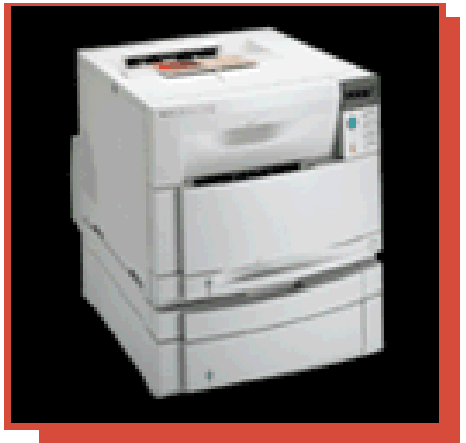
The MIPS Processor Implementation: Datapath & Control

“Computer Organization & Design ”

第四章

MIPS Processor

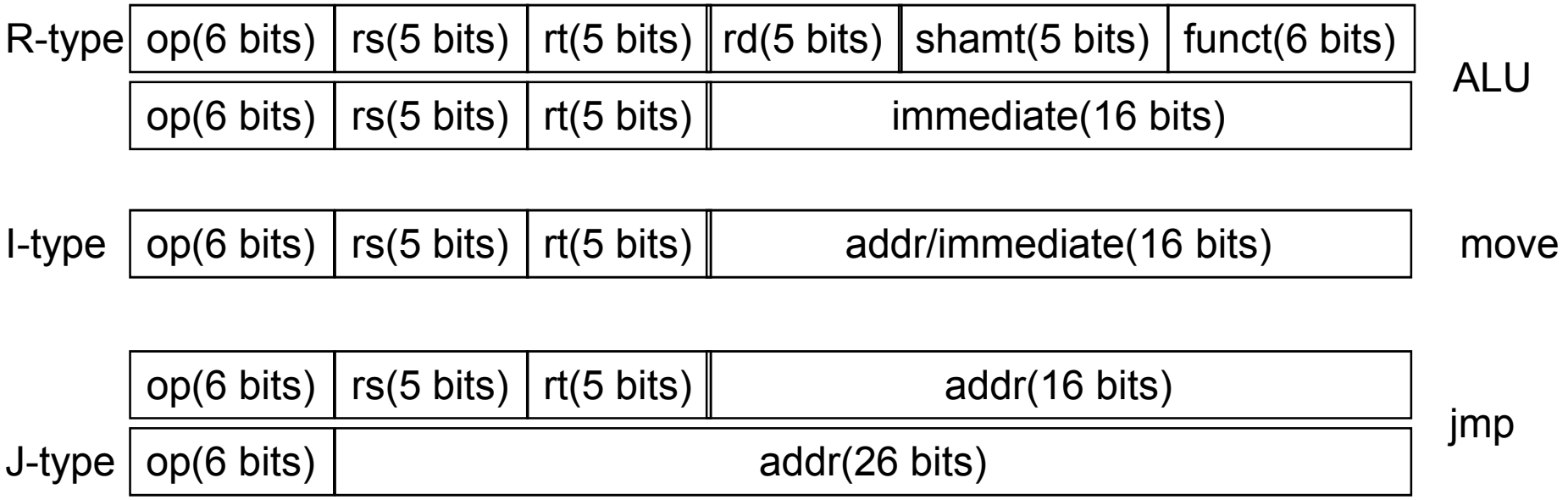
- MIPS：无互锁流水级的微处理器 (Microprocessor without Interlocked Piped Stages)
  - interlock单元：检测RAW相关，推迟后续指令执行（互锁状态）
  - 尽量利用软件办法避免流水线中的数据依赖问题
    - R4000以后开始使用interlock
  - 设计哲学：MIPS is simple, elegant.
    - Simplicity favors regularity！
- 1983年，John Hennessy在Stanford大学成功地完成了第一个采用RISC理念的MIPS微处理器。



Most HP Laserjet workgroup printers are driven by MIPS-based™ 64-bit processors.

