清華大學
Tsinghua University

第二单元　第三讲

单周期CPU控制器设计

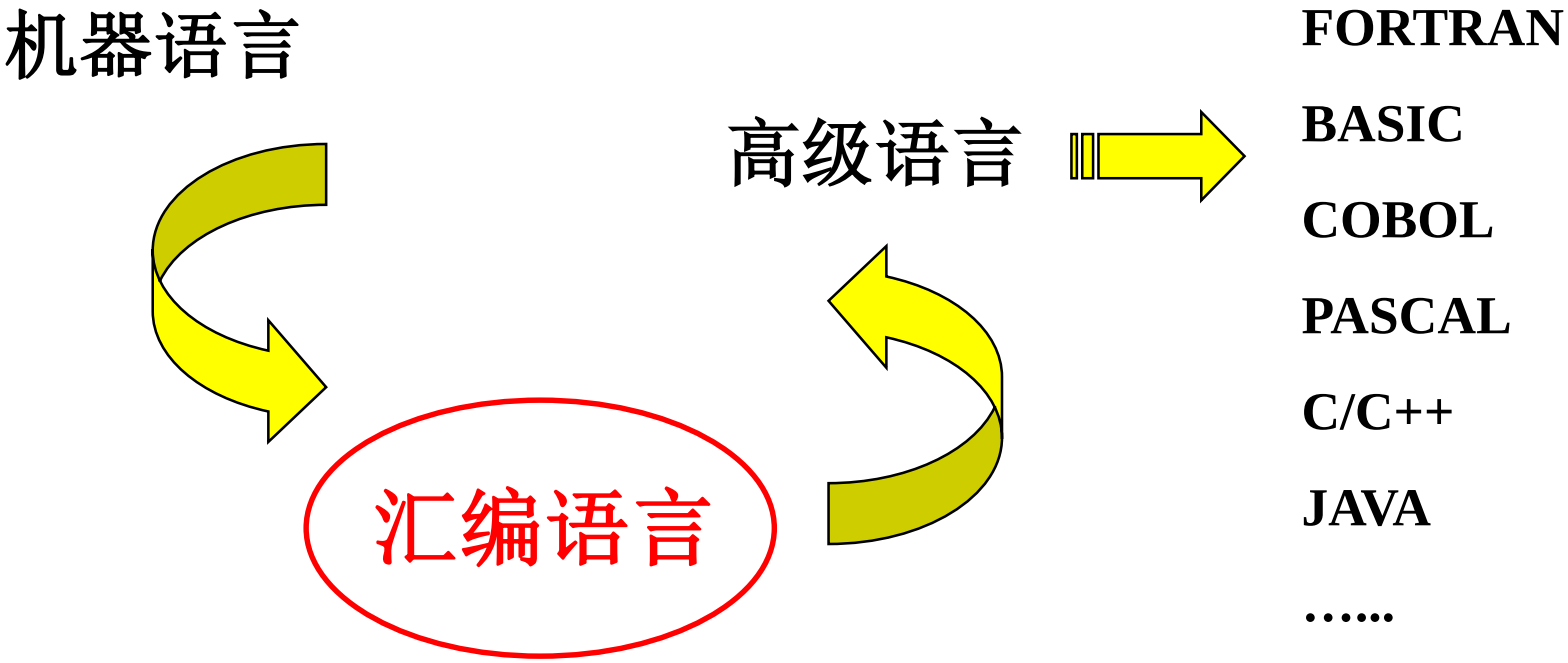
刘卫东

计算机科学与技术系

本讲提要

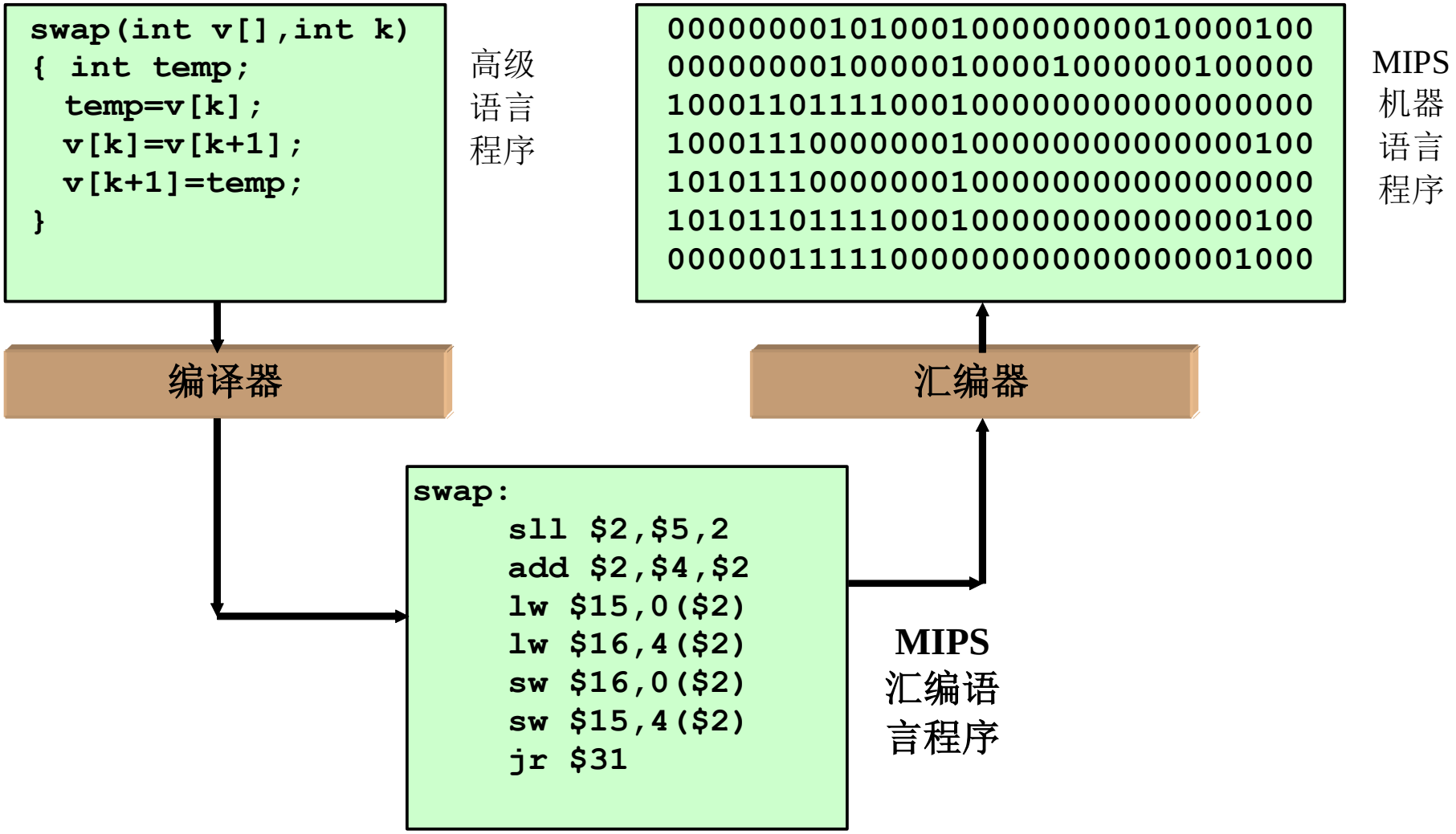
- 程序的组成和执行
- 指令执行方式和步骤
- 单周期CPU设计
  - 指令集：7条典型指令
  - 数据通路
  - 控制器
- Project 3
  - 存储器及串口访问

计算机程序语言的发展

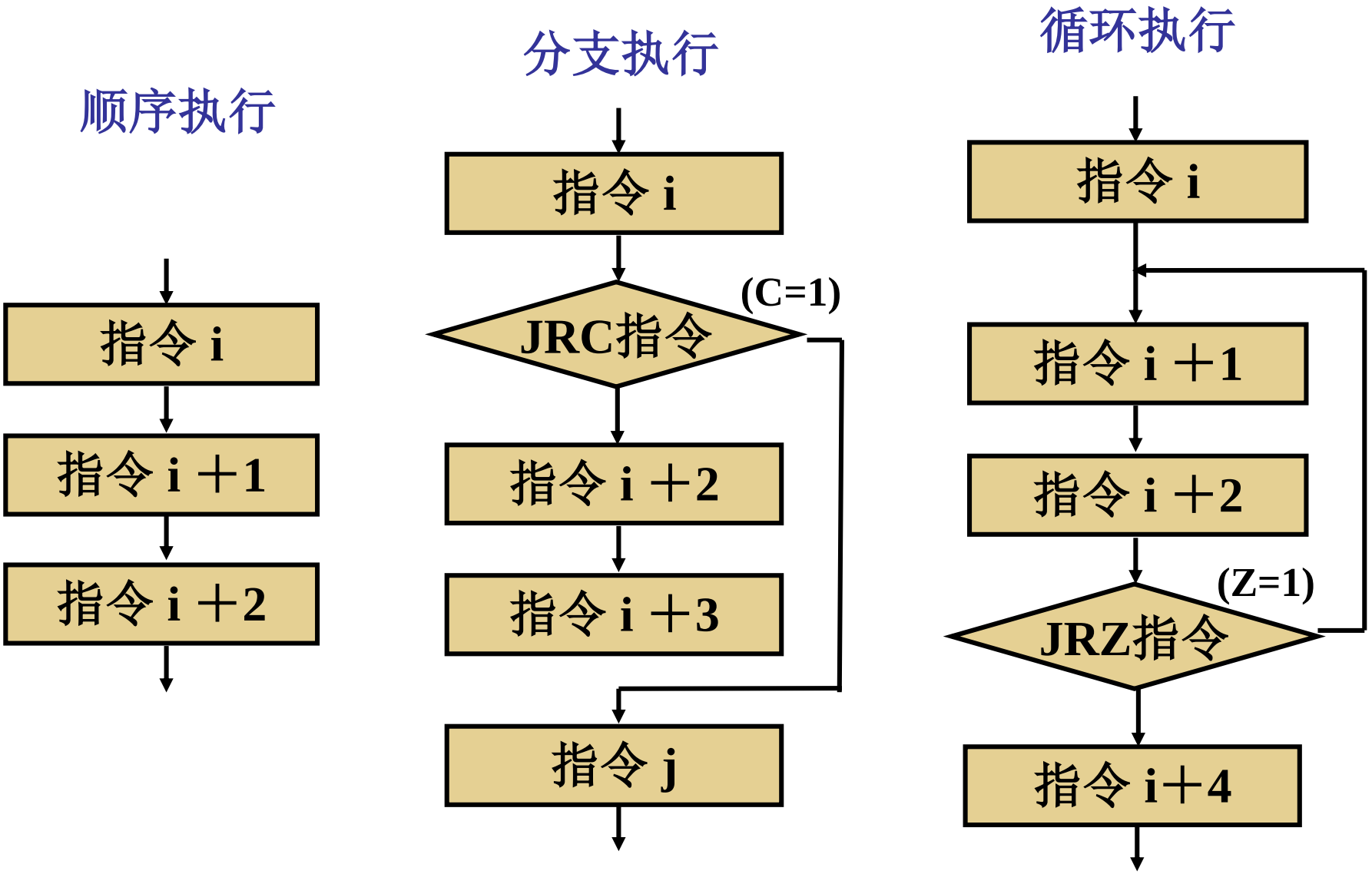


发展历程

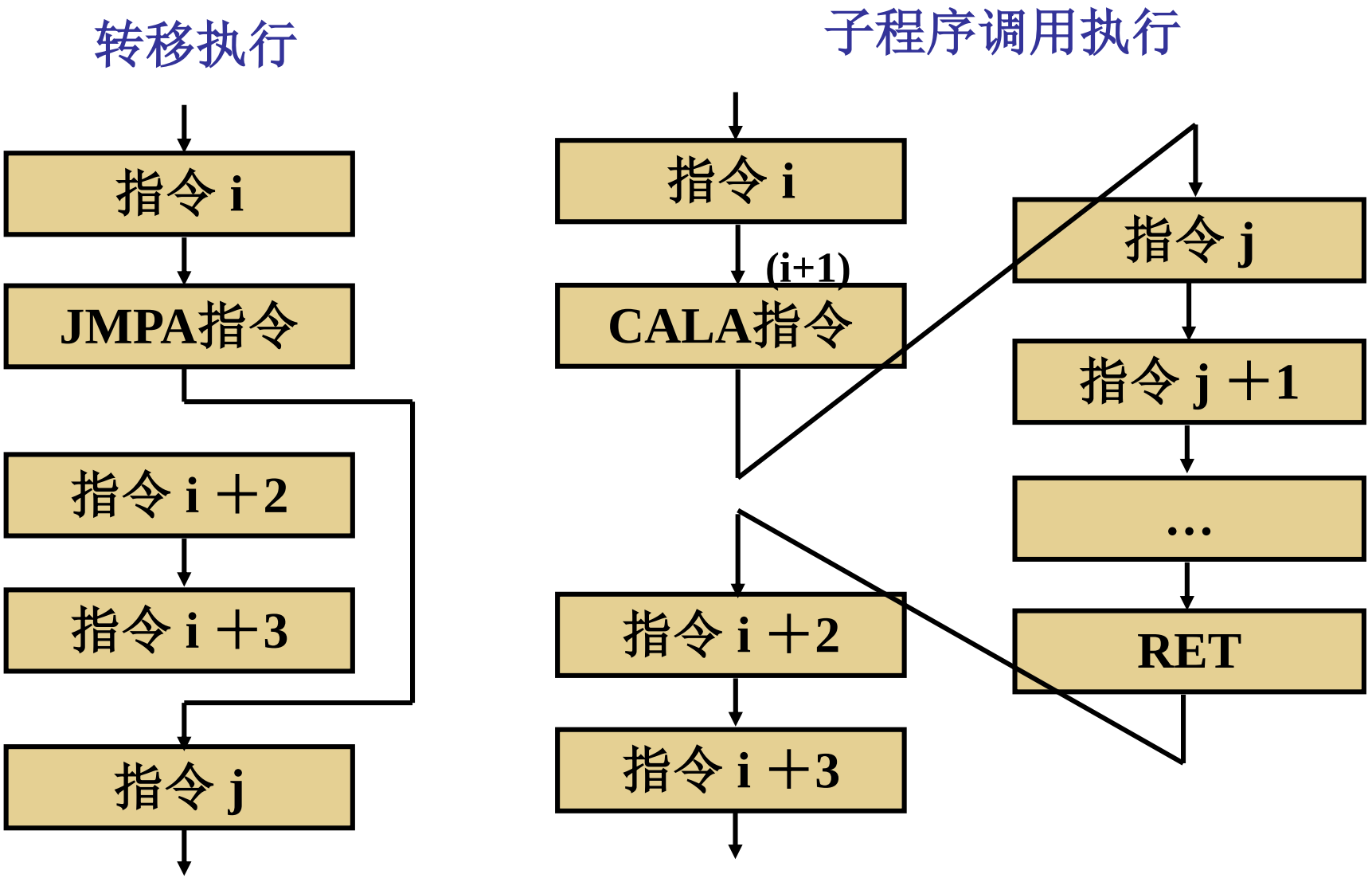
计算机程序的不同层次


swap(int v[],int k)
{ int temp;
temp=v[k];
v[k]=v[k+1];
v[k+1]=temp;
}
高级语言程序
00000000101000100000000010000100
00000000100000100001000000100000
10001101111100010000000000000000
10001110000000100000000000000100
10101110000000100000000000000000
10101101111100010000000000000100
00000011111000000000000000001000
MIPS机器语言程序
编译器
汇编器
swap:
sll $2,$5,2
add $2,$4,$2
lw $15,0($2)
lw $16,4($2)
sw $16,0($2)
sw $15,4($2)
jr $31
MIPS汇编语言程序

