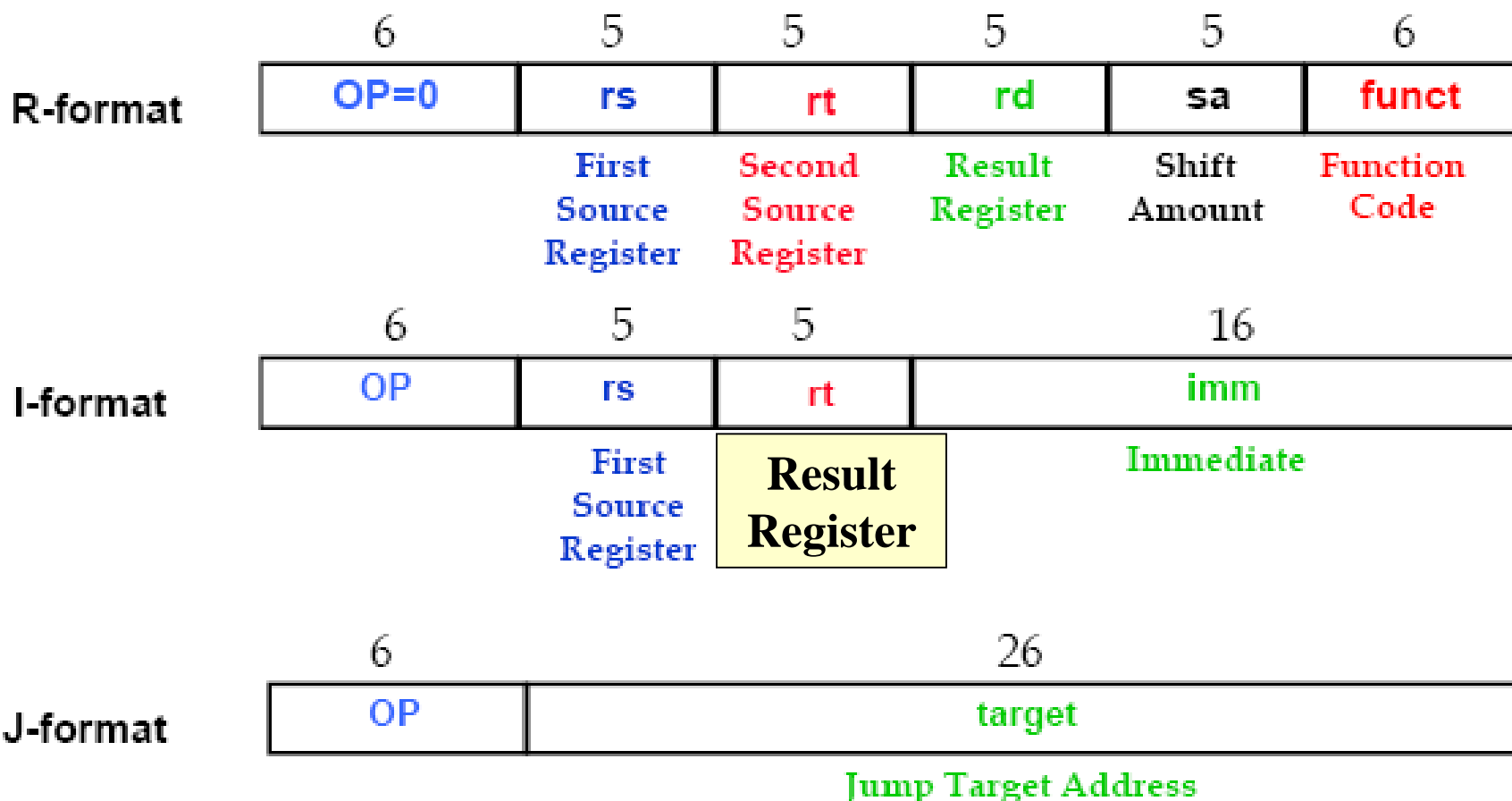


- ❖ 当前指令存储在存储器中，用PC指示指令地址
- ❖  $\text{Instr} = \text{MEM}[\text{PC}]$ 
  - ❖ 从存储器中读出指令
  - ❖ 给地址，一定时间的延迟后，存储器输出当前的指令
- ❖ 修改PC，使其指向下一条指令
  - ❖ 下一条指令的地址是？
- ❖ 取指令需要哪些控制信号？

# 取指的实现



# 取到了些什么？



# MIPS指令的特点



## ❖ 指令长度固定

- ❖ 不需要根据当前指令的类型来确定下一条指令的地址
- ❖ 只需对PC增量即可

## ❖ 寄存器位置基本固定

- ❖ 目的寄存器位置有所变动，但是
- ❖ 源寄存器位置总是固定的
  - ◆ 或者不需要源寄存器
- ❖ 在指令译码完成前就可以取操作数了



# 寄存器间运算指令



## 两条算术指令

Addu rd rs rt

Subu rd rs rt

## 运算

$R[rd] \leftarrow R[rs] + R[rt];$  Add operation

$R[rd] \leftarrow R[rs] - R[rt];$  Sub operation

Bits

6

5

5

5

5

6

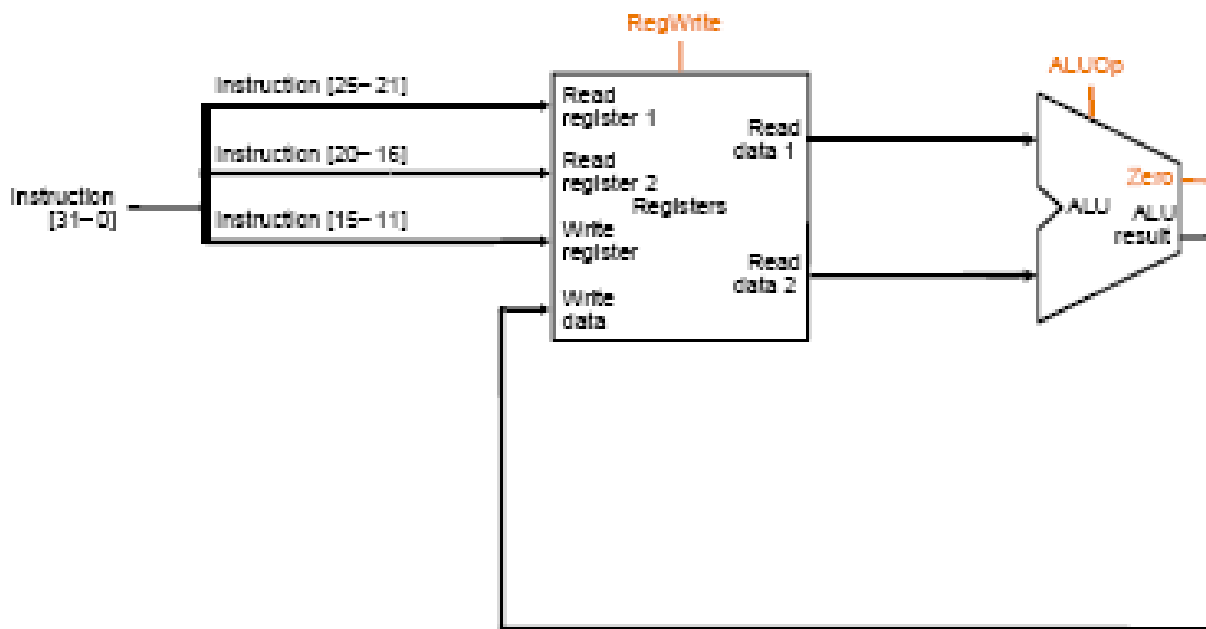
OP=0	rs	rt	rd	sa	funct
------	----	----	----	----	-------

First	Second	Result	Shift	Function
Source	Source	Register	Amount	Code
Register	Register			

# 实现R型指令的数据通路



- ✚  $R[rd] \leftarrow R[rs] \text{ op } R[rt]$
- ✚ 通过指令译码得到
  - ▣ ALU功能码ALUOp
  - ▣ 写寄存器控制信号RegWrite
- ✚ 根据指令格式，从指令字段中得到寄存器地址