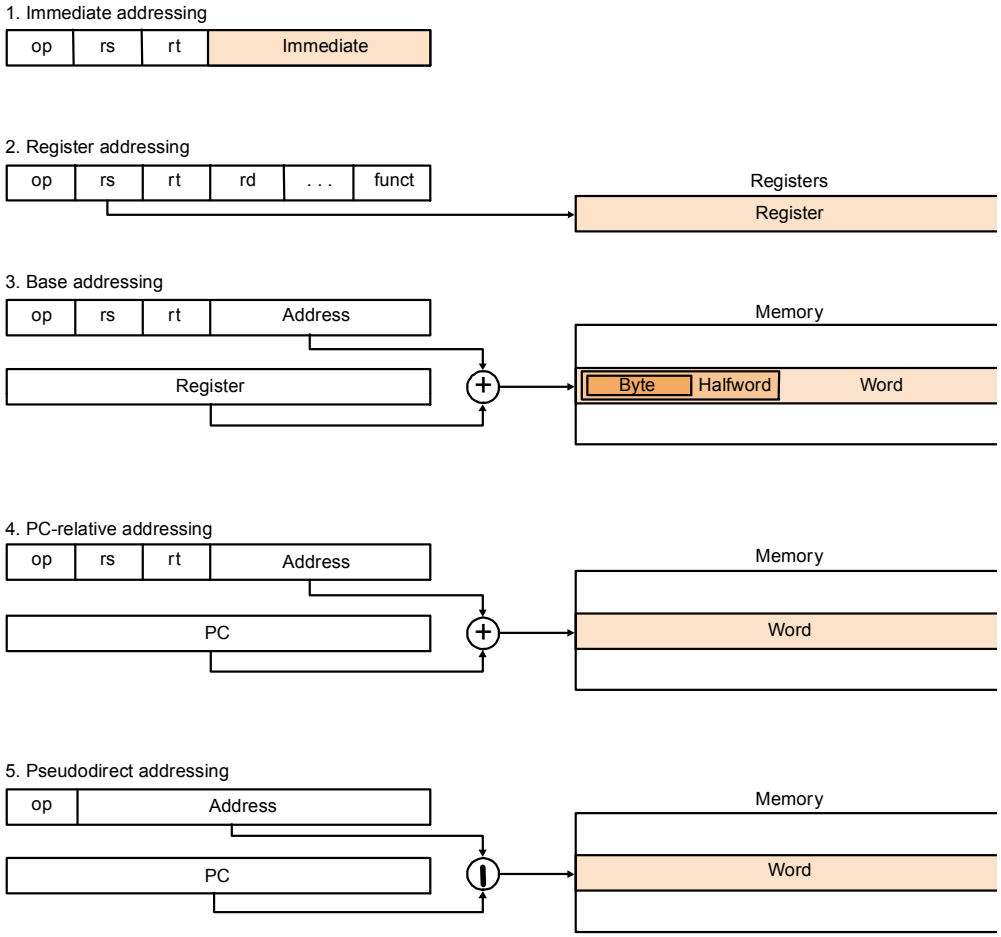
MIPS指令字格式

- 100余条指令（P&H 33条），32个通用寄存器
- 指令格式：定长32位
  - R-type：arithmetic instruction
  - I-type：data transfer, arithmetic instruction（如addi）
  - J-type：branch instruction(conditional & unconditional)



MIPS寻址模式

- 寄存器寻址：R-type
- 基址寻址：I-type
- 立即寻址： I-type
- 相对寻址：J-type
- 伪直接寻址：J-type
  - pseudodirect addressing
  - 26位形式地址左移2位，与PC的高4位拼接



Policy of Use Conventions for registers

REGISTER	NAME	USAGE
$0	$zero	常量0(constant value 0)
$1	$at	保留给汇编器(Reserved for assembler)
$2-$3	$v0-$v1	函数调用返回值(values for results and expression evaluation)
$4-$7	$a0-$a3	函数调用参数(arguments)
$8-$15	$t0-$t7	暂时的(或随便用的)
$16-$23	$s0-$s7	保存的(或如果用，需要SAVE/RESTORE的)(saved)
$24-$25	$t8-$t9	暂时的(或随便用的)
$28	$gp	全局指针(Global Pointer)
$29	$sp	堆栈指针(Stack Pointer)
$30	$fp	帧指针(Frame Pointer)
$31	$ra	返回地址(return address)

- $26..$27：？

A subset of core MIPS instruction set

- Arithmetic-logical instruction（R-type）： add, sub, and, or
  - add $t1, $t2, $t3; $t2+$t3->$t1
  - slt $s1, $s2, $s3（ if($s2<$s3)then $s1=1, else $s1=0 ）
    - 注意：汇编指令中源操作数和目的操作数的排列顺序与机器指令字不同！

R-type

op(6 bits)	rs(5 bits)	rt(5 bits)	rd(5 bits)	shamt(5 bits)	funct(6 bits)