典型的程序执行流程


顺序执行
指令 i
指令 i +1
指令 i +2
分支执行
指令 i
JRC指令
(C=1)
指令 i +2
指令 i +3
指令 j
循环执行
指令 i
指令 i +1
指令 i +2
JRZ指令
(Z=1)
指令 i+4

典型的程序执行流程


转移执行
指令 i
JMPA指令
指令 i ＋2
指令 i ＋3
指令 j
子程序调用执行
指令 i
(i+1)
CALA指令
指令 i ＋2
指令 i ＋3
指令 j
指令 j ＋1
…
RET

汇编语言程序

- 程序设计
  - 由顺序、分支、循环三种程序结构构成
  - 采用指令系统中的指令编写
  - 汇编语句与机器指令相对应
- 程序功能实现
  - 实现每条指令的功能
  - 寻址下一条指令，并自动执行
    - 顺序
    - 分支
    - 循环

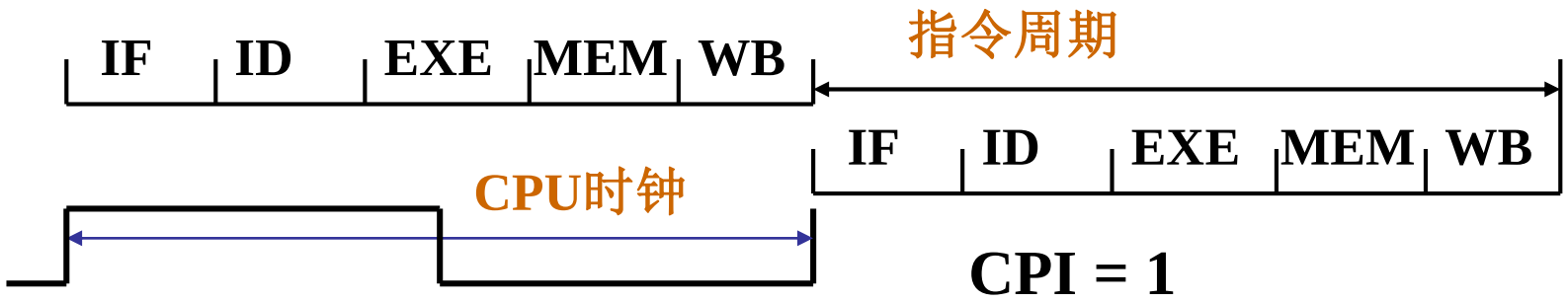
CPU设计

- 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

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。



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）

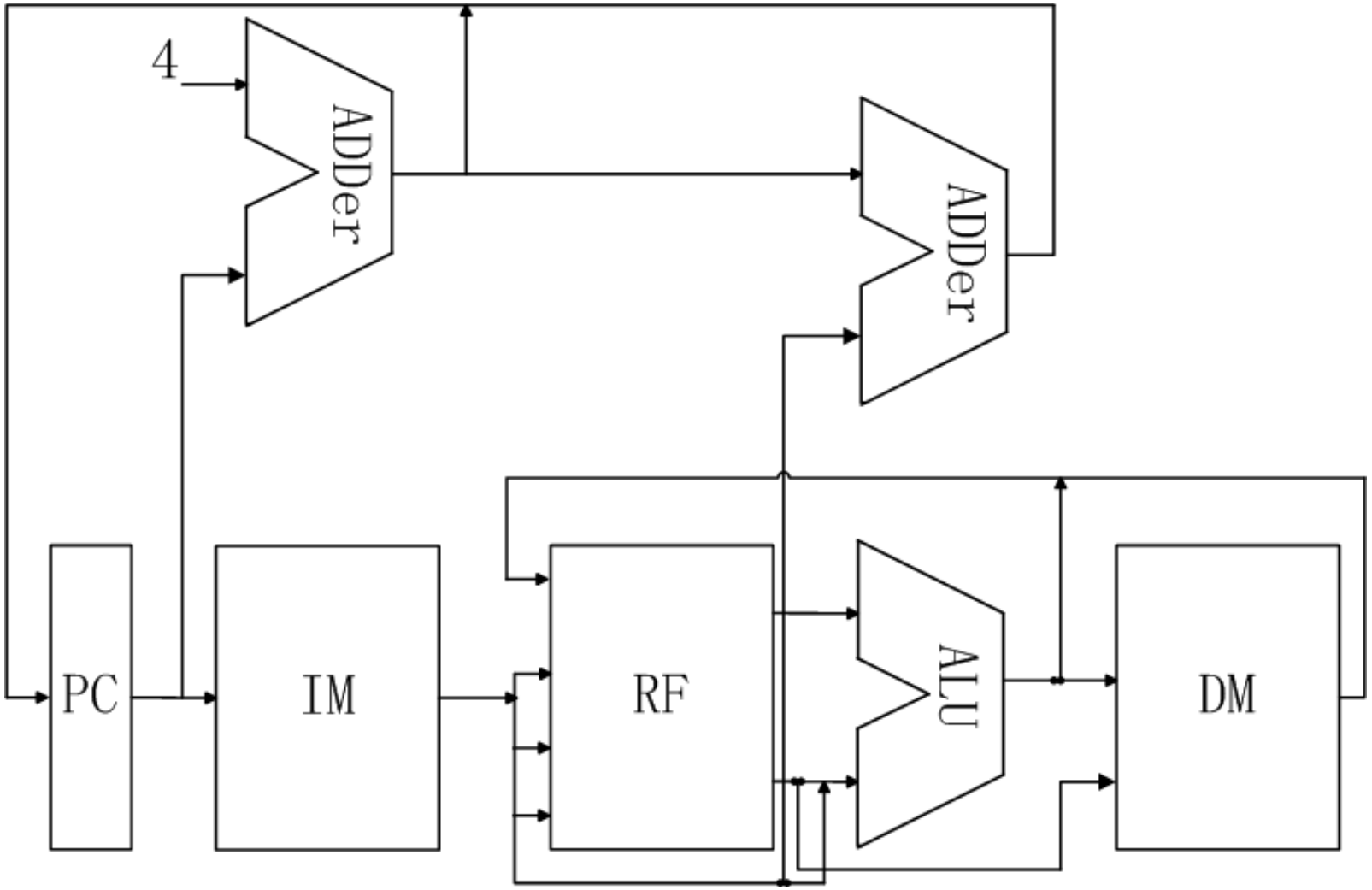
实现的指令集

- 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

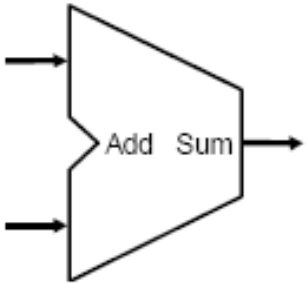
设计思路

- 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

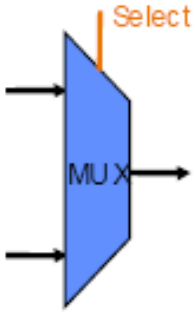
最初步的Datapath



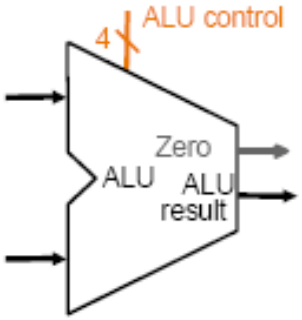
使用的组合逻辑部件



Adder



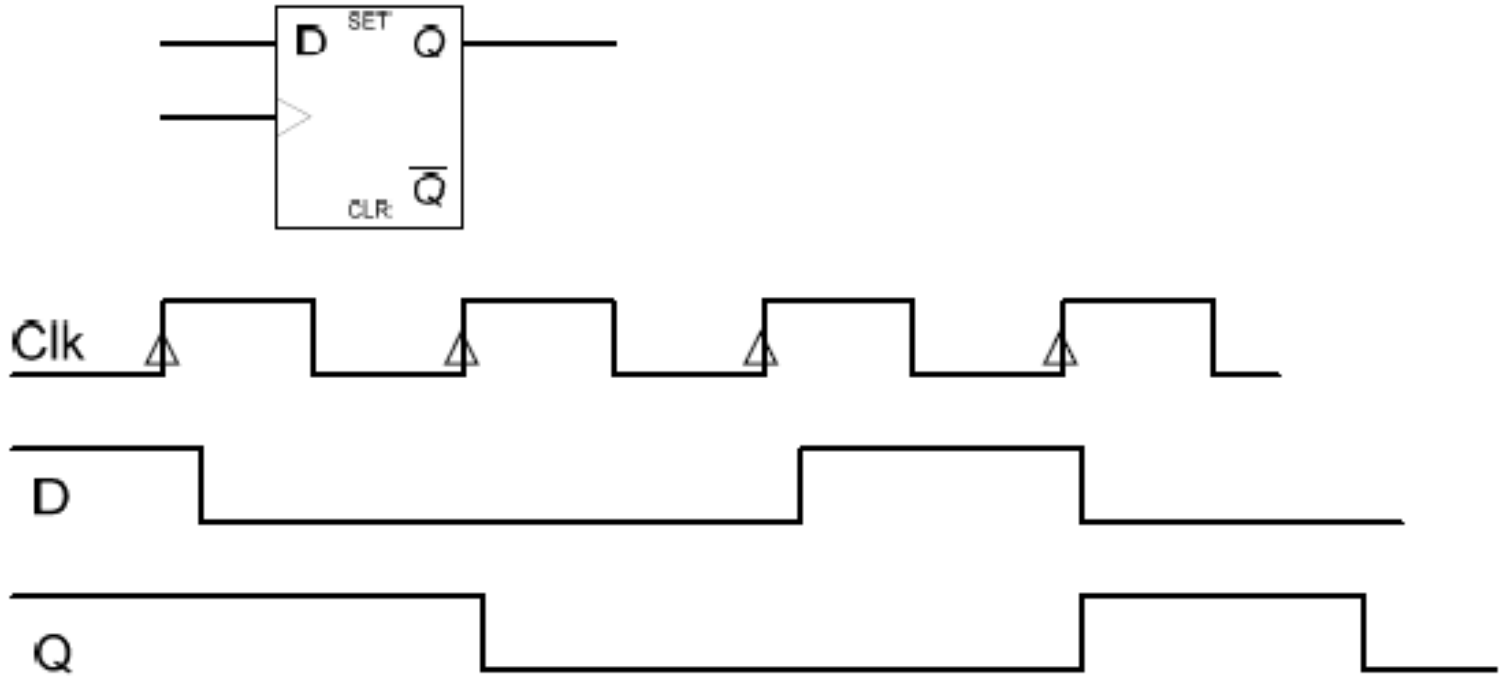
MUX



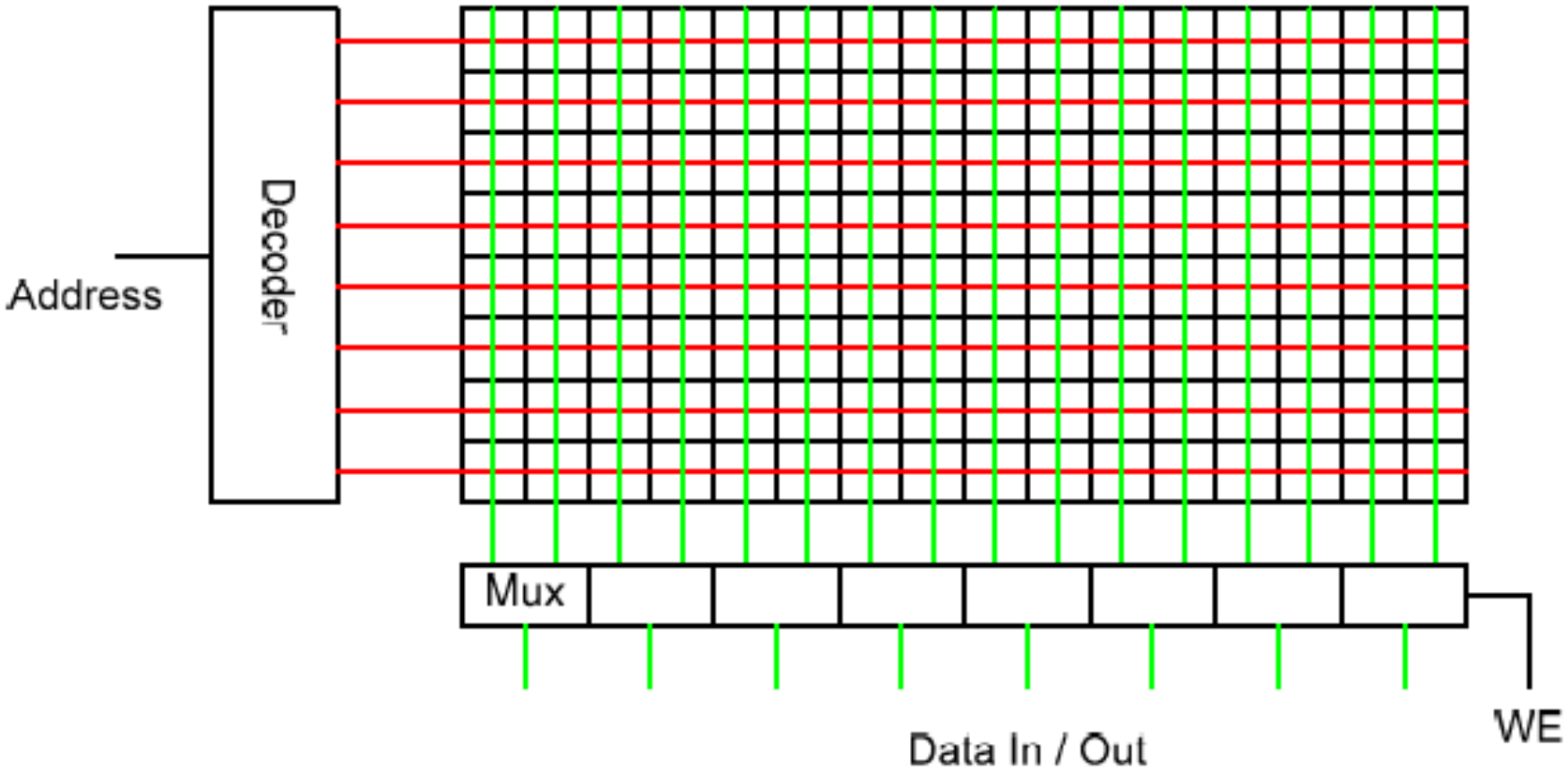
ALU

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

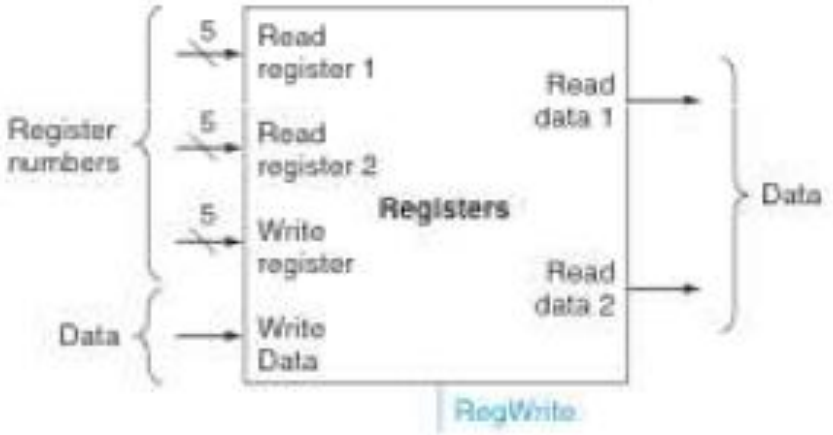


存储器



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

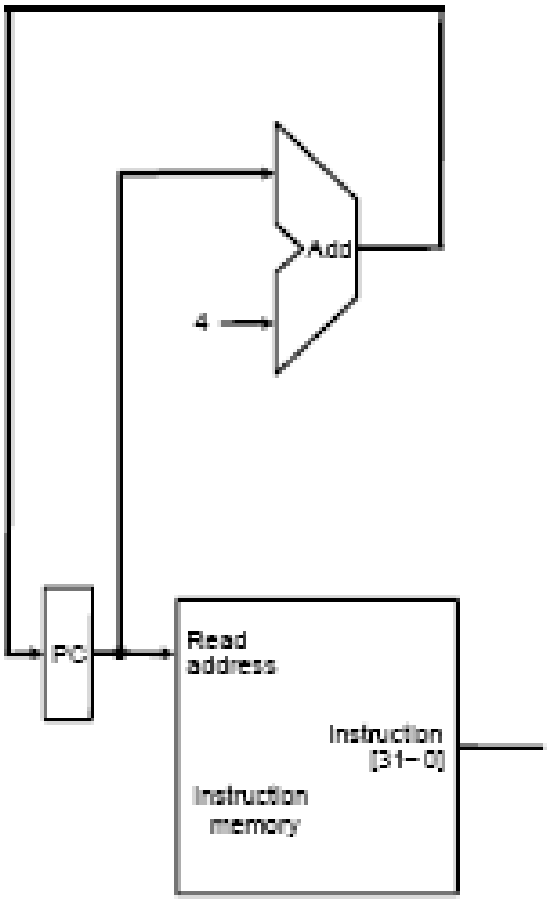


- 3个地址端口，2个用来读，1个用来写
- 每个时钟周期可以完成3次访问

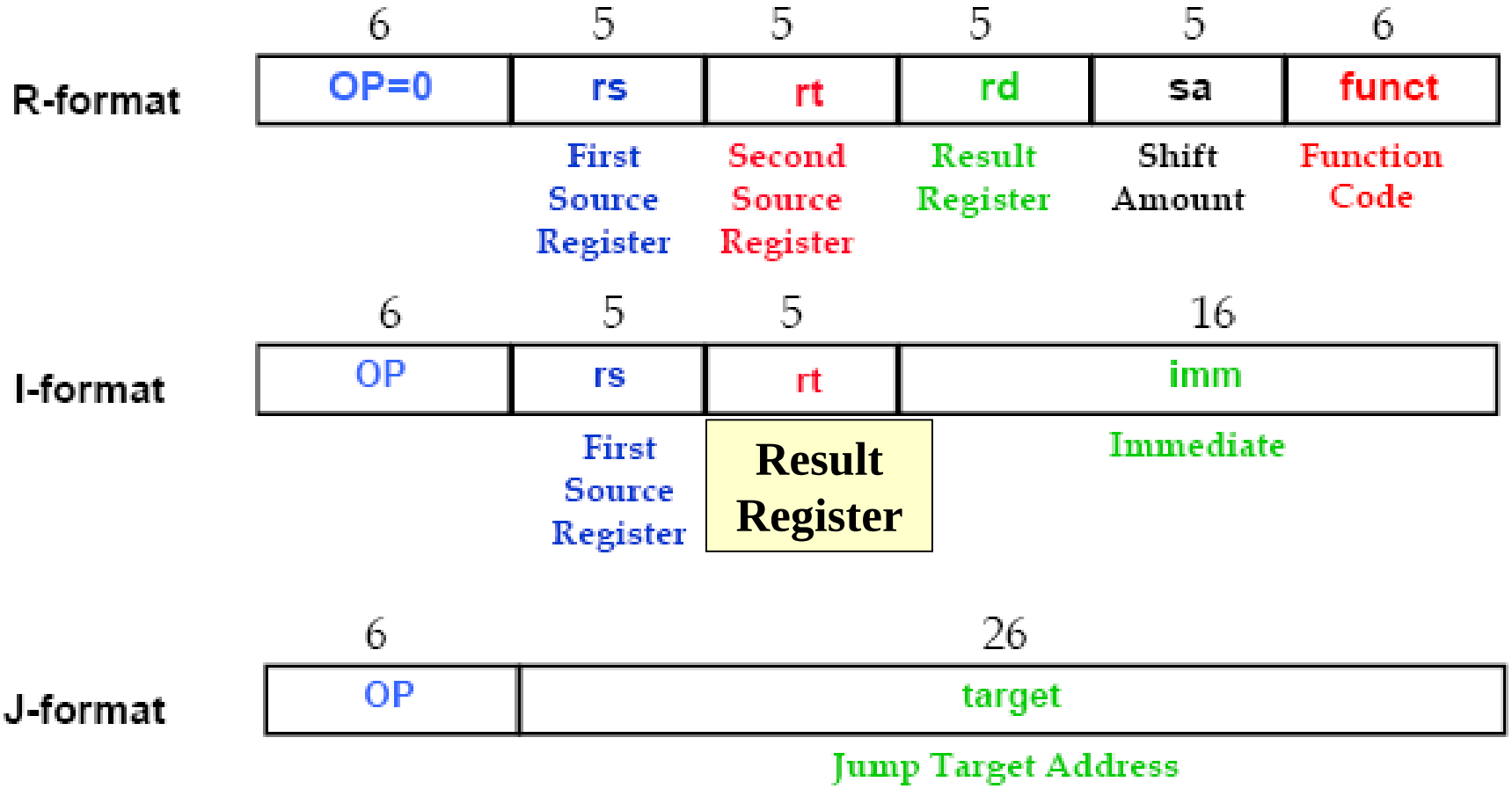
第一步：取指

- 当前指令存储在存储器中，用PC指示指令地址
- Instr = MEM[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？


R-format
6
5
5
5
5
6
OP=0
rs
rt
rd
sa
funct
First Source Register
Second Source Register
Result Register
Shift Amount
Function Code
I-format
6
5
5
16
OP
rs
rt
imm
First Source Register
Result Register
Immediate
J-format
6
26
OP
target
Jump Target Address