# ORI指令



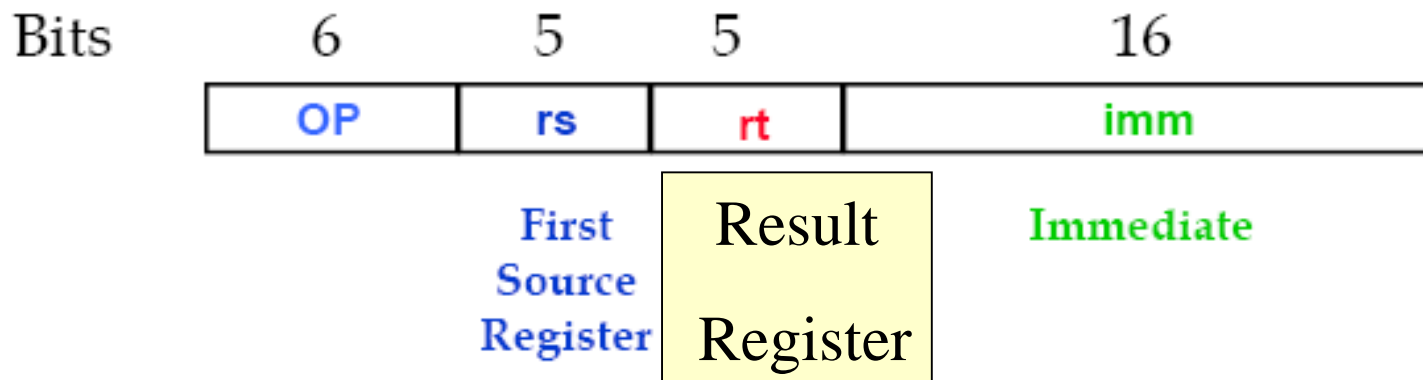
## OR 立即数指令

ORI rt rs imm

$R[rt] \leftarrow R[rs] \text{ or } \text{ZeroExt}(\text{imm})$

需要将Instr[15:0]送到ALU中

目的寄存器字段有变化

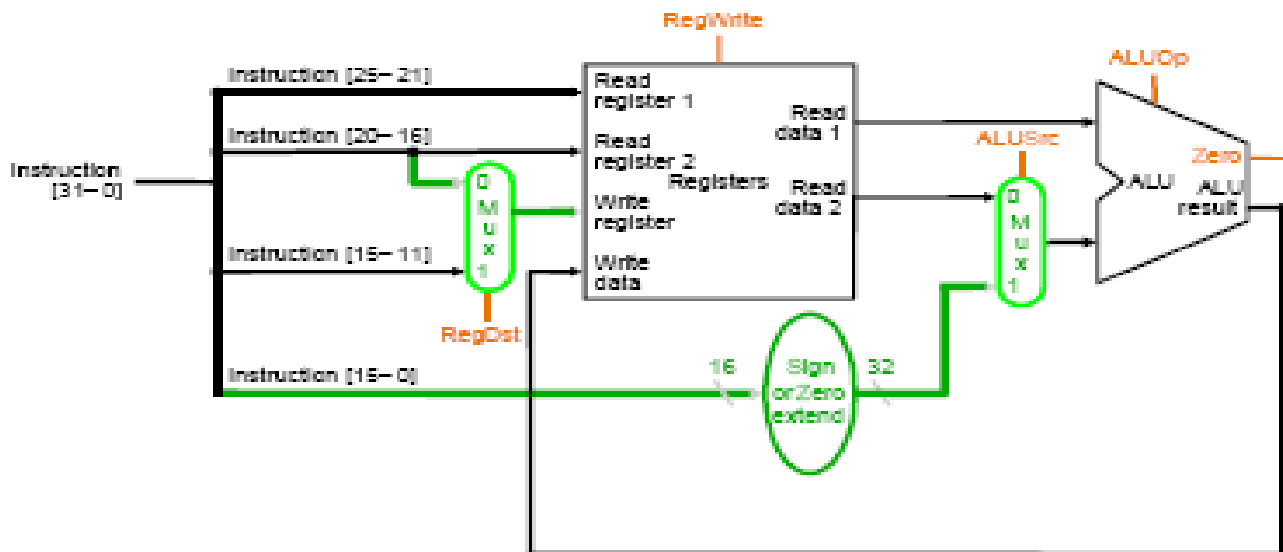


# 立即数型指令的实现



## ✚ 将Datapath扩展以实现立即数型指令

- ❑ 目的寄存器：rt 或 rd
- ❑ 立即数字段要零扩展或者是符号扩展
- ❑ ALUOp：来自指令功能译码
- ❑ ALUSrc：来自指令操作码译码（寄存器/立即数）



# LOAD指令



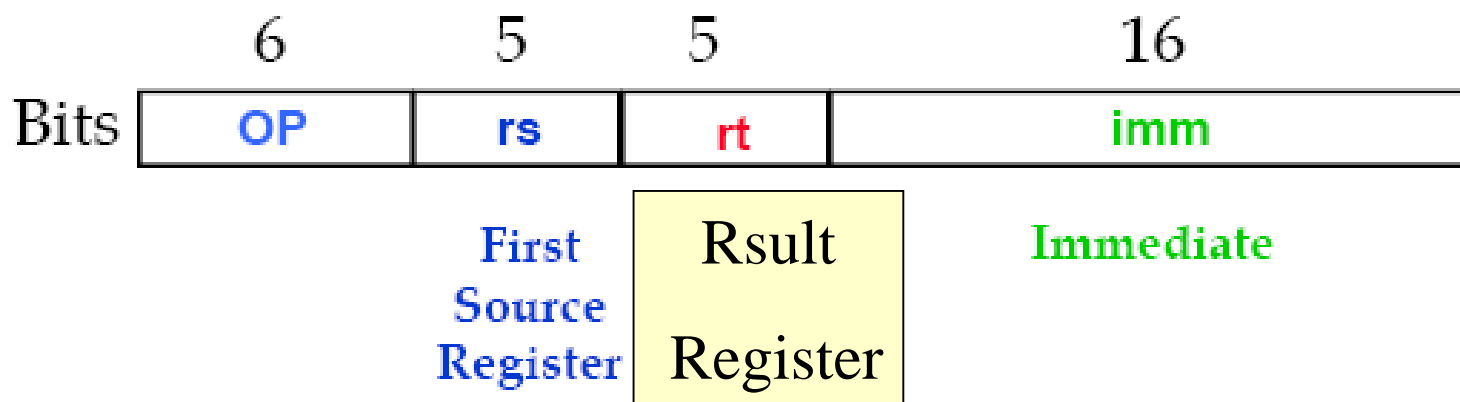
## LOAD 指令

❏ `lw rt, rs, imm`

❏  $\text{Addr} \leftarrow R[\text{rs}] + \text{SignExt}(\text{imm})$       计算地址

❏  $R[\text{rt}] \leftarrow \text{MEM}[\text{Addr}]$       读数据

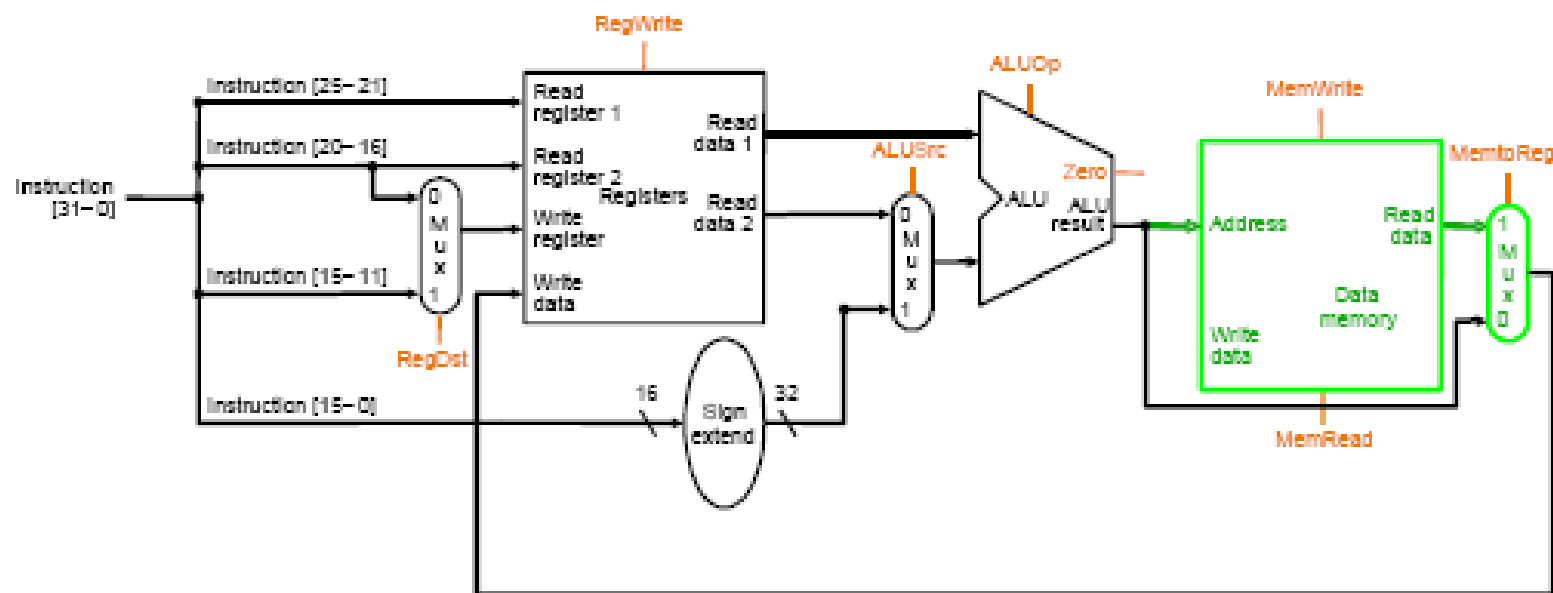
❏ 使用和立即数指令基本相同的Datapath，  
但需要增加数据存储器读操作