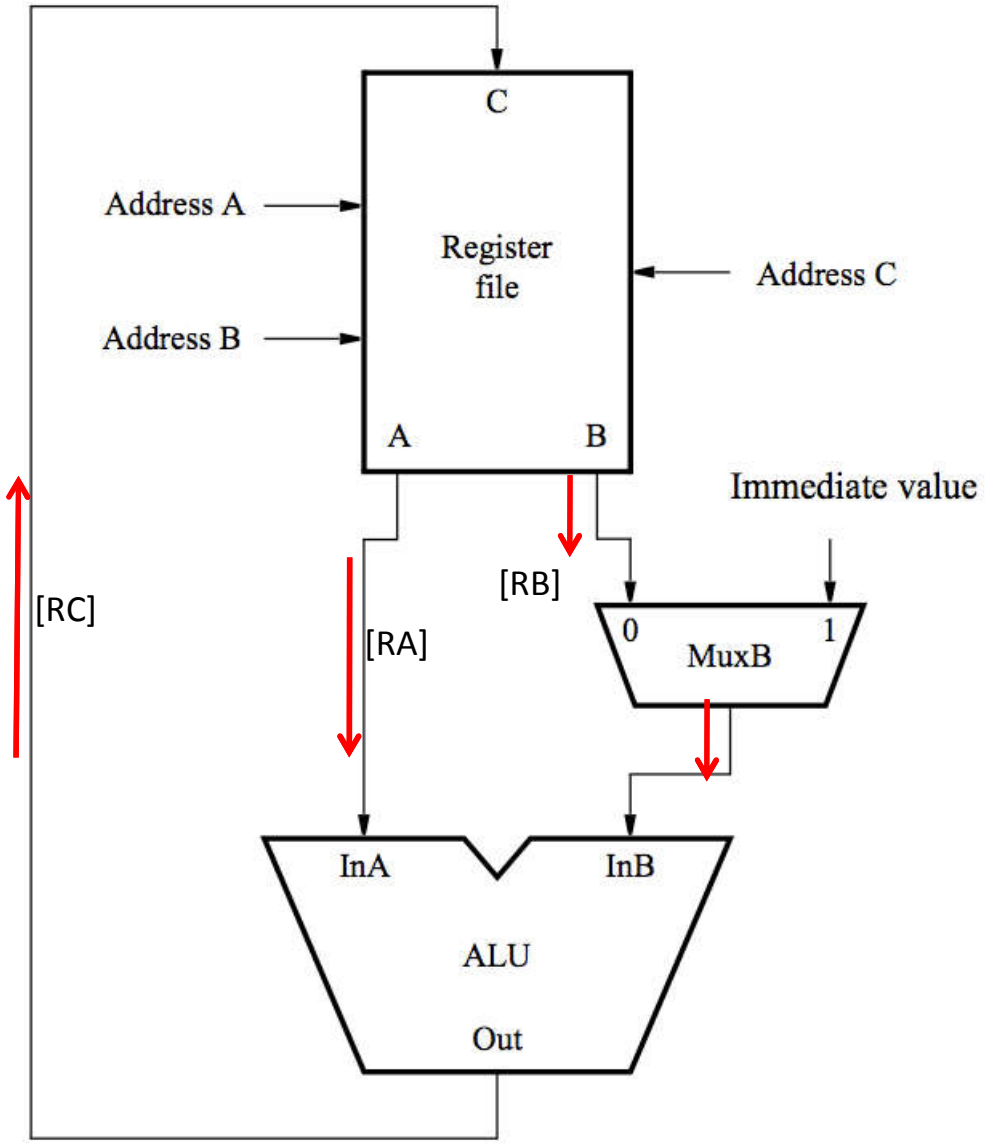
- Memory-reference instruction （I-type）： lw, sw
  - lw $s1,100($s2) ； loads words, based $s2(rs)
  - sw $s1,100($s2)； stores words, based $s2(rs)

I-type

op(6 bits)	rs(5 bits)	rt(5 bits)	addr (16 bits)

A conceptual view – computational instructions

- Both source operands and the destination location are in the register file.