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

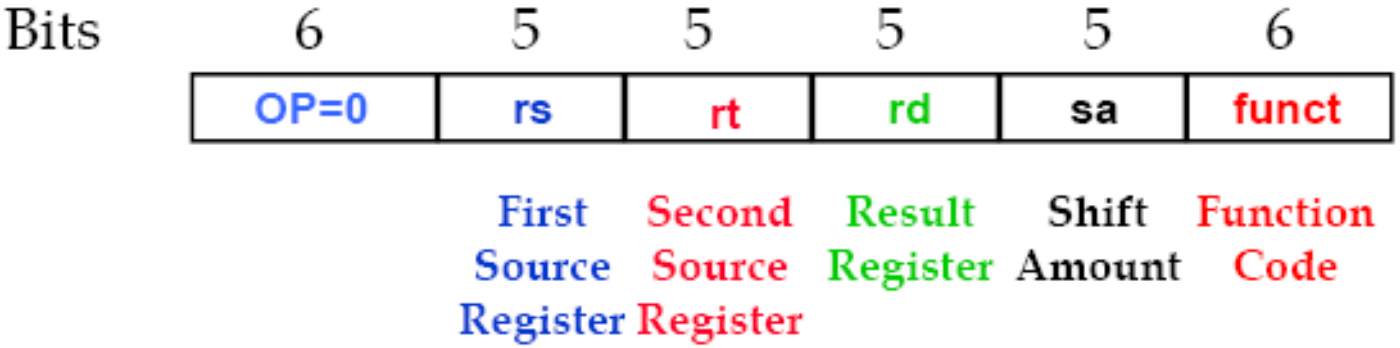
寄存器间运算指令

- 两条算术指令
  - Addu rd rs rt
  - Subu rd rs rt

- 运算

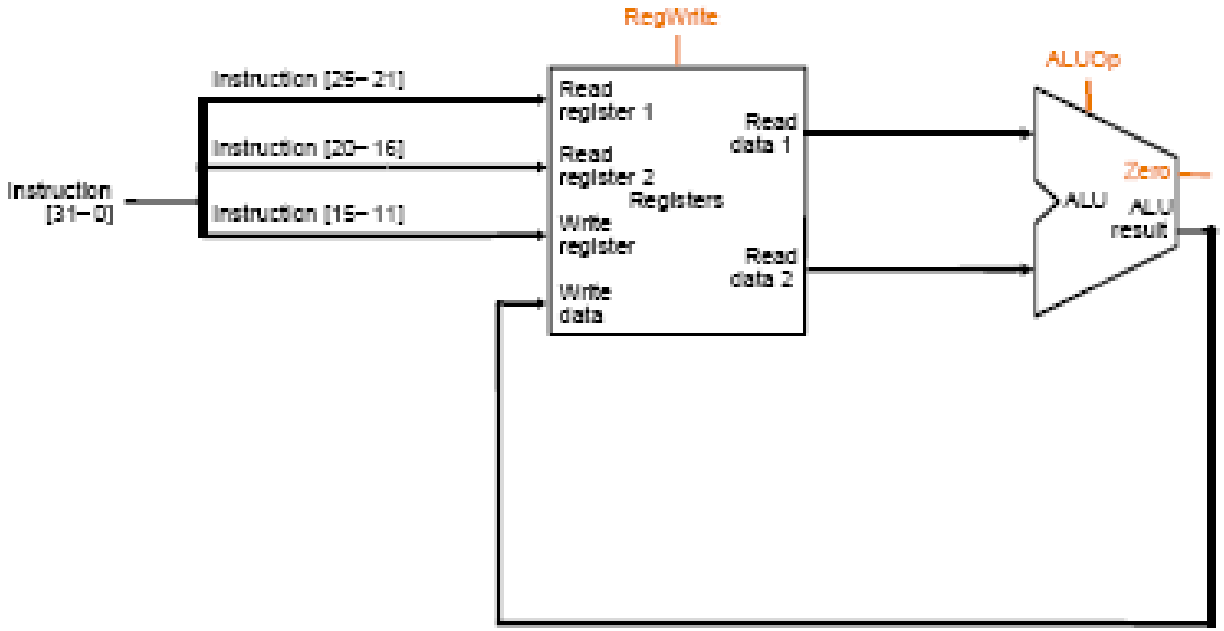
R[rd] <- R[rs] + R[rt]; Add operation

R[rd] <- R[rs] - R[rt]; Sub operation



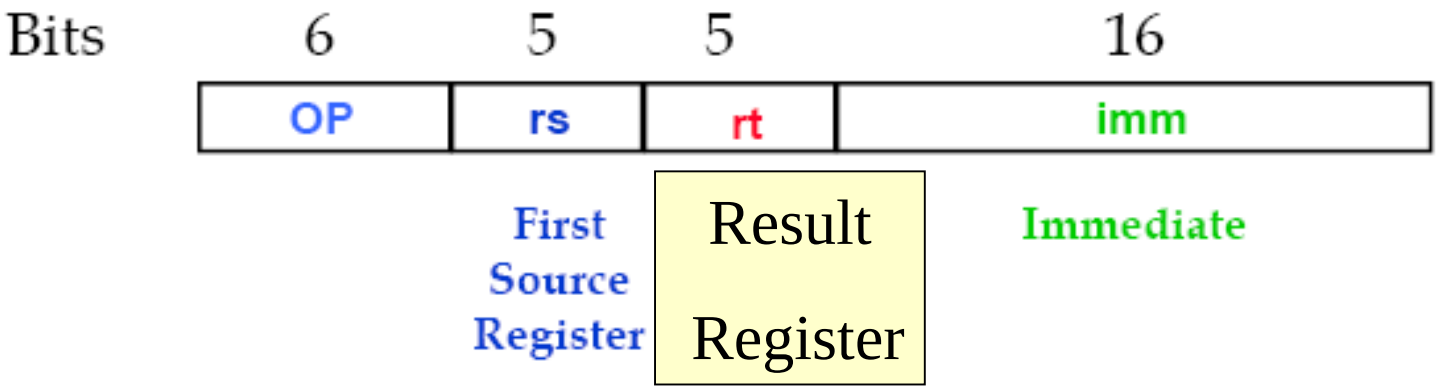
实现R型指令的数据通路

- R[rd] ← R[rs] 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



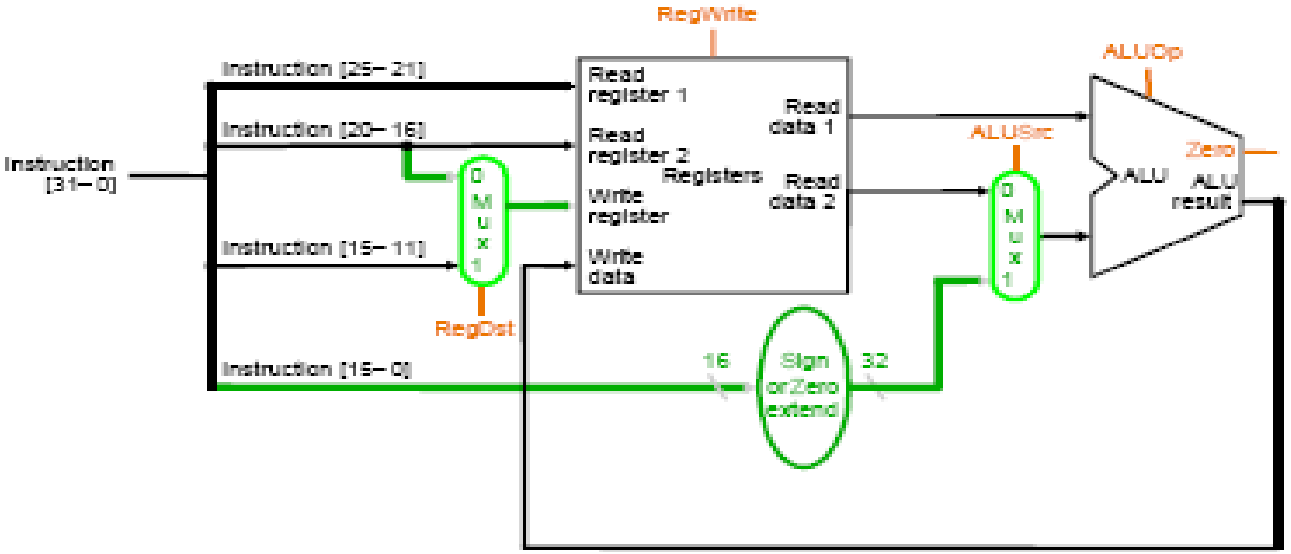
ORI指令

- OR 立即数指令
  - Ori rt rs imm
  - R[rt] ← R[rs] or ZeroExt(imm)
- 需要将Instr[15:0]送到ALU中
- 目的寄存器字段有变化



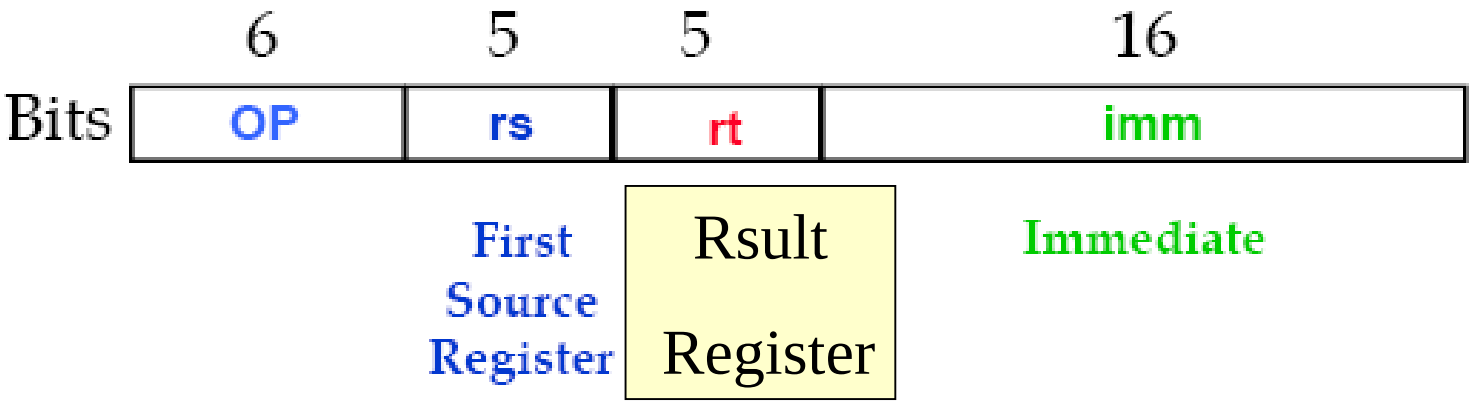
立即数型指令的实现

- 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）



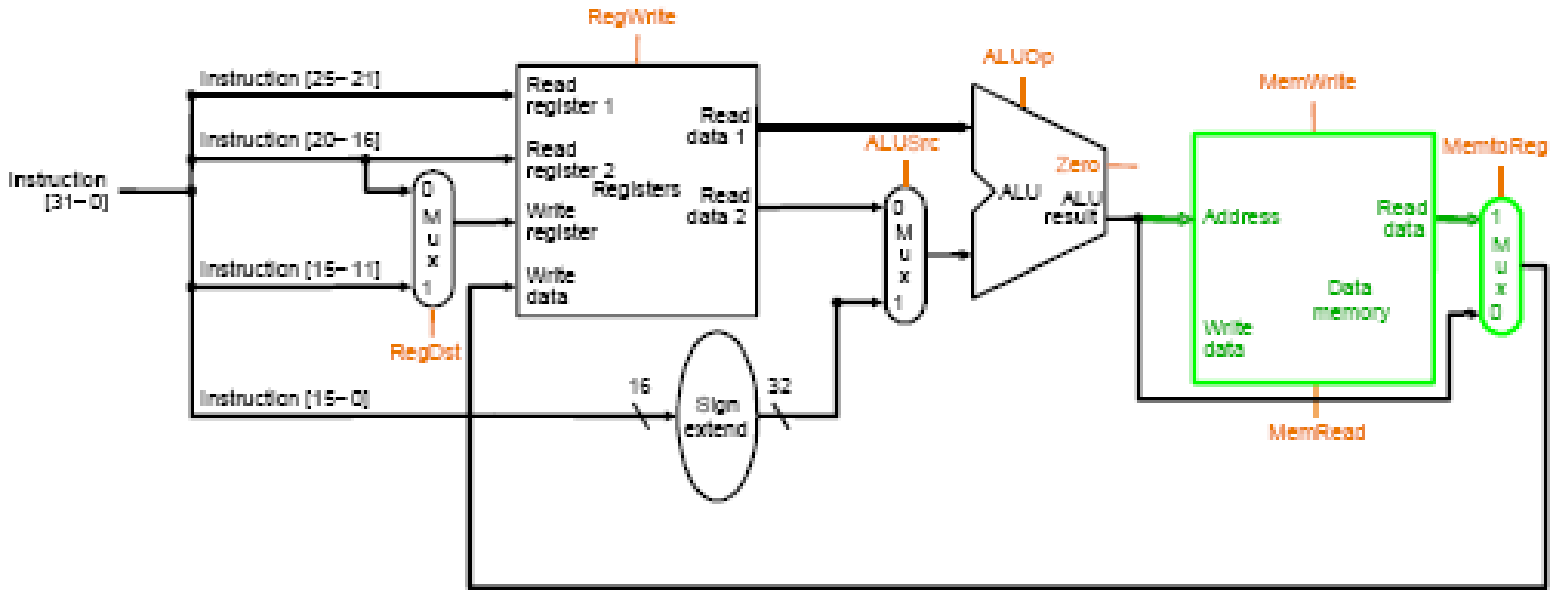
LOAD指令

- LOAD 指令
  - lw rt, rs, imm
  - Addr ← R[rs] + SignExt(imm)　　计算地址
  - R[rt] ← MEM[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的读操作



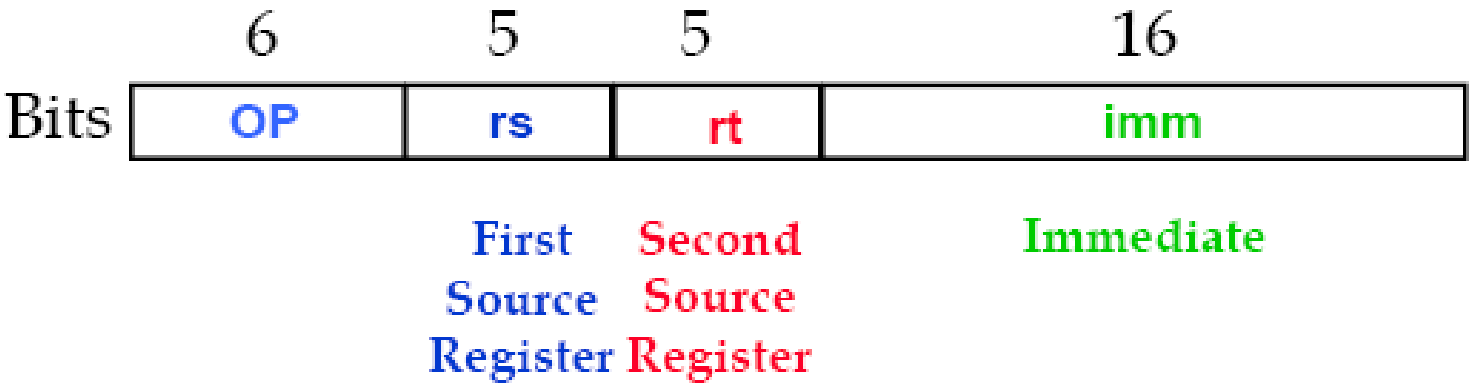
支持Load指令的Datapath

- 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



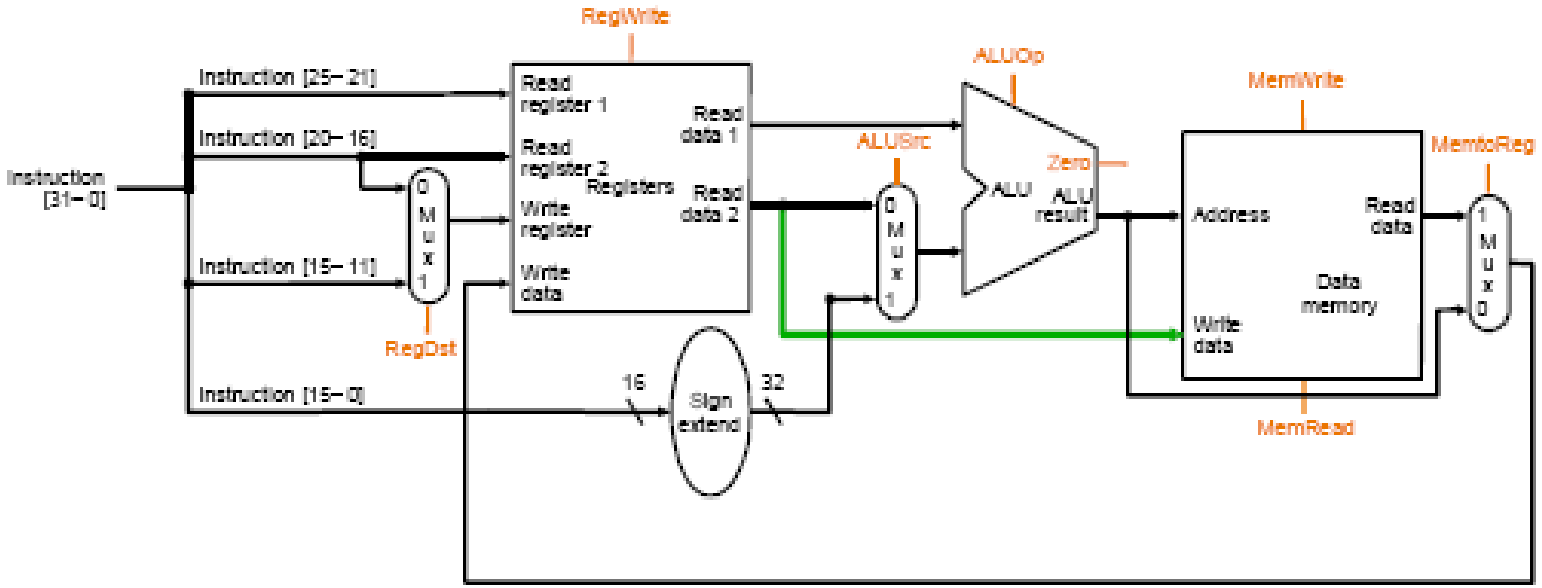
STORE指令

- Store 指令
  - sw rt, rs, imm
  - Addr ← R[rs] + SignExt(imm)　　计算地址
  - MEM[Addr] ← R[rt]　　写数据