# 支持Load指令的Datapath



- 增加符号扩展和零扩展选择
- 增加读数据存储器的通路
- 增加ALU输出结果和数据存储器输出的选通



# STORE指令



## Store 指令

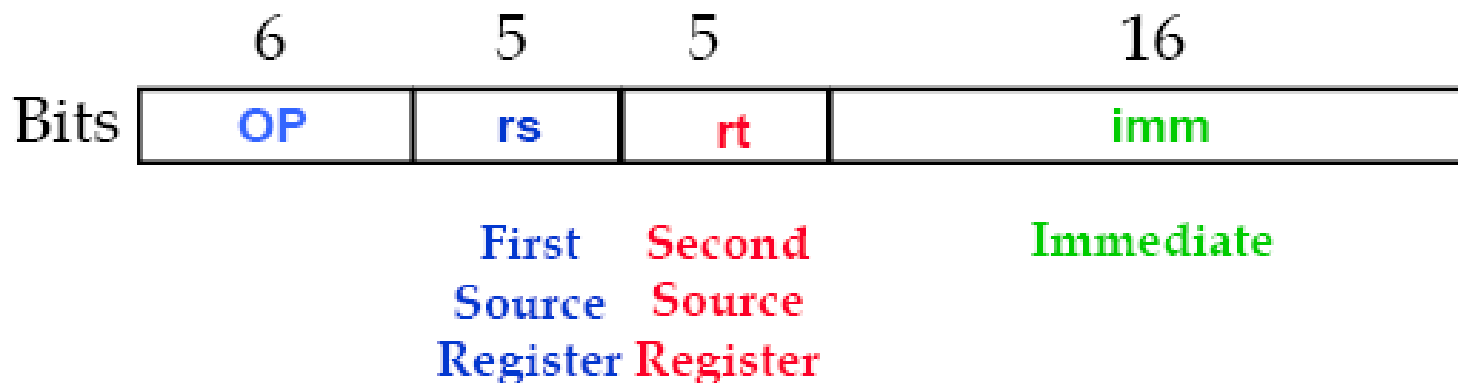
❏  $sw \ rt, rs, imm$

❏  $Addr \leftarrow R[rs] + SignExt(imm)$

计算地址

❏  $MEM[Addr] \leftarrow R[rt]$

写数据



- 基本和Load指令的相同
- 增加读出数据2到数据存储器的通道
- 增加写存储器的控制信号





# BEQ指令



## 跳转指令

- BEQ rs, rt, imm
- 改变PC的值
- $\text{cond} \leftarrow R[\text{rs}] - R[\text{rt}]$  计算条件
- If ( cond eq 0) then 判断条件
- $\text{pc} \leftarrow \text{pc} + 4 + \text{SignExt}(\text{imm}) * 4$  else 计算下一条指令的地址
- $\text{pc} \leftarrow \text{pc} + 4$