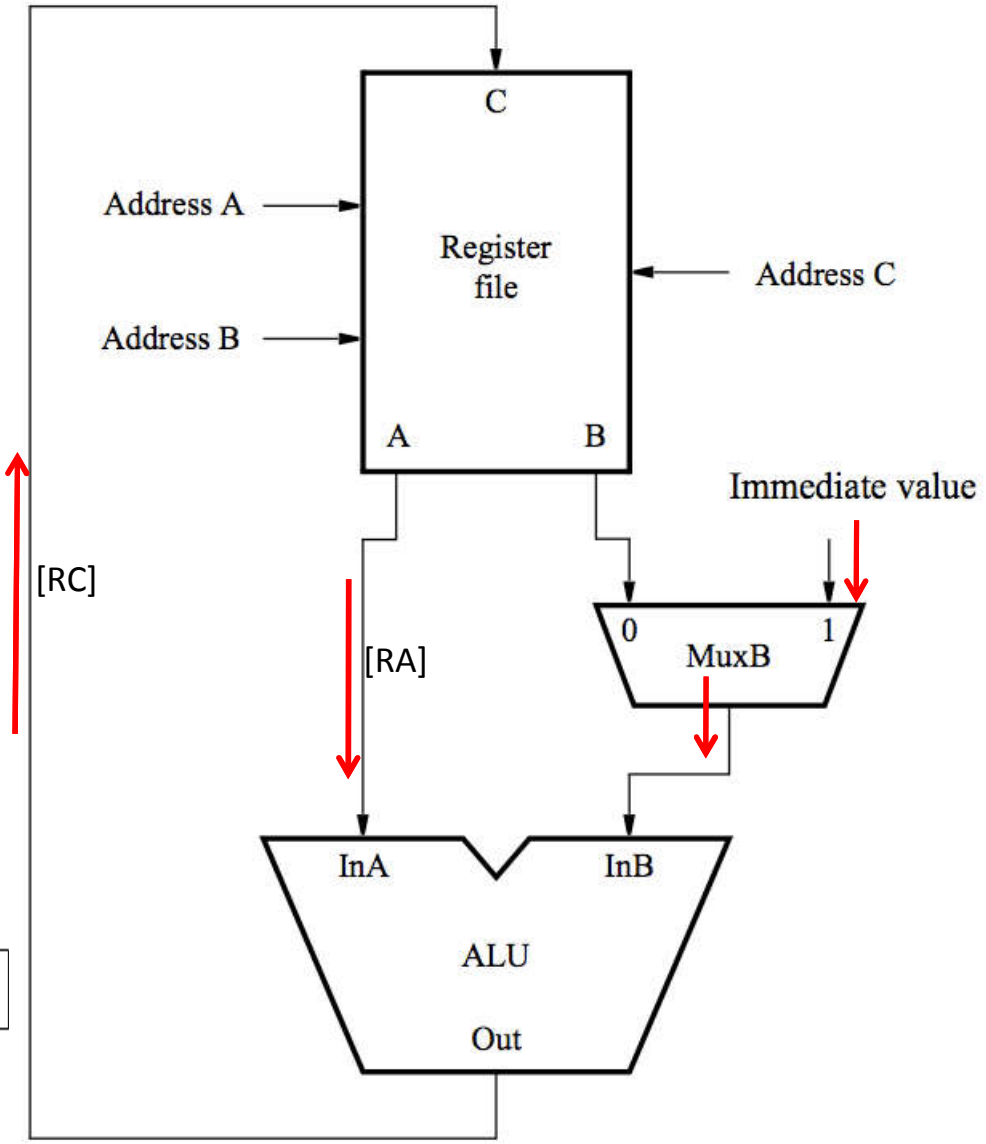
add $t1, $t2, $t3; $t2+$t3->$t1

R-type

op(6 bits)	rs(5 bits)	rt(5 bits)	rd(5 bits)	shamt(5 bits)	funct(6 bits)

A conceptual view – immediate instructions

- One of the source operands is the immediate value in the IR.



addi $s3,$s3,4； $s3 = $s3 + 4

I-type	op(6 bits)	rs(5 bits)	rt(5 bits)	immediate(16 bits)