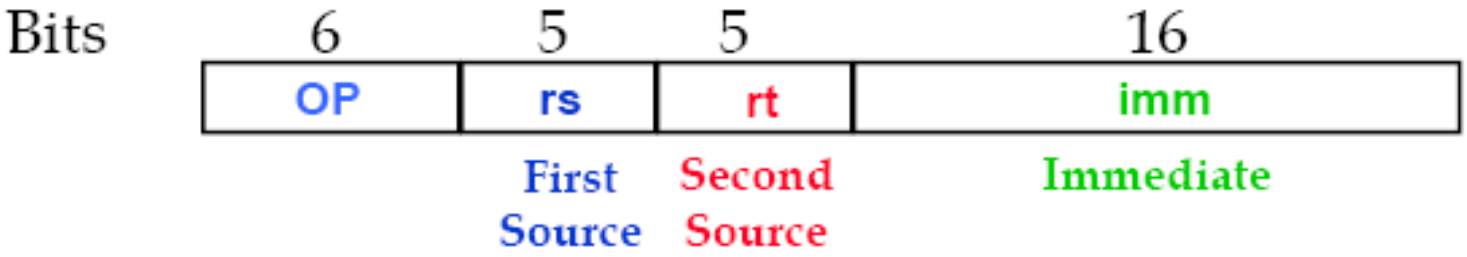
支持Store指令的Datapath

- 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



BEQ指令

- 跳转指令
  - BEQ rs, rt, imm
  - 改变PC的值
  - cond← R[rs] – R[rt]　　计算条件
  - If (cond eq 0) then　　判断条件
  - pc ← pc+4 + SignExt(imm)* 4 else　　计算下一条指令的地址
  - pc ← pc+4