# 下一条指令的地址



## 指令存储器按字节编址

顺序执行

$$PC \leftarrow PC + 4$$

跳转

$$PC \leftarrow PC + 4 + \text{SignExt}(\text{imm}) * 4$$

# Jump指令



## Jump指令

J target

PC[31: 0] ← PC[31: 28] || target[25:0] || 00

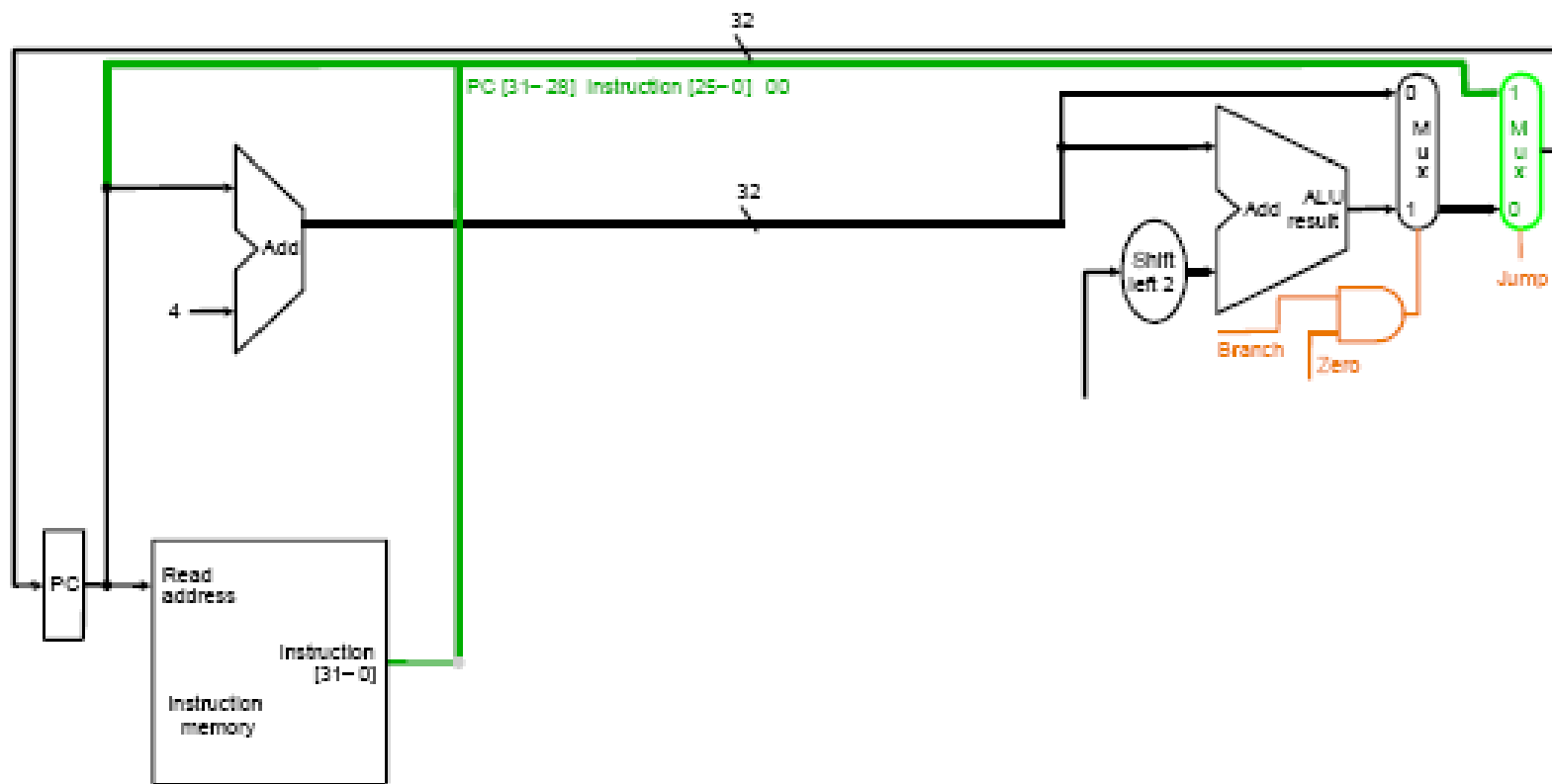


Jump Target Address

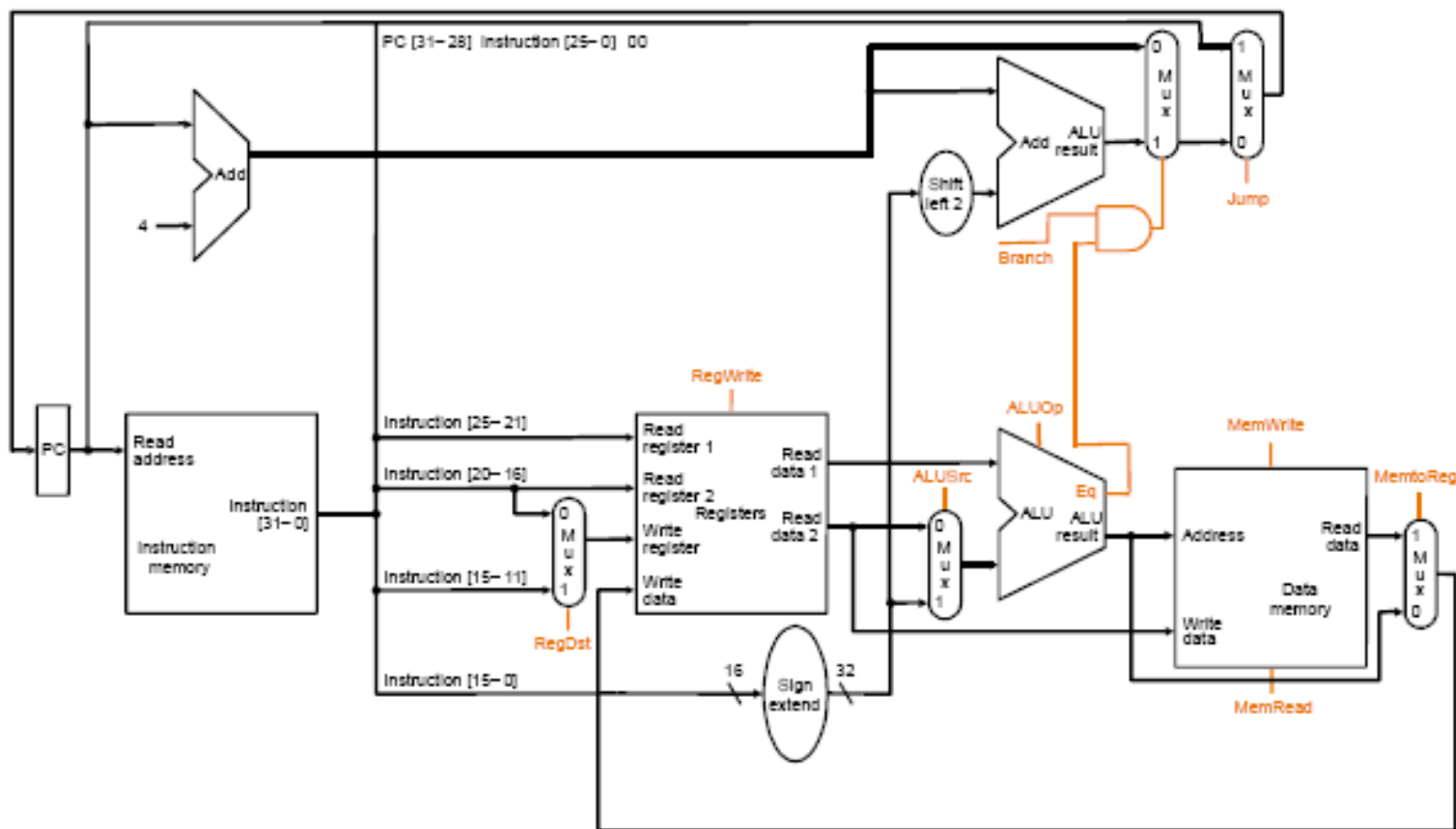
# 支持BEQ、Jump指令



在取指部件增加多路选择器



# 组装到一起



## 所需要的控制信号

- RegDst、RegWrite、ExtOp、ALUSrc、ALUOp、Branch、Jump、MemRead、MemWrite、MemtoReg