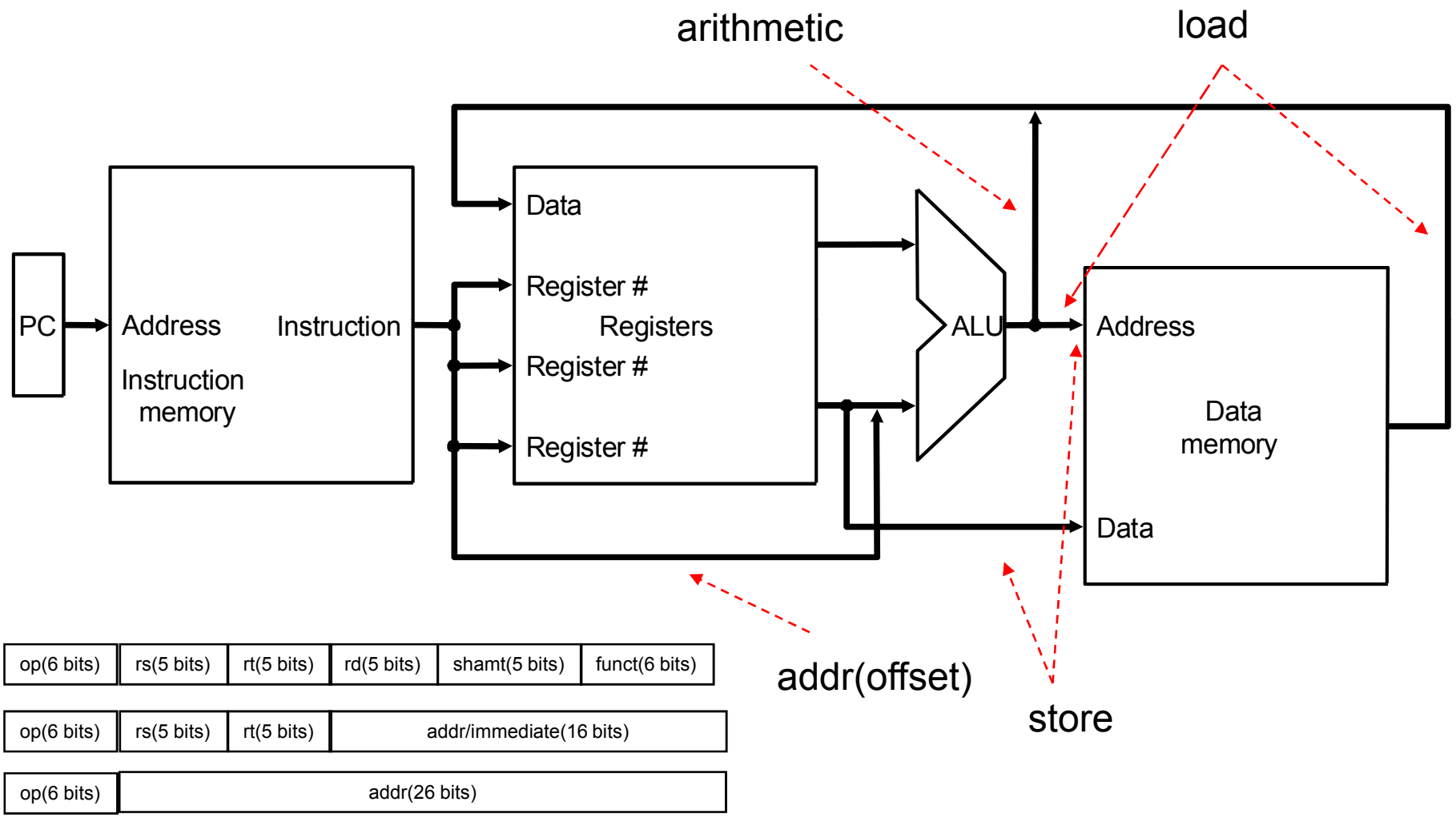
A subset of core MIPS instruction set(con't)

- Branch instruction （J-type）: beq, jump
  - beq为相对寻址：以npc为基准，指令中的target为16位，进行32位有符号扩展后左移两位（补“00”，字对准）。
  - jump为pseudodirect：指令中的target为26位，而PC为32位。将target左移2位拼装在PC的低28位上，PC高4位保持不变。

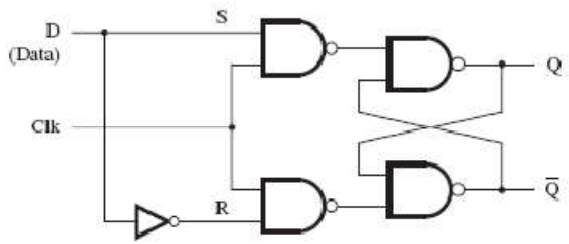
MIPS指令数据通路Overview

- 实现不同指令的多数工作都是相同的，与指令类型无关
  - 取指：将PC送往MEM
  - 取数：根据指令字中的地址域读寄存器
    - 对于ld/st，只需一个寄存器；其他指令，需要两个寄存器
- 执行操作各个指令不同，但同类指令非常类似
  - 不同类型指令也有相同之处，如都要使用ALU
    - 访存指令使用ALU计算地址
    - 算逻指令使用ALU完成计算
    - 分支指令使用ALU进行条件比较
  - 其后，各个指令的工作就不同了
    - 访存指令对存储器进行读写
    - 算逻指令将ALU结果写回寄存器
    - 分支指令将基于比较结果修改下一条指令的地址