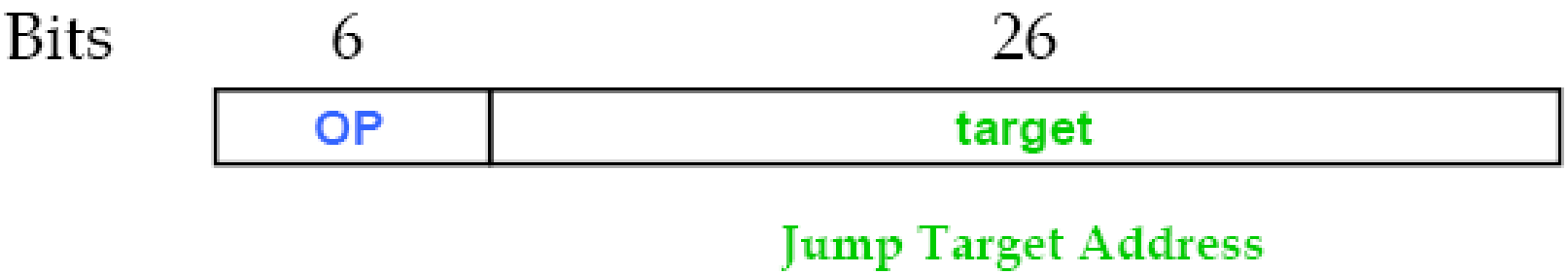


下一条指令的地址

- 指令存储器按字节编址
  - 顺序执行
    - PC←PC+4
  - 跳转
    - PC←PC+4+SignExt(imm)*4

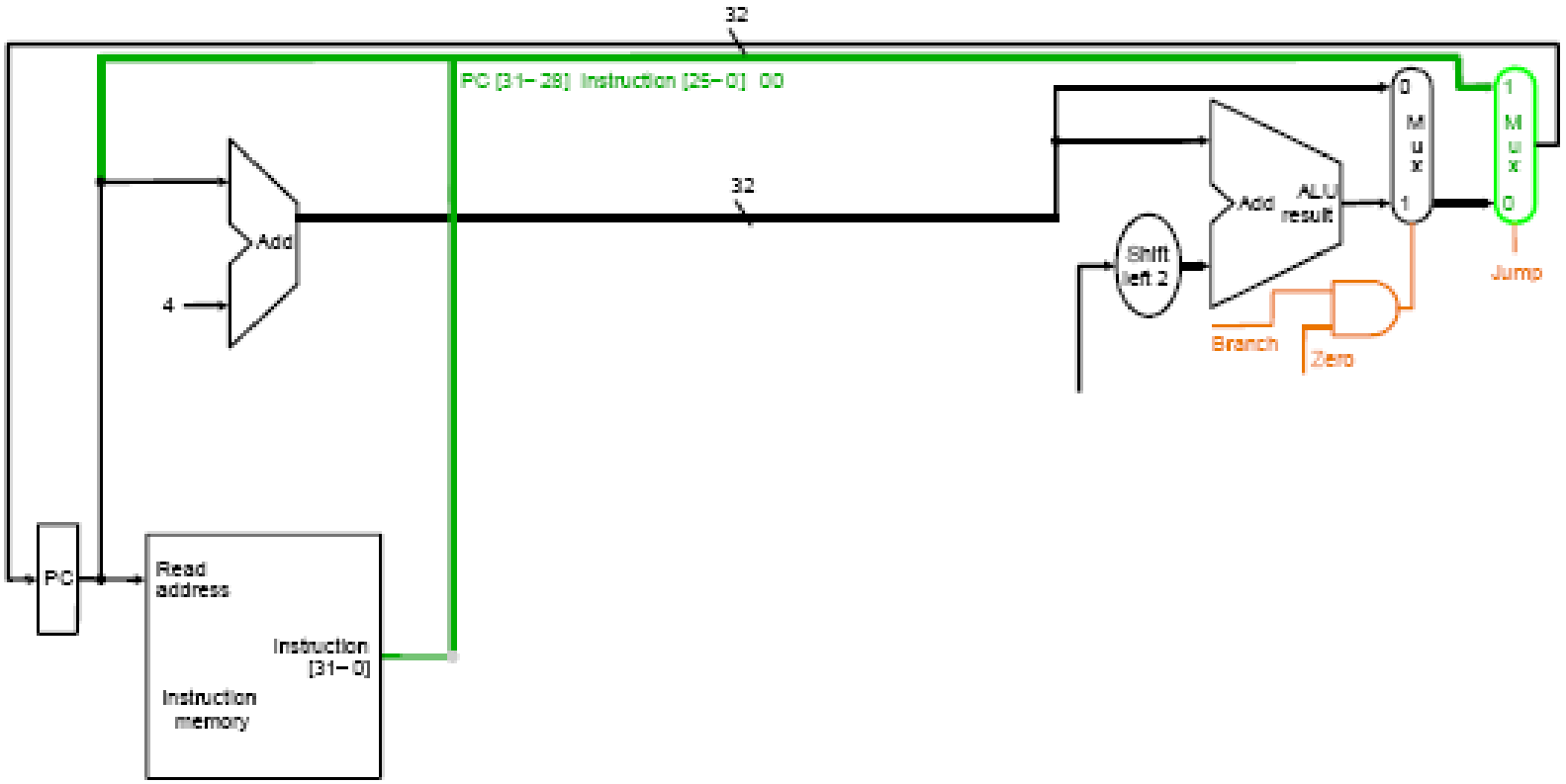
Jump指令

- Jump指令
  - J target
    - PC[31：0]←PC[31：28] || traget[25:0]||00

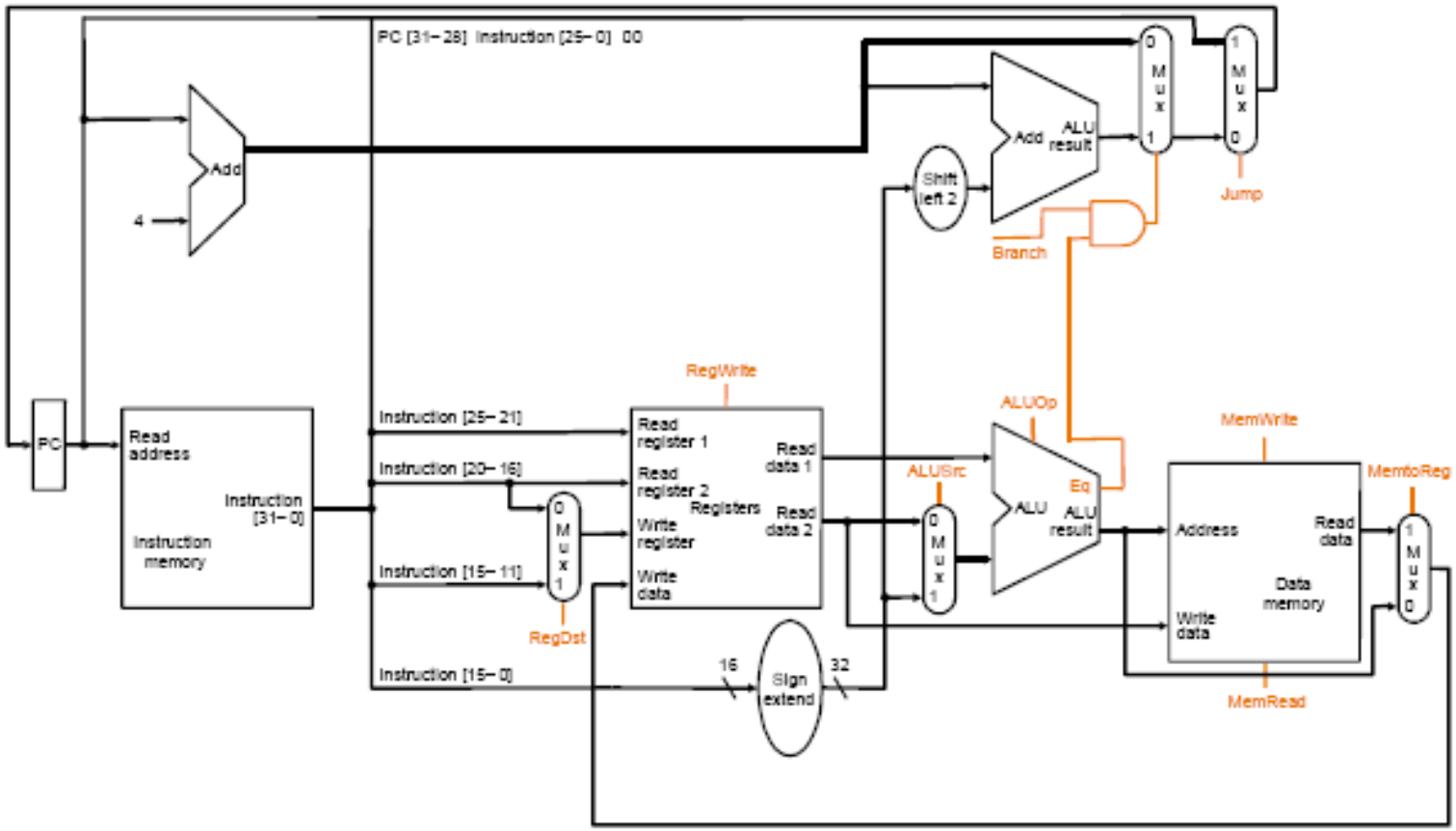


支持BEQ、Jump指令

- 在取指部件增加多路选择器



组装到一起

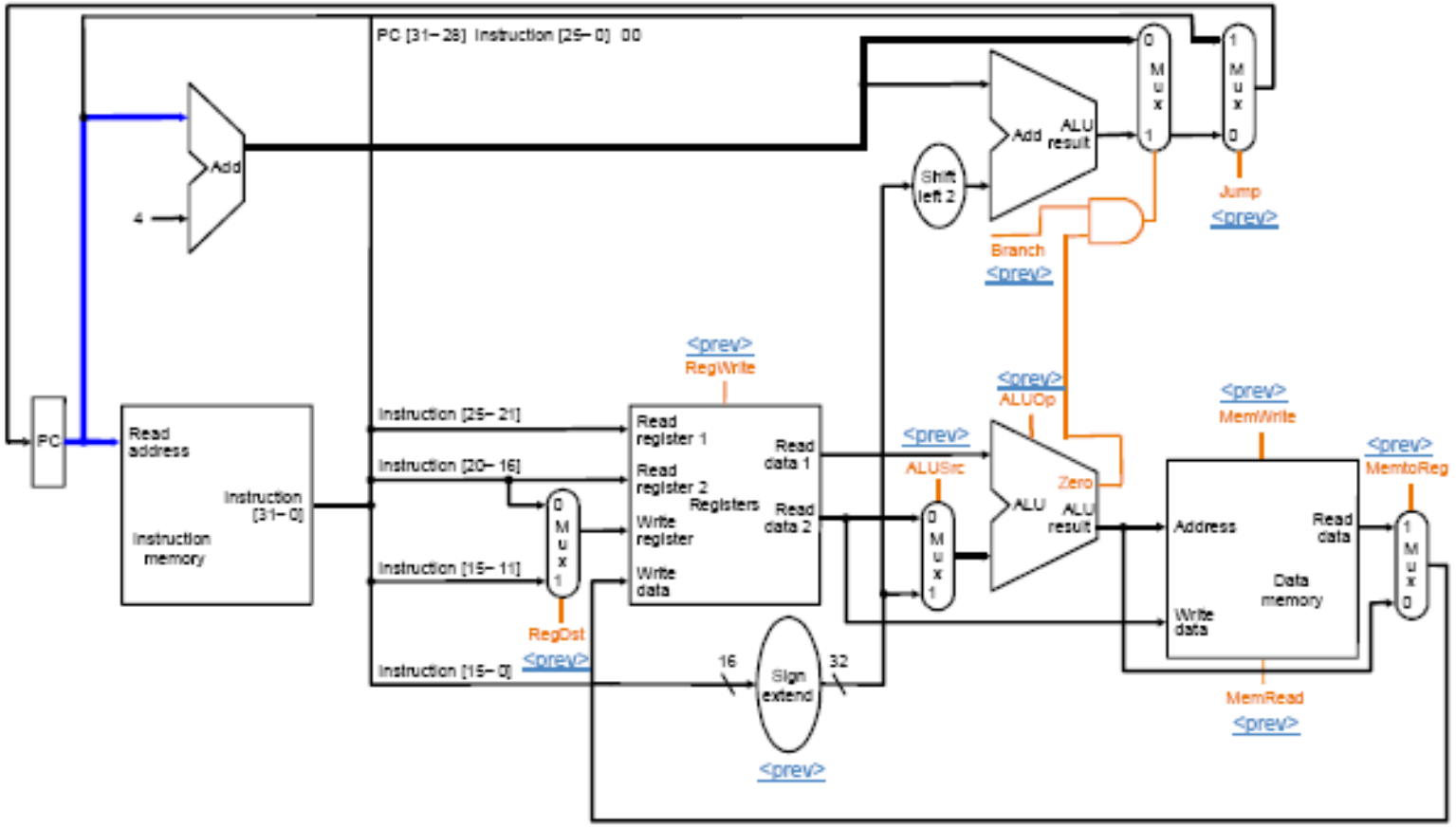


- 所需要的控制信号
  - RegDst 、RegWrite、 Extend、 ALUsrc、 ALUop、 Branch、 Jump MemRead、 MemWrite、 MemtoReg

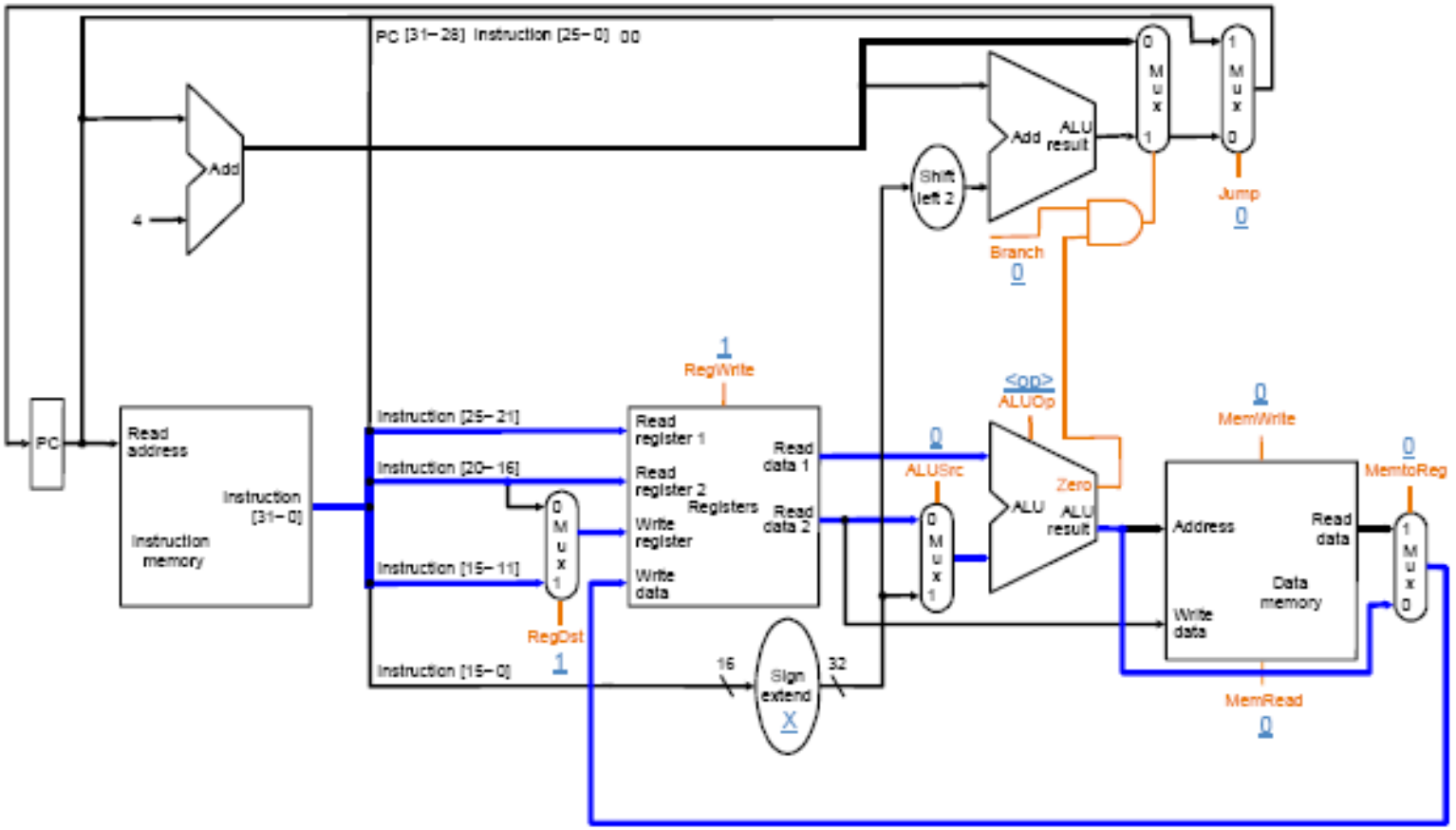
时序设计

- 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

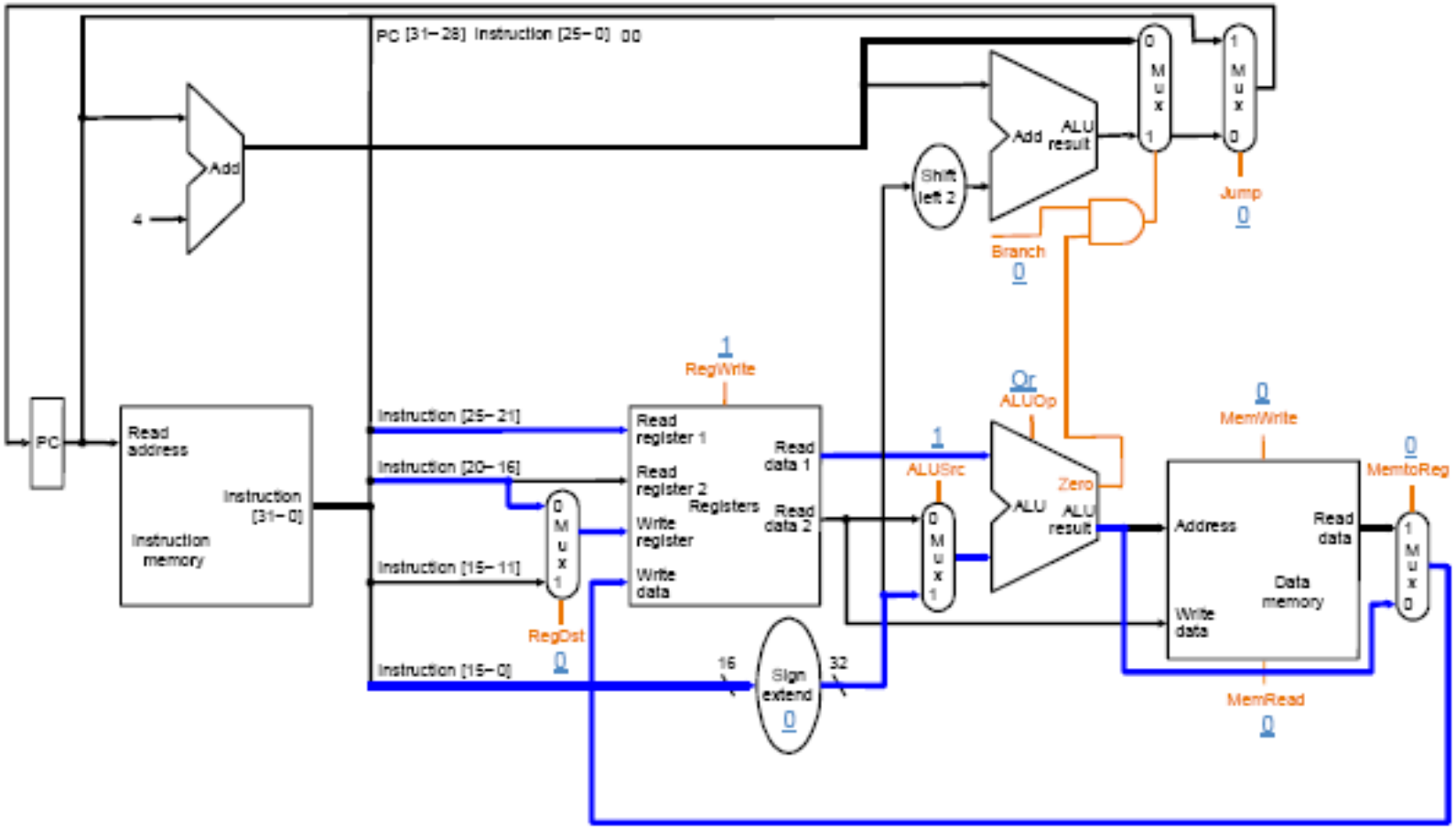
时钟周期开始时的控制


PC [31–28] Instruction [25–0] 00
Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
RegDst
<prev>
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign extend
32
ALUSrc
ALUOp
Zero
ALU
ALU result
Add
Shift left 2
Branch
Jump
MemWrite
MemtoReg
MemRead
Address
Read data
Data memory
Write data
M u x
0
1

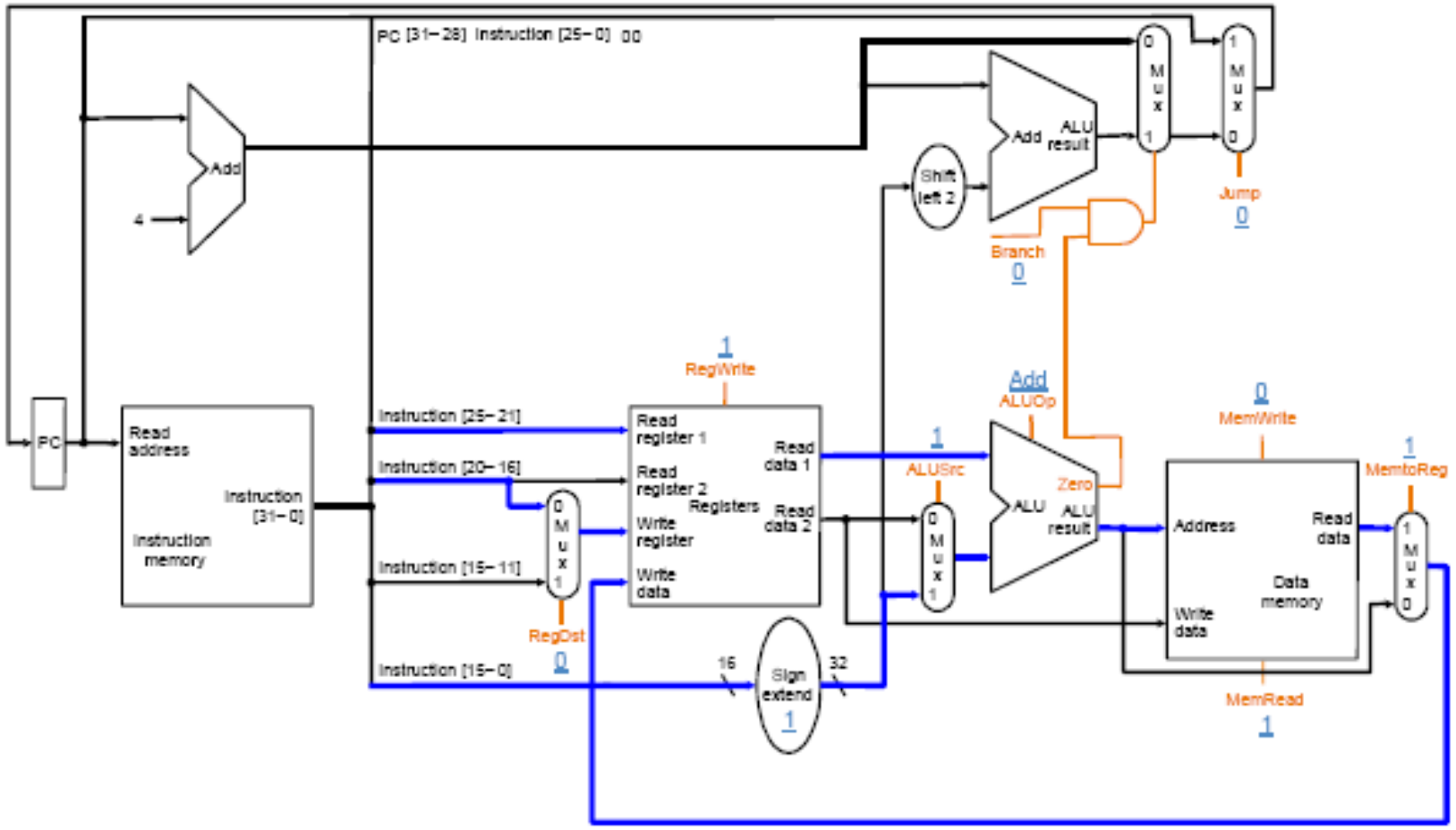
算逻运算指令的控制


PC [31–28] Instruction [25–0] 00
Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
RegDst
1
RegWrite
1
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
ALUSrc
0
<op>
ALUOp
ALU
Zero
ALU result
Add
Shift left 2
Branch
0
Jump
0
MemWrite
0
MemtoReg
0
Address
Read data
Data memory
Write data
MemRead
0
16
Sign extend
32
X
M u x