## ✚ 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

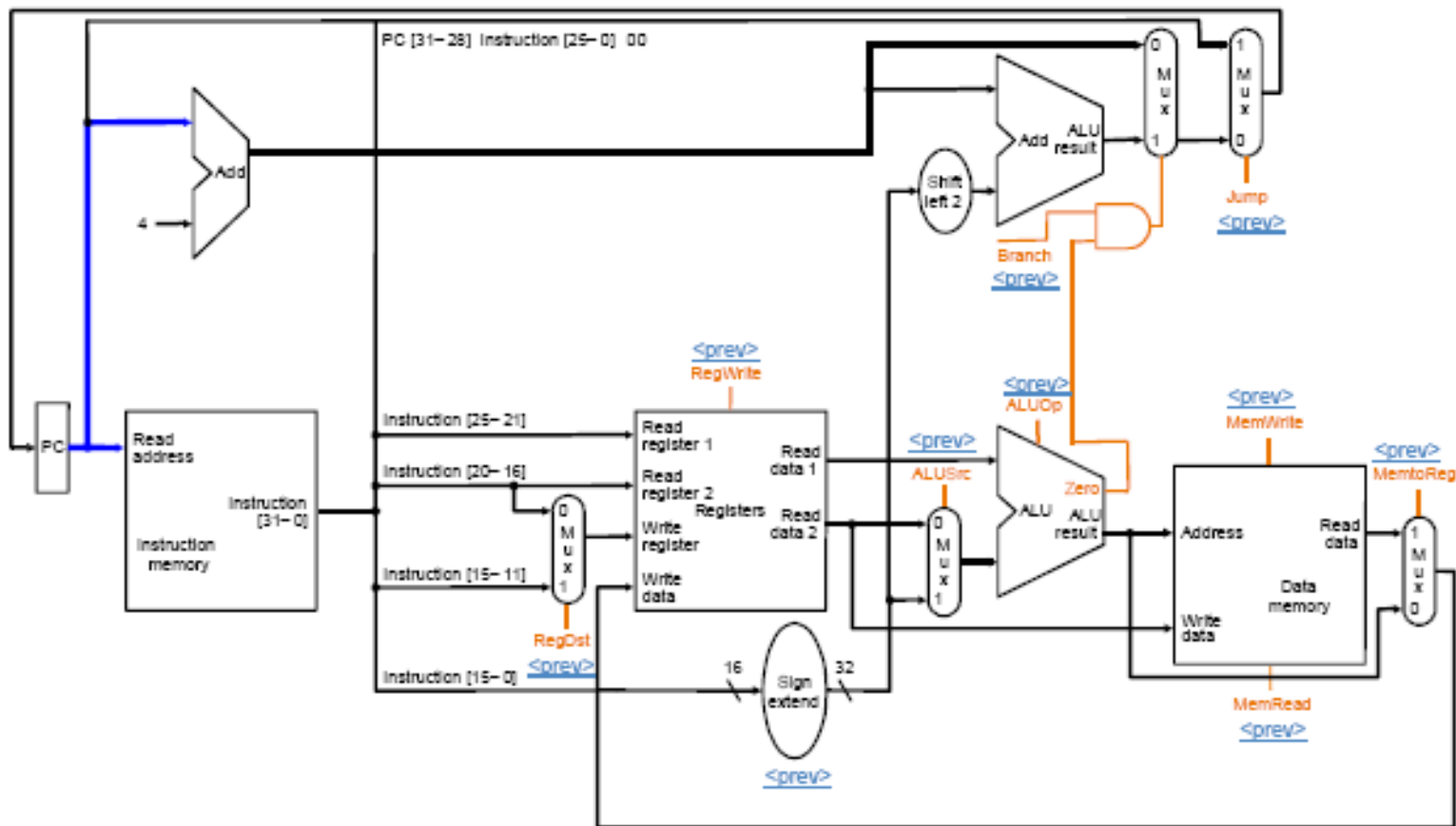
## ✚ 控制对象

- ✚ ALU的运算
- ✚ 寄存器组和存储器的写入
- ✚ 多路选通器

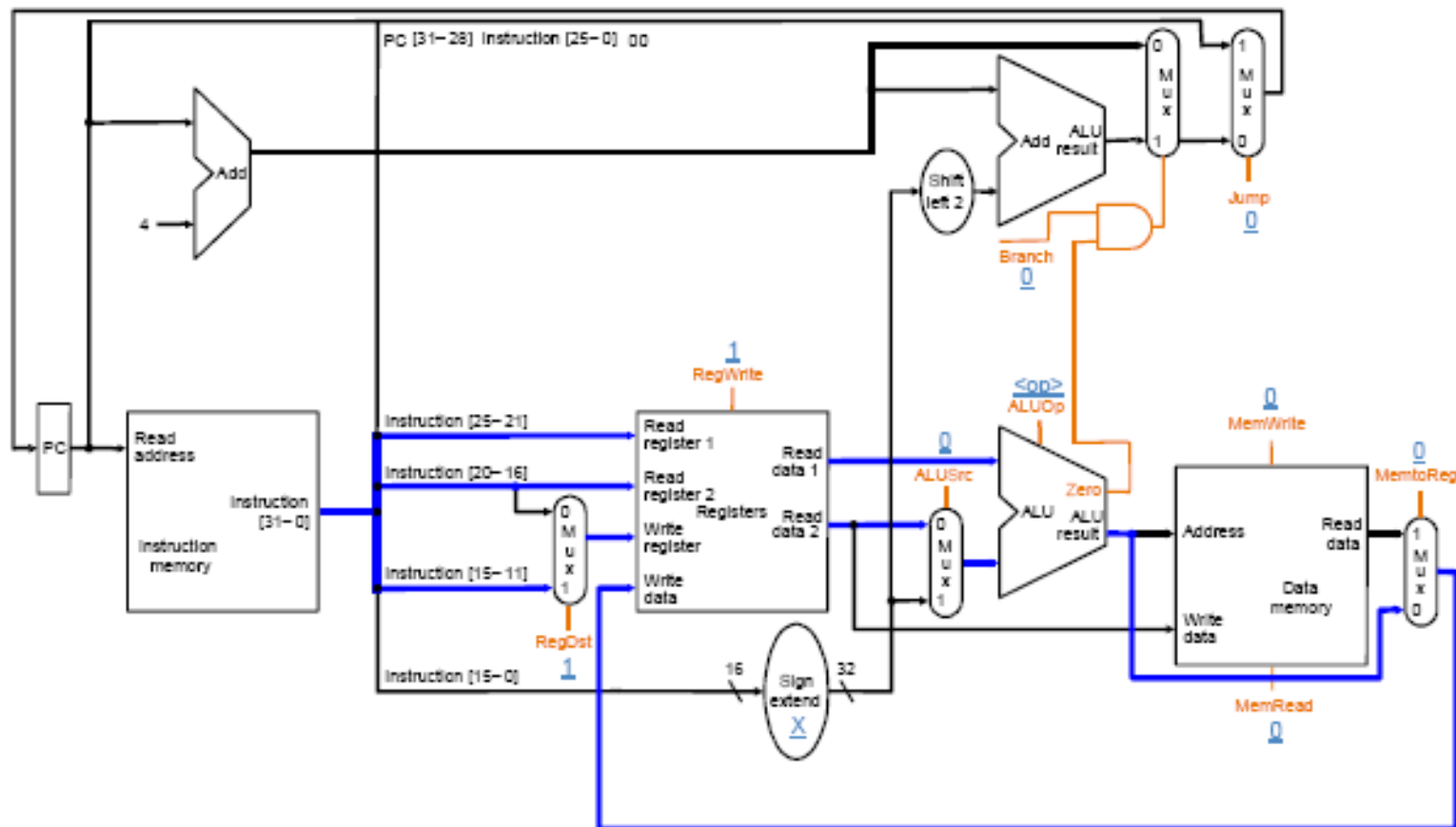
## ✚ 时钟周期开始时读取指令

- ✚ 与具体指令无关

# 时钟周期开始时的控制

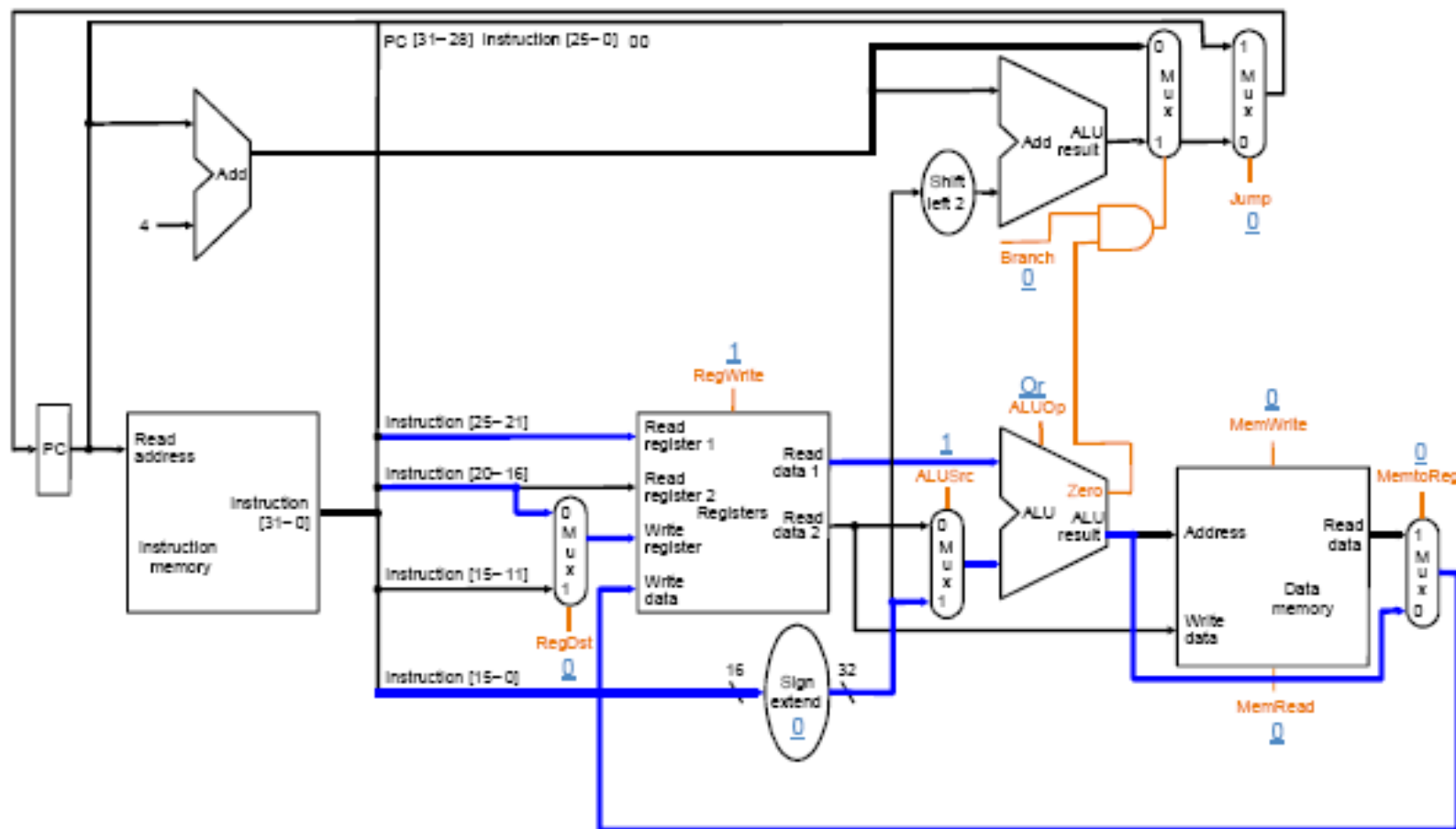


# 算逻运算指令的控制

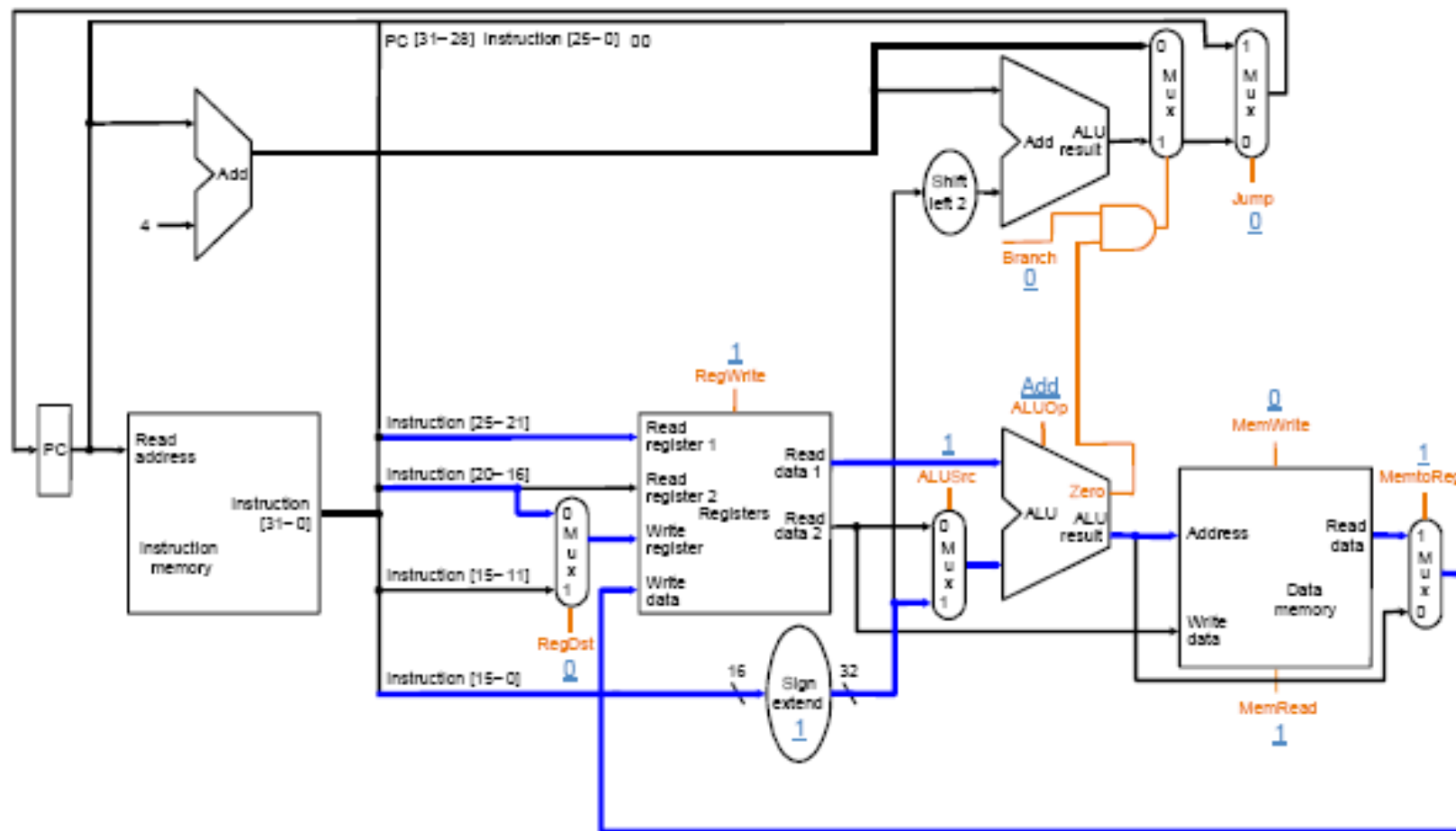




# ORI指令的控制



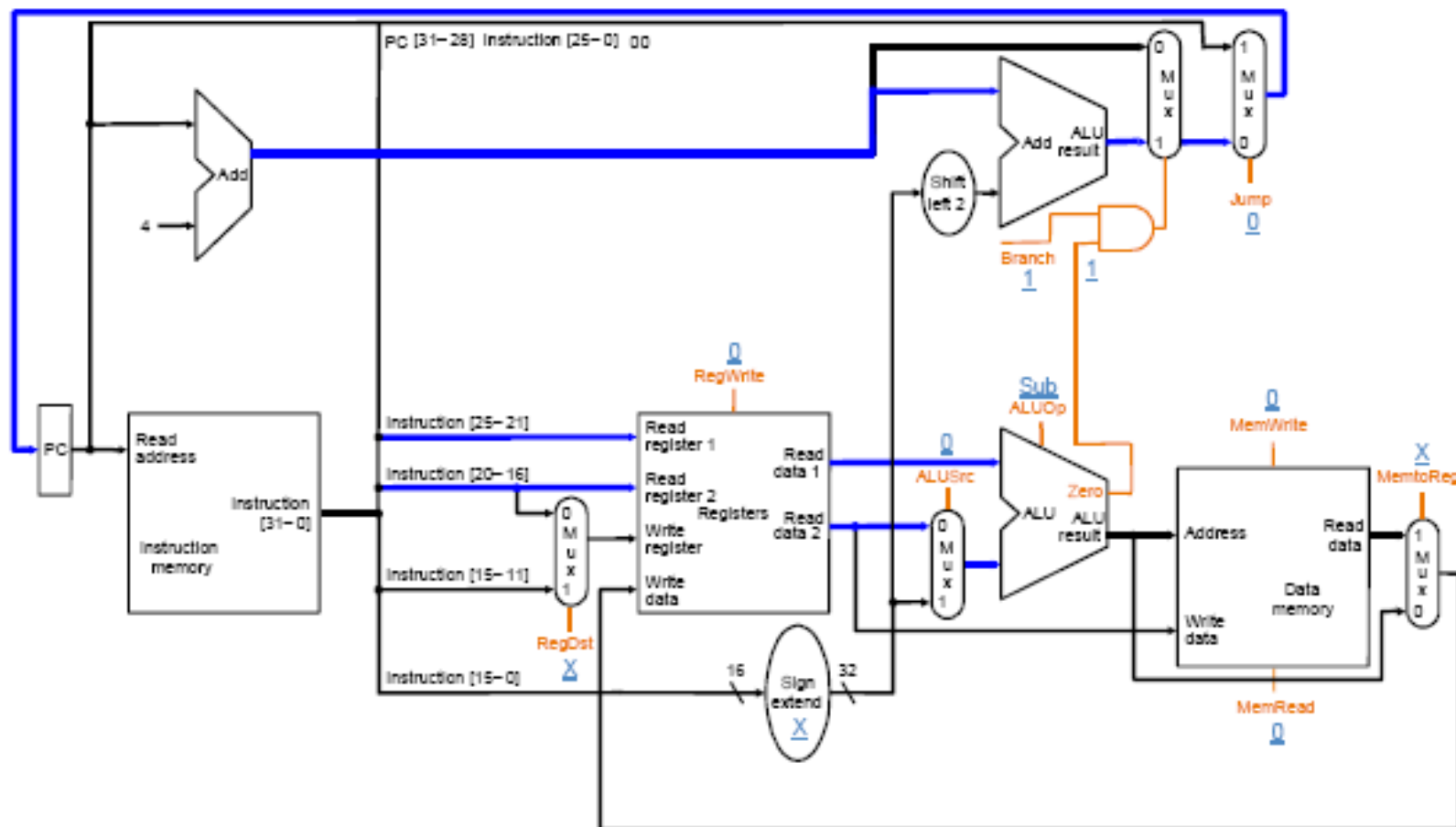
# Load指令



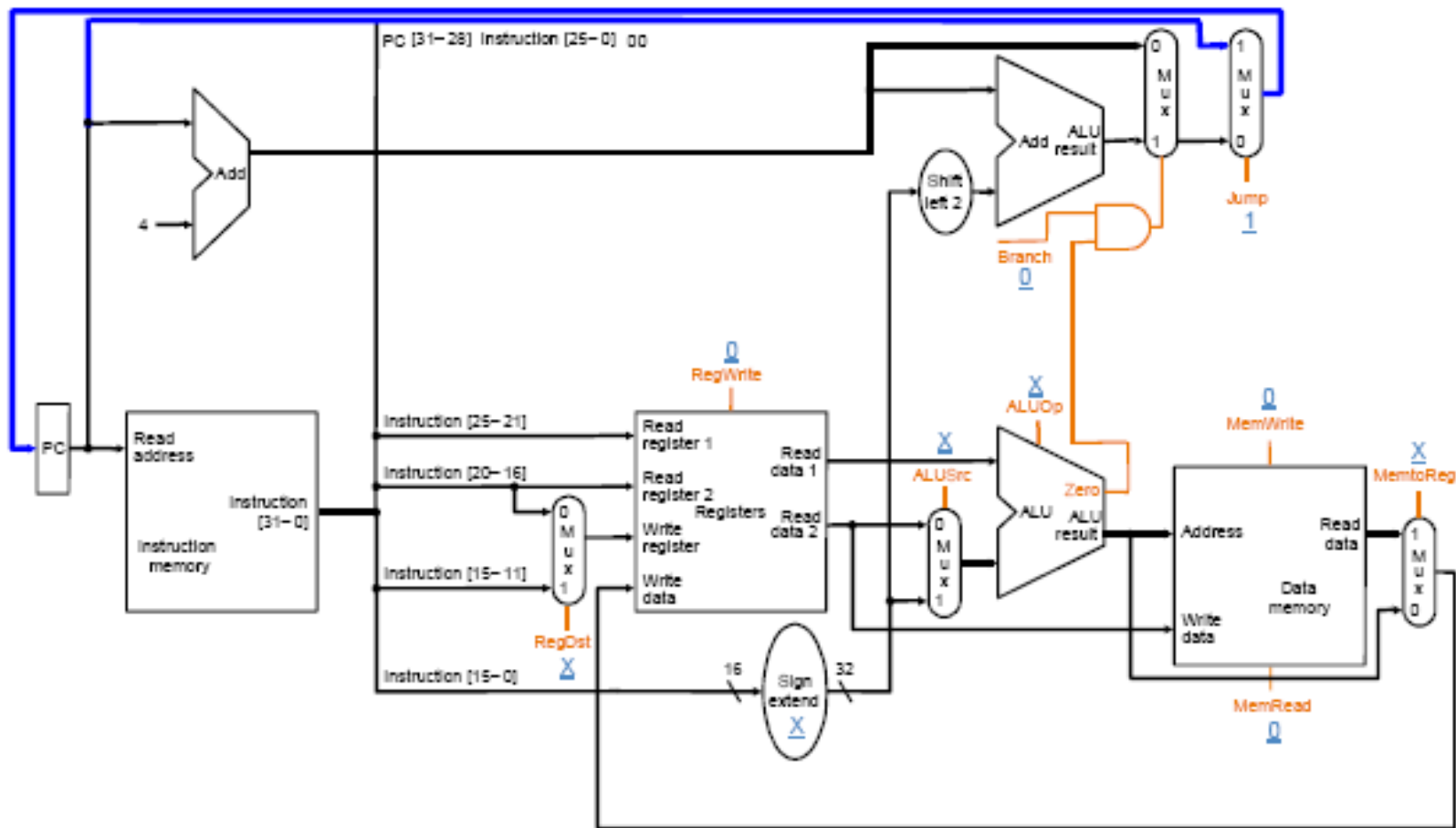
清华大学  
Tsinghua University