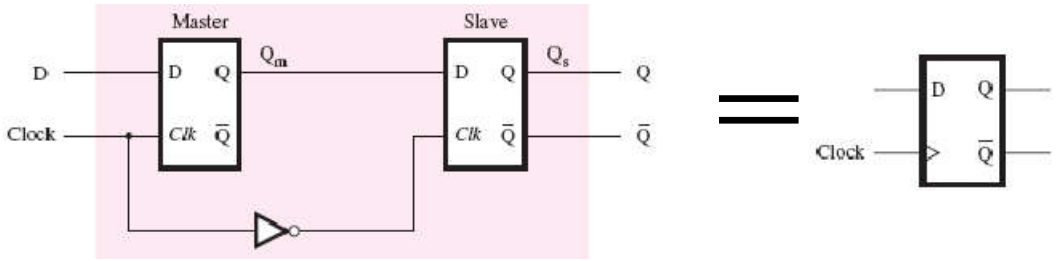
MIPS指令数据通路总图



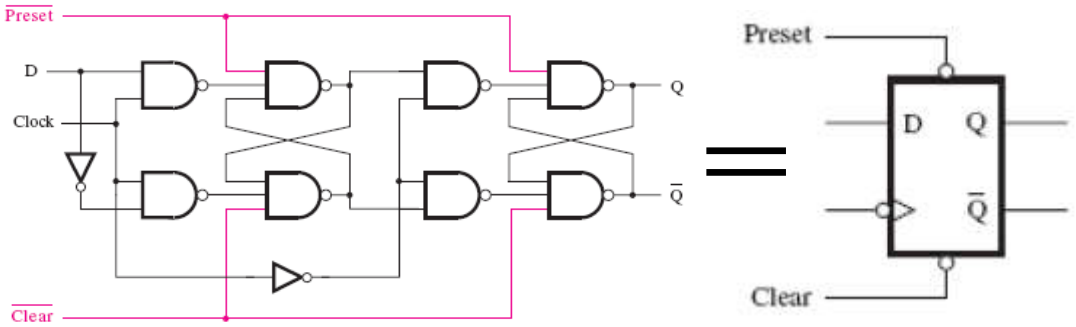
Register



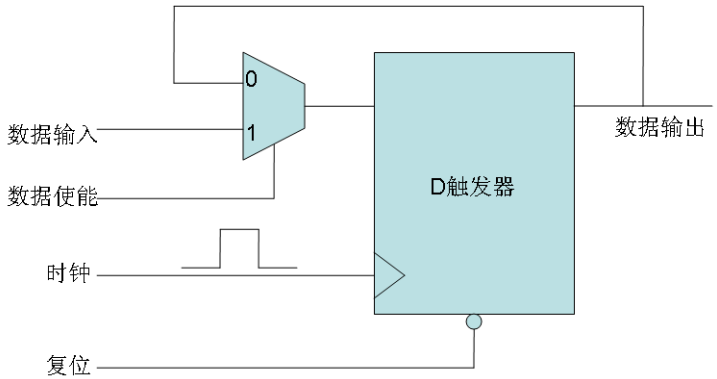
D 锁存器



一位 **D** 触发器