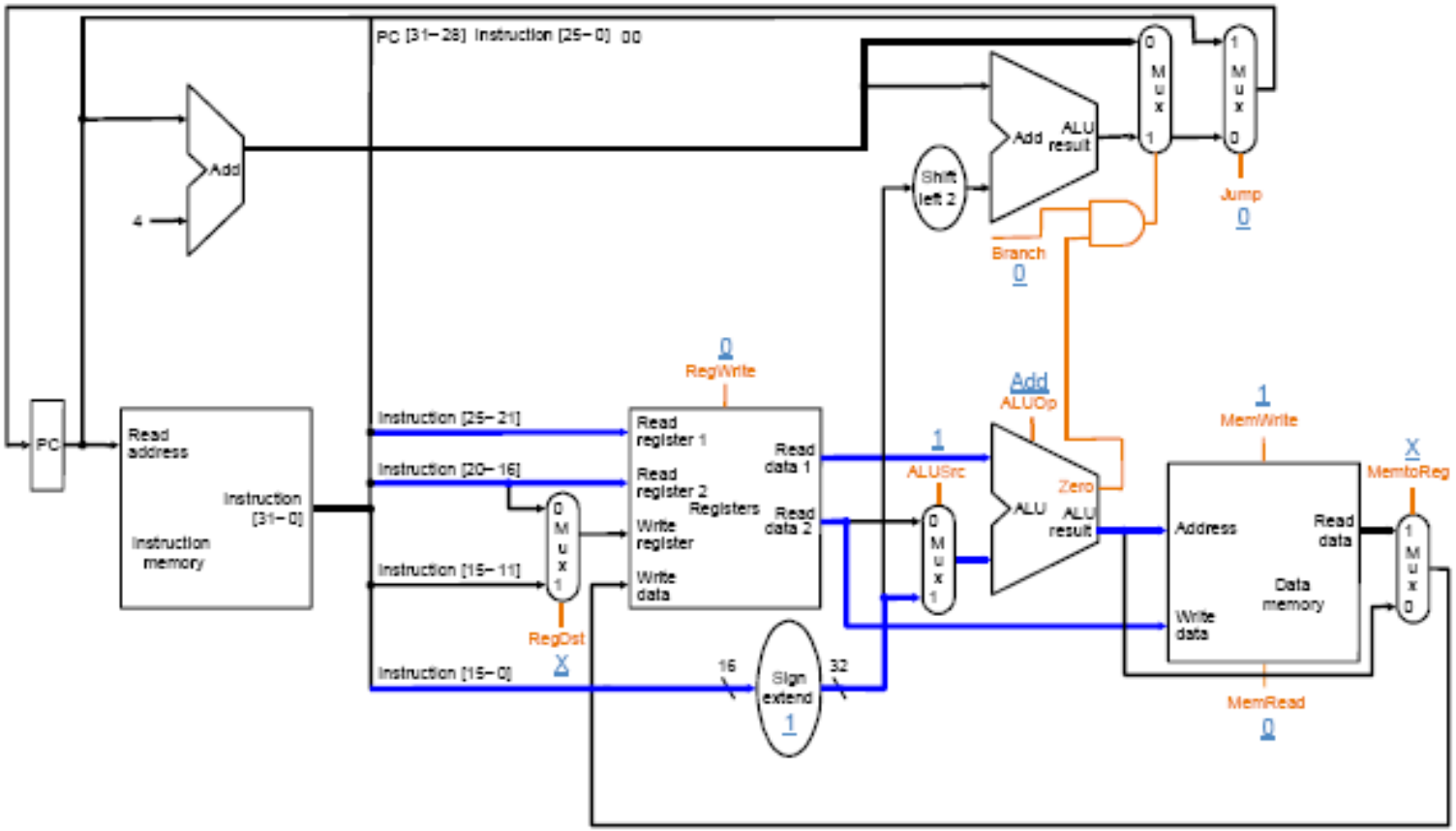
ORI指令的控制


PC [31–28] Instruction [25–0] 00
Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
RegWrite
1
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegDst
0
Sign extend
16
32
0
ALUSrc
1
Add
ALU result
Shift left 2
Branch
0
Jump
0
Or
ALUOp
Zero
ALU
ALU result
MemWrite
0
Address
Read data
Data memory
Write data
MemRead
0
MemtoReg
0
M u x

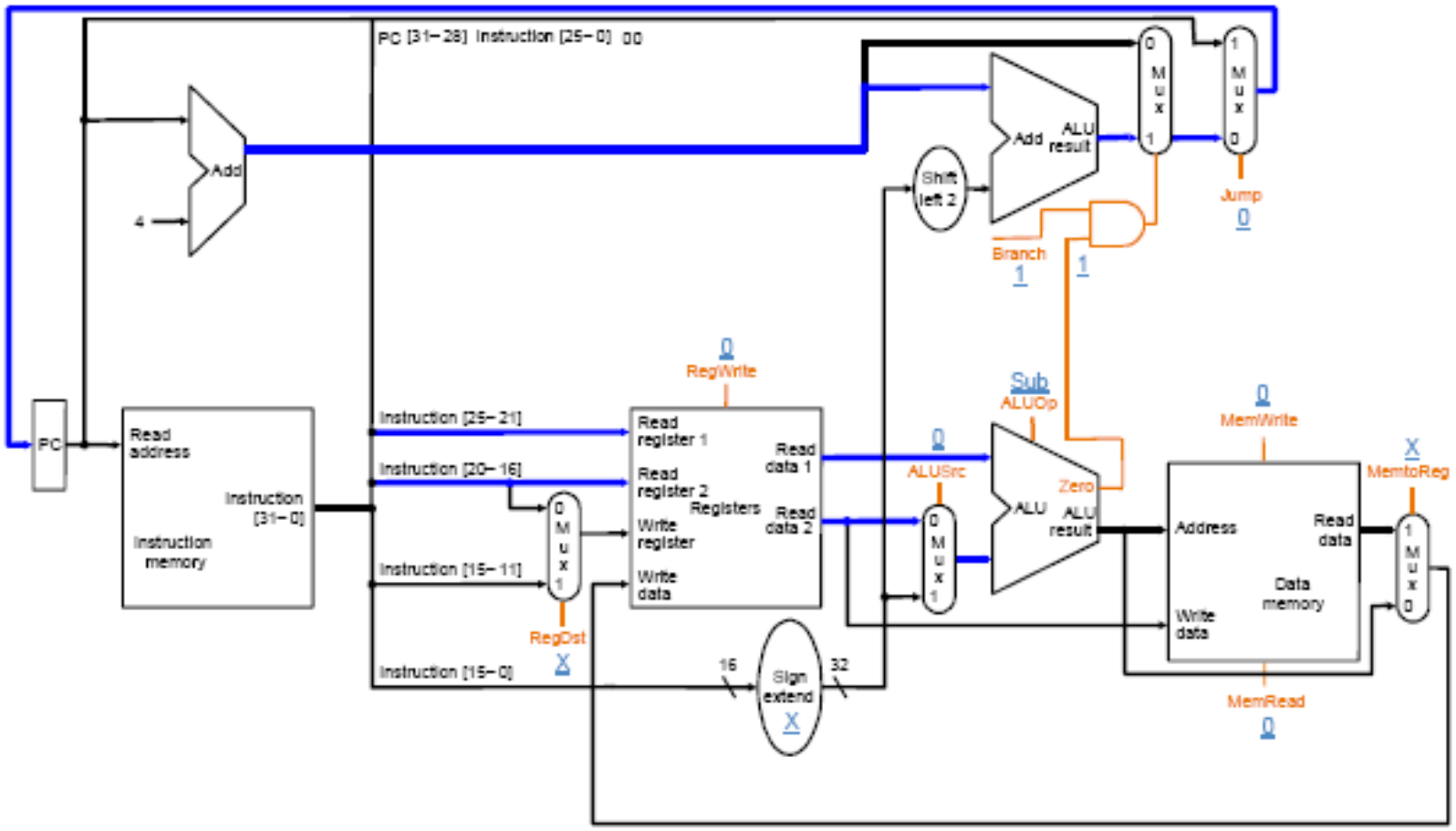
Load指令


PC [31–28] Instruction [25–0] 00
Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
RegWrite
1
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegDst
0
M u x
1
0
16
Sign extend
32
1
ALUSrc
Add
ALUOp
Shift left 2
Add
ALU result
Zero
ALU
Branch
0
Jump
0
MemWrite
0
Address
Read data
Data memory
Write data
MemRead
1
MemtoReg
1

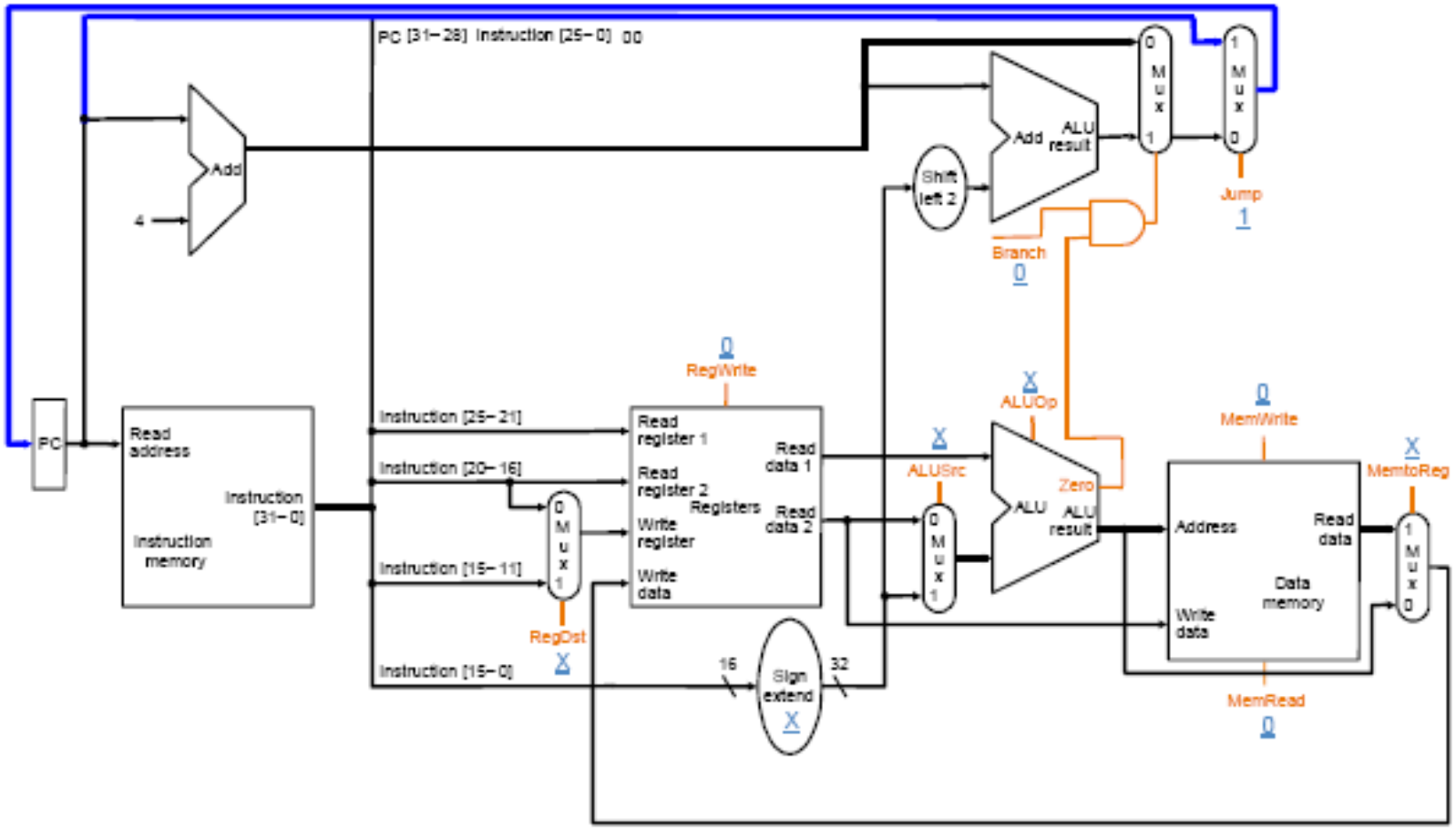
Store指令


PC [31–28] Instruction [25–0] 00
Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
RegWrite
0
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegDst
X
16
Sign extend
32
1
ALUSrc
Add
ALUOp
ALU
Zero
ALU result
Shift left 2
Add
ALU result
Branch
0
Jump
0
MemWrite
1
Address
Read data
Data memory
Write data
MemRead
0
MemtoReg
X
M u x