# Beq指令



# Jump指令



# 控制信号表



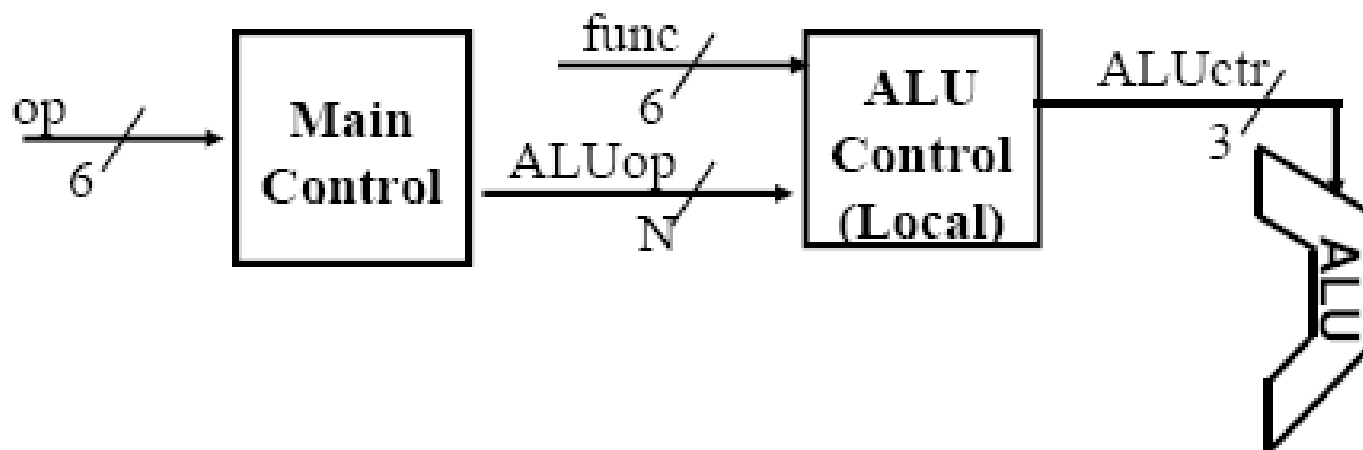
func	10 0000	10 0010	Not Important				
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	add	sub	ori	lw	sw	beq	jump
RegDst	1	1	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MemtoReg	0	0	0	1	x	x	x
RegWrite	1	1	1	1	0	0	0
MemWrite	0	0	0	0	1	0	0
Branch	0	0	0	0	0	1	0
Jump	0	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x	x
ALUctr<2:0>	Add	Sub	Or	Add	Add	Sub	xxx

# 分级控制



只有ALU需要Func字段