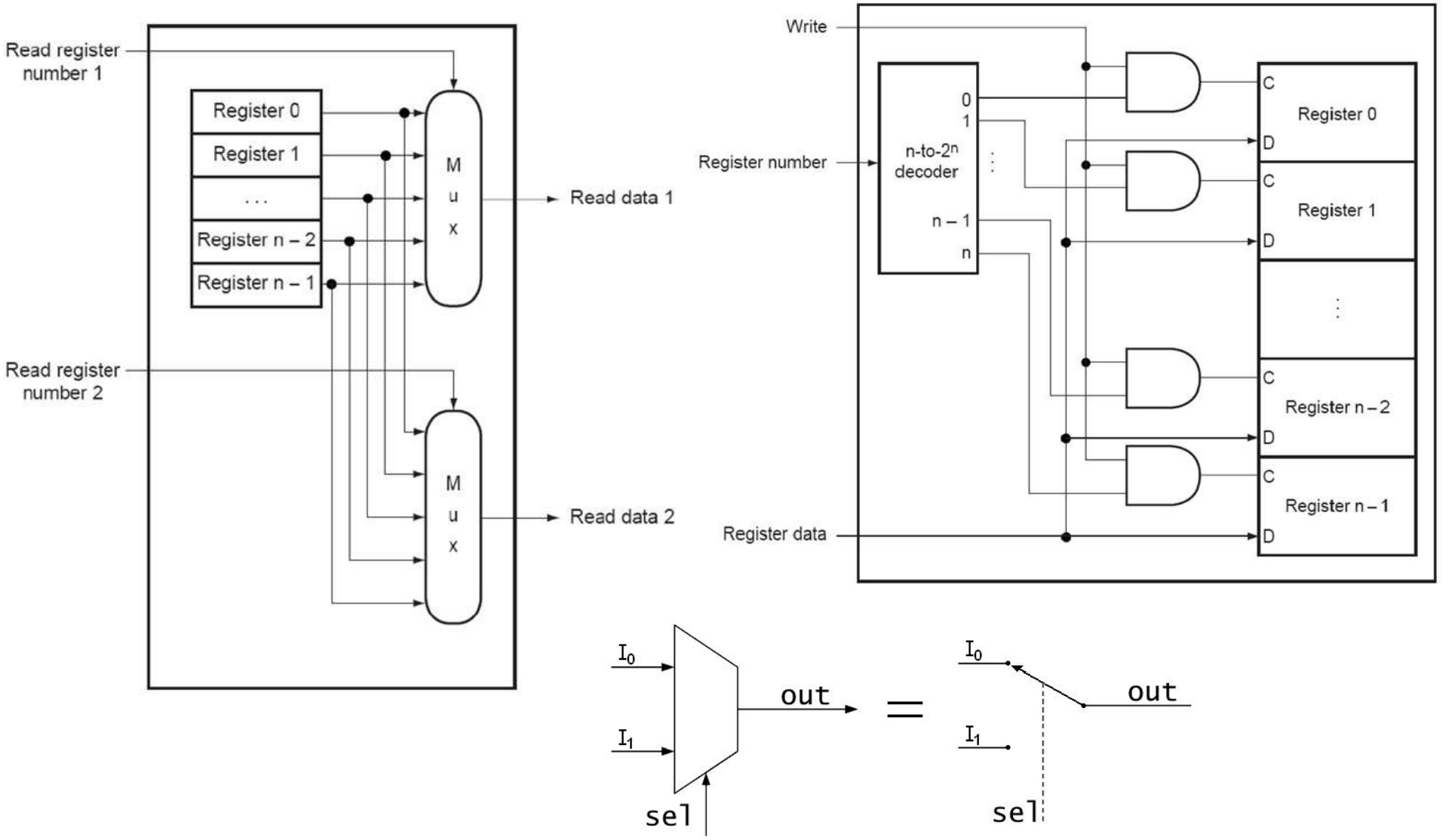


带清零和预置的**D** 触发器



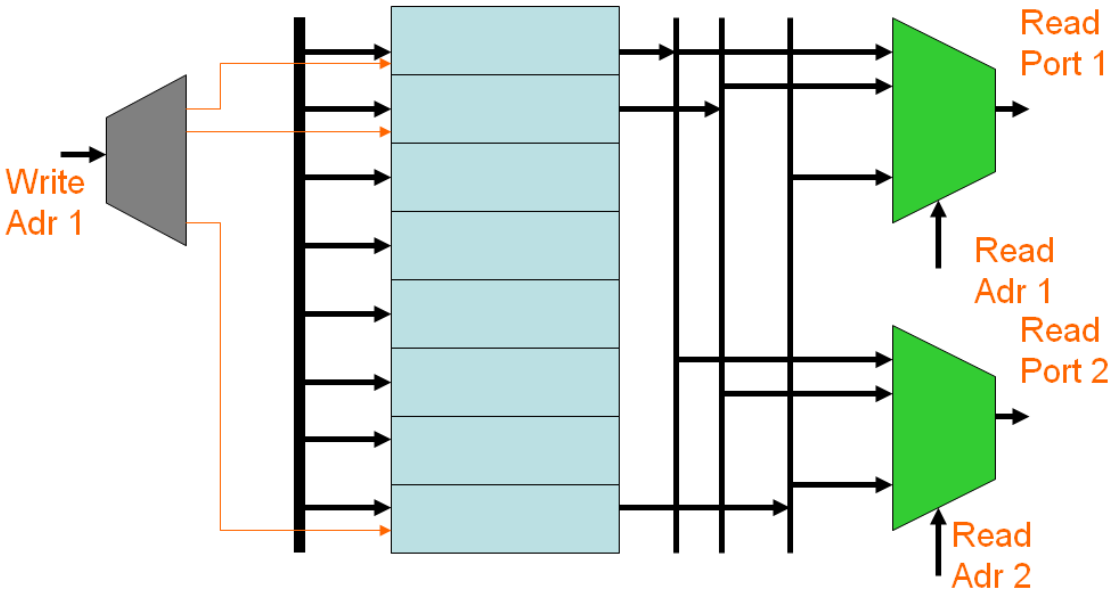
一位寄存器

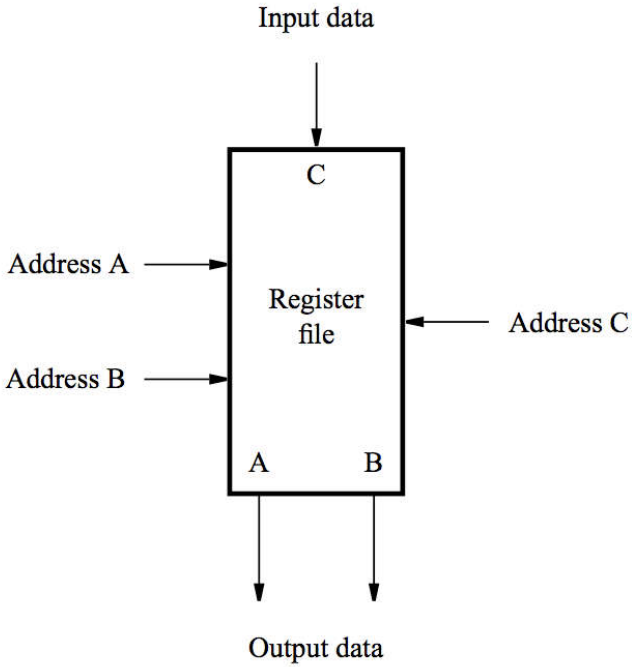
RegisterFile结构



RegFile读写操作

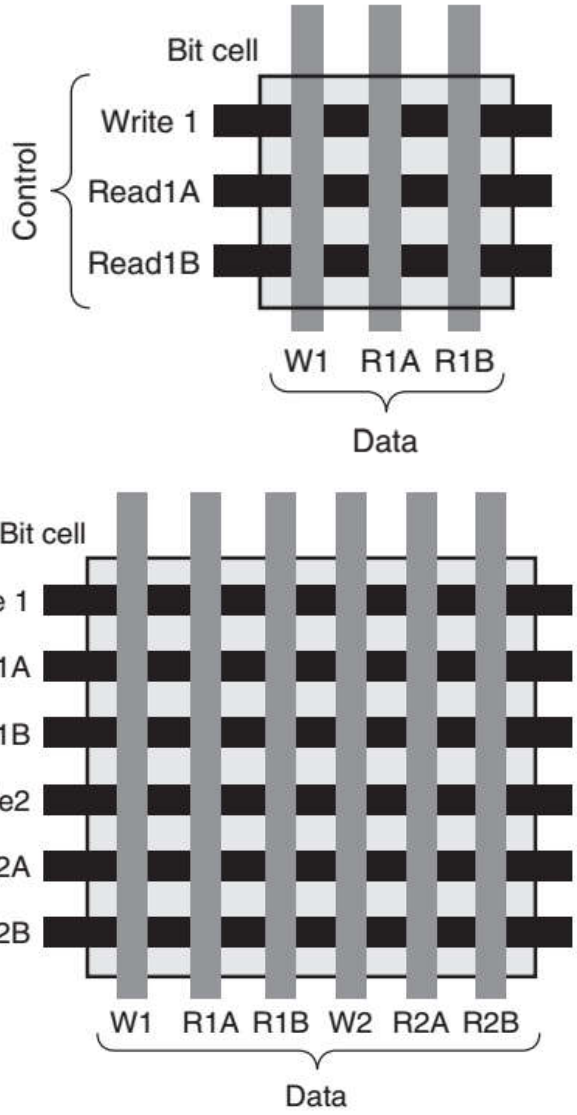
- 寄存器文件在一个周期中有两个读和一个写操作。
  - 在一个周期内，某个REG可以同时完成读写操作，但读出的是上一个周期写入的值
- 显然，寄存器文件不能同时进行读和写操作。两种设计方法：
  - 后写（late write）：在前半周期读数据，在后半周期写数据。
  - 先写（early write）：与后写相反。
  - 写控制信号RegWrite