Beq指令


PC [31–28] Instruction [25–0] 00
Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
0
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
0
M u x
1
RegDst
X
16
Sign extend
32
X
Shift left 2
Add
ALU result
0
M u x
1
Branch
1
1
1
M u x
0
Jump
0
Sub
ALUOp
0
ALUSrc
0
M u x
1
ALU
Zero
ALU result
0
MemWrite
Address
Read data
Data memory
Write data
X
MemtoReg
1
M u x
0
MemRead
0

Jump指令


PC [31–28] Instruction [25–0] 00
Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
RegWrite
0
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegDst
X
16
Sign extend
32
ALUSrc
X
ALUOp
X
Add
ALU result
Shift left 2
Branch
0
Jump
1
ALU
Zero
ALU result
MemWrite
0
MemRead
0
Address
Read data
Data memory
Write data
MemtoReg
X
M u x

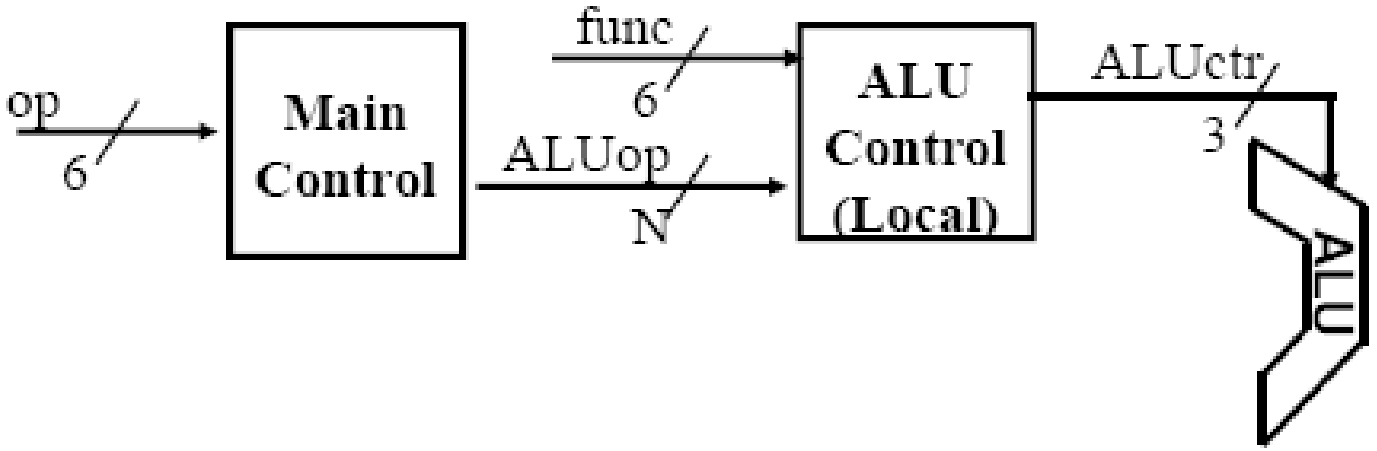
控制信号表

Tsinghua University

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

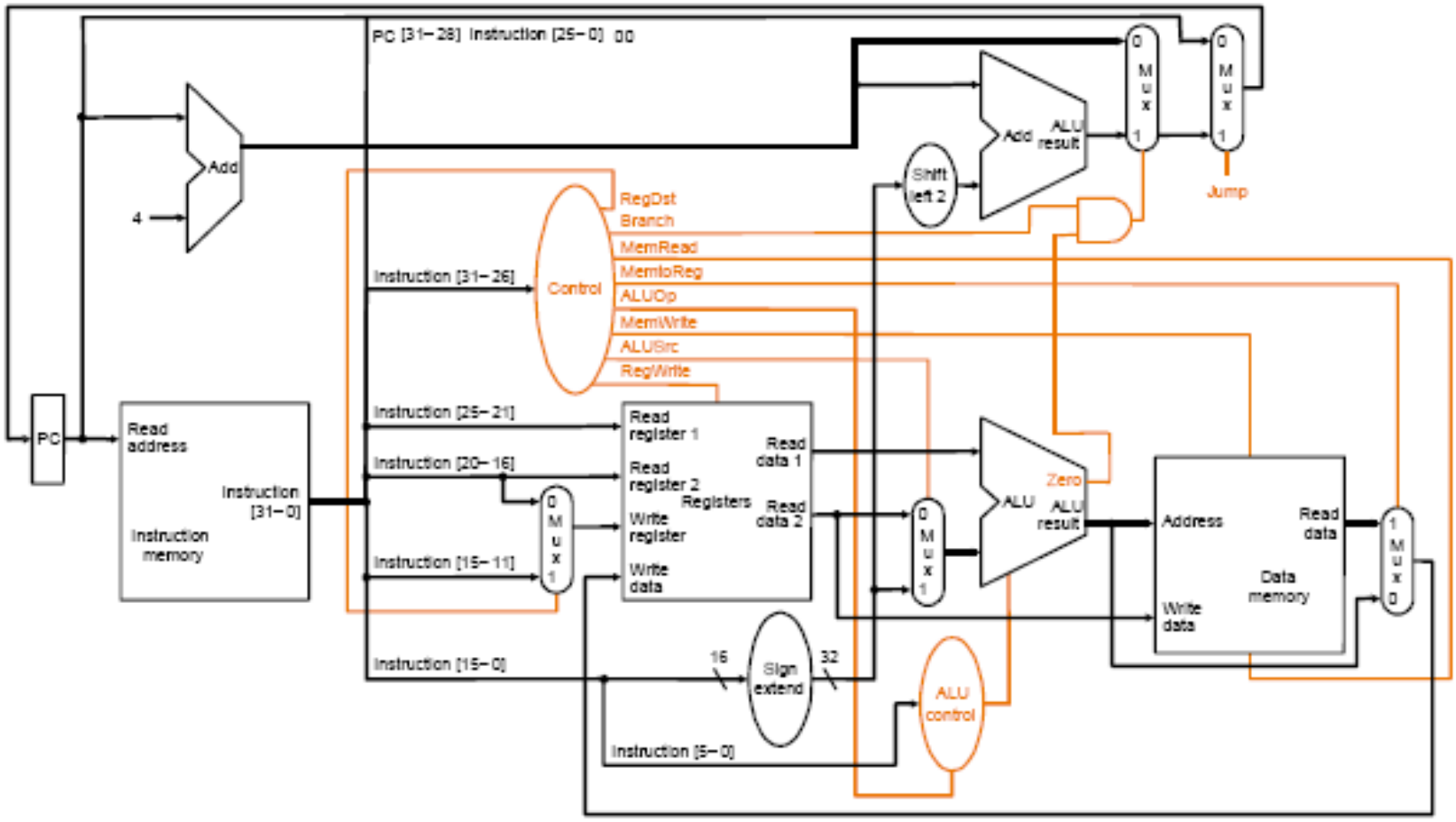


多级控制的信号汇总

清華大學 Tsinghua University

op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	R-type	**ori**	**lw**	**sw**	**beq**	**jump**
RegDst	1	0	0	x	x	x
ALUSrc	0	1	1	1	0	x
MemtoReg	0	0	1	x	x	x
RegWrite	1	1	1	0	0	0
MemWrite	0	0	0	1	0	0
Branch	0	0	0	0	1	0
Jump	0	0	0	0	0	1
ExtOp	x	0	1	1	x	x
ALUop<N:0>	"R-type"	Or	Add	Add	Subtrac	xxx

完整的单周期CPU


PC [31–28] Instruction [25–0] 00
Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [31–26]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
0 M u x 1
16
Sign extend
32
Shift left 2
Add
ALU result
Jump
ALU
Zero
ALU control
Address
Read data
Data memory
Write data
1 M u x 0

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
    - 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

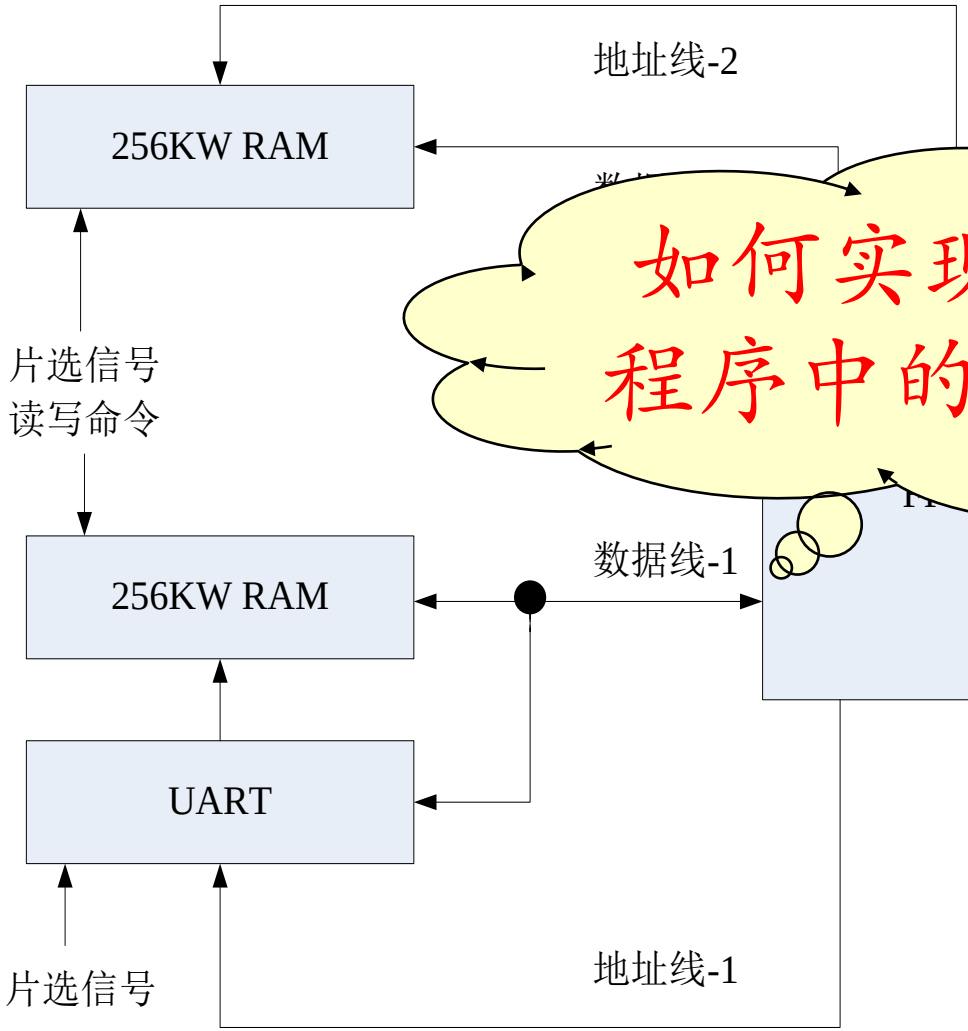
小结

- 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



硬件地址空间：

如何实现与Kernel程序中的地址对应？

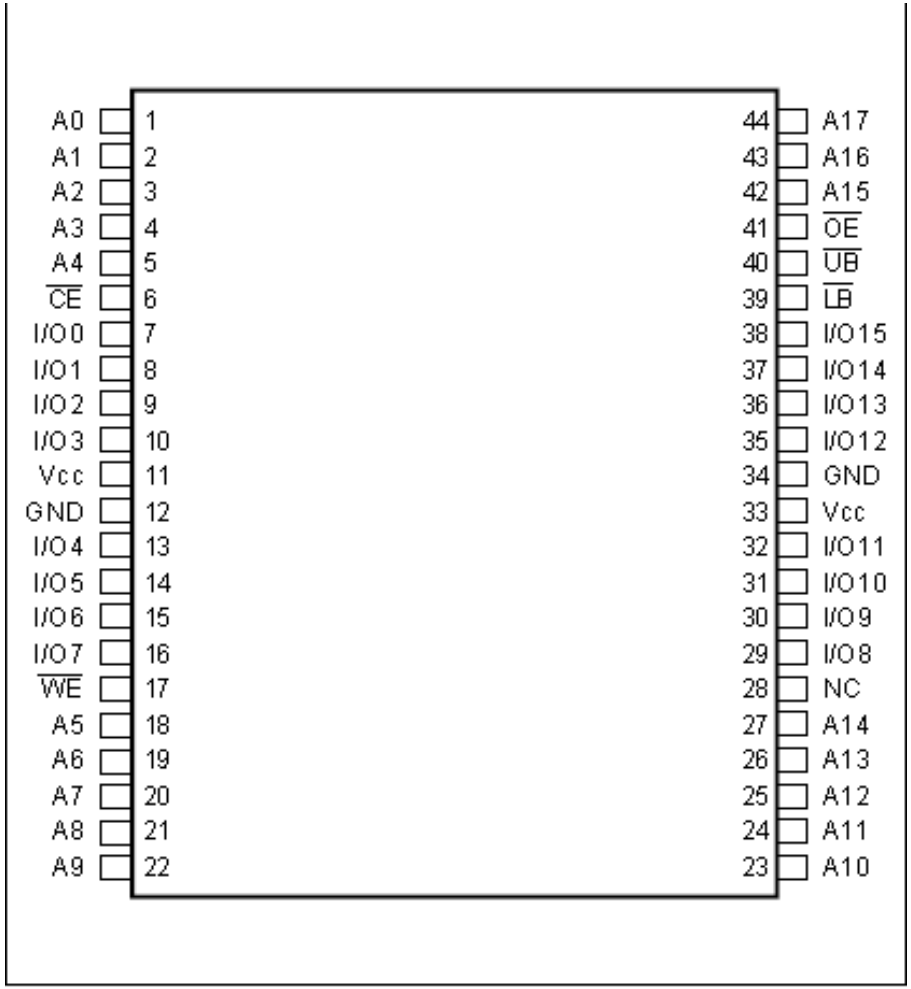
(0)0000~(3)FFFF

串口：UART

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

/EN	/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6日前完成,实验报告截止至11月13日