将FUNC字段直接传给ALU，在ALU处进行译码



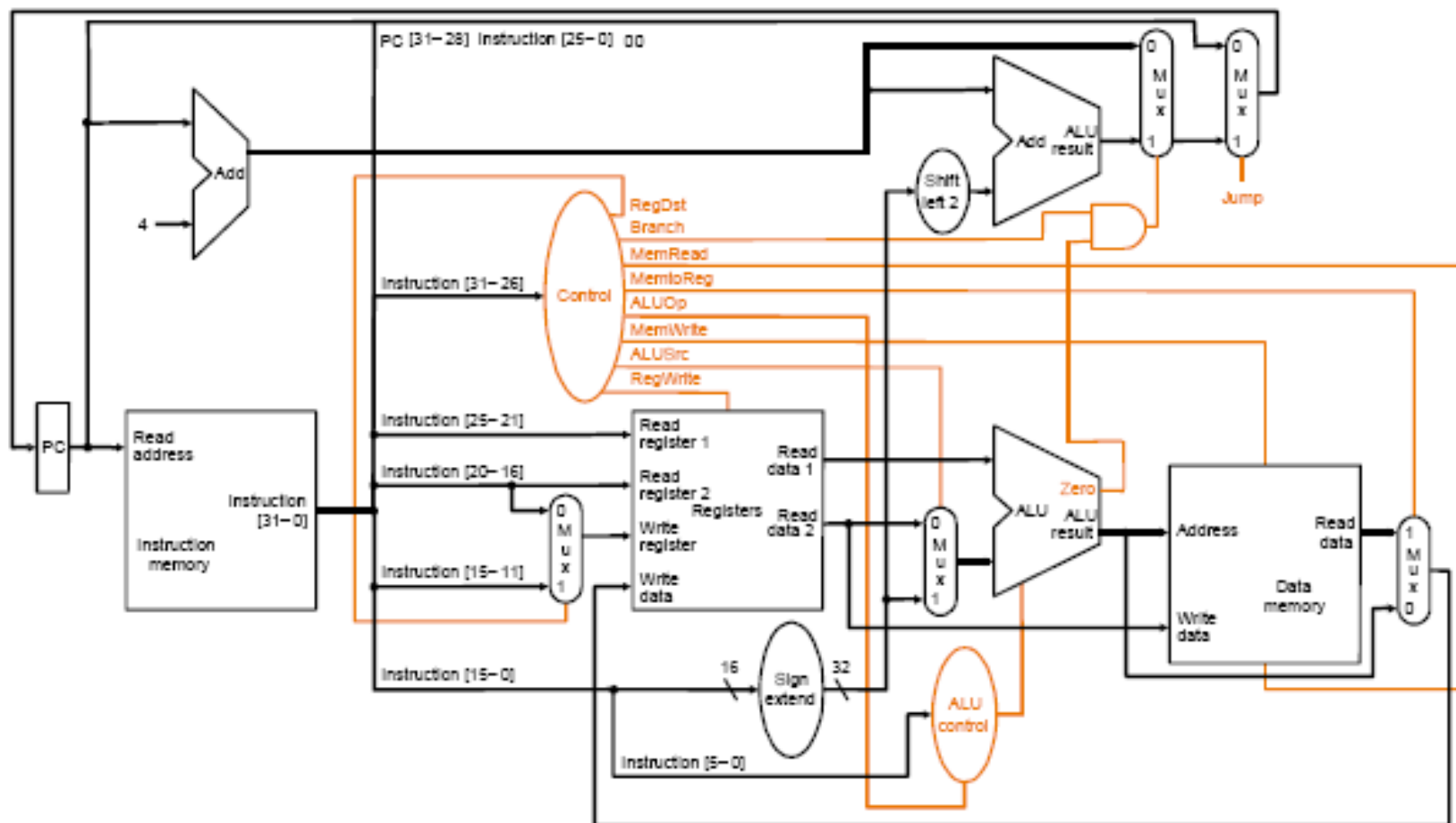
# 多级控制的信号汇总



op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	<b>R-type</b>	<b>ori</b>	<b>lw</b>	<b>sw</b>	<b>beq</b>	<b>jump</b>
<b>RegDst</b>	1	0	0	x	x	x
<b>ALUSrc</b>	0	1	1	1	0	x
<b>MemtoReg</b>	0	0	1	x	x	x
<b>RegWrite</b>	1	1	1	0	0	0
<b>MemWrite</b>	0	0	0	1	0	0
<b>Branch</b>	0	0	0	0	1	0
<b>Jump</b>	0	0	0	0	0	1
<b>ExtOp</b>	x	0	1	1	x	x
<b>ALUOp&lt;N:0&gt;</b>	“R-type”	Or	Add	Add	Subtract	xxx



# 完整的单周期CPU



# 单周期CPU特点



## 优点

- ❑ 每条指令占用一个时钟周期
- ❑ 逻辑设计简单，时序设计也简单

## 缺点

- ❑ 各组成部件的利用率不高
  - ◆ 维持有效信号
- ❑ 时钟周期应满足执行时间最长指令的要求
  - ◆ Load指令
- ❑  $CPI = 1$

# 小结



## ❖ 单周期CPU设计

- ❖ 全部控制信号

- ❖ 无需状态信息

- ❖ 控制信号生成：组合逻辑电路

  - ◆ 输入：OP、FUNC

  - ◆ 输出：完成指令功能所需要的全部控制信号

# Project3



## ❖ 实验目的和要求

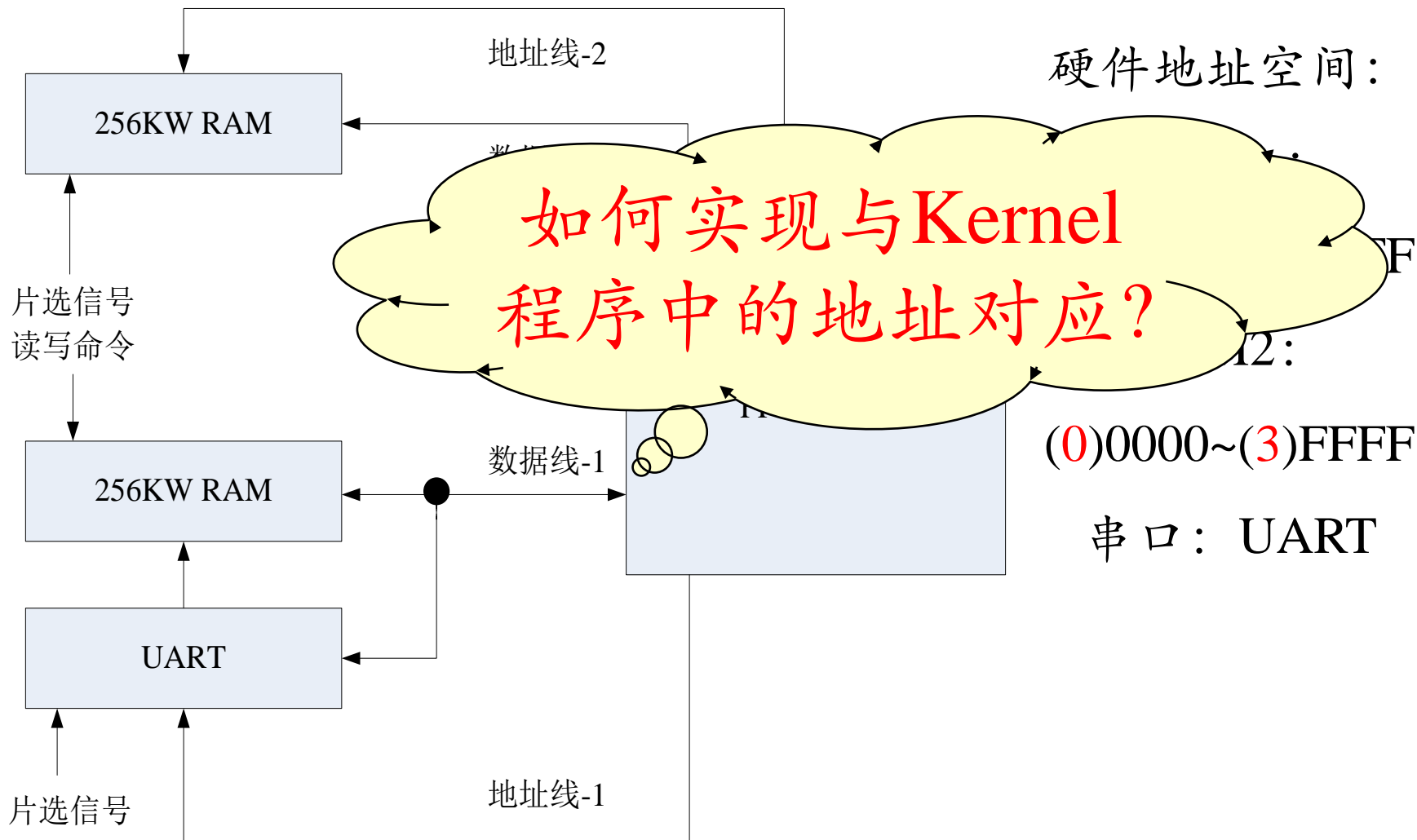
- ❖ 熟悉THINPAD的内存和串口访问方式
- ❖ 完成基本要求，并可进行一定的扩展

## ❖ 连接方式和地址空间分配

- ❖ 双存储体（型号：IS61LV25616-10TI）
- ❖ 指令系统地址空间
  - ◆ 64MW RAM（实际256MW）
  - ◆ 64MW RAM（实际256MW）
  - ◆ 多个串口(UART、USB等)

## ❖ 控制信号生成

# THINPAD主存



# 对存储/外设的控制信号



## ✚ 信号源

- ✚ RAM1\_EN、RAM1\_OE、RAM1\_RW
- ✚ RAM2\_EN、RAM2\_OE、RAM2\_RW

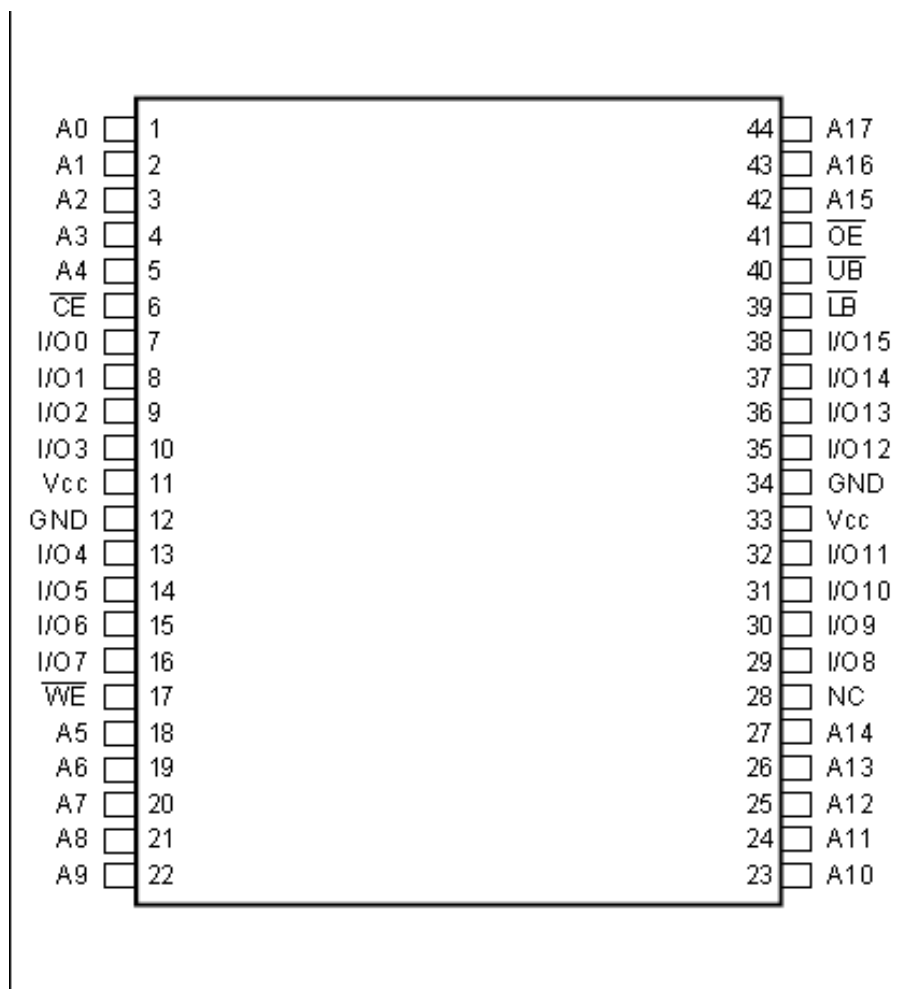
## ✚ 对存储器的控制信号

- ✚ 基本存储： /CE /WE /OE /UB /LB
- ✚ 扩展存储： /CE /WE /OE /UB /LB

## ✚ 对串口UART的控制信号

- ✚ CLK RAM1\_EN RAM1\_OE RAM1\_RW
- ✚ Rdn dataready wrn tbre tsre
- ✚ 指示书中表5.6有错

# 管脚图



IS61LV25616

# 控制总线设计



## ⊕ 时钟信号

⊞ CLK1与CLK0

⊞ 保证时序关系

## ⊕ 读写信号

/CE	/OE	/WE	
0	1	0	内存写
0	0	1	内存读
1	1	0	I/O写
1	1	1	I/O读



# 阅读和思考



## 阅读

- 教材4.1~4.4

## 思考

- 单周期CPU设计中是否存在什么不足?
- 可以从哪几个方面对其进行改进?

## 实践

- 设计能实现THCO MIPS指令系统的数据通路(单周期), 并列出各硬件模块所需要的控制信号。
- Project3: 实验指导书5.4&5.5, **11月8日**前完成, 实验报告截止至**11月11日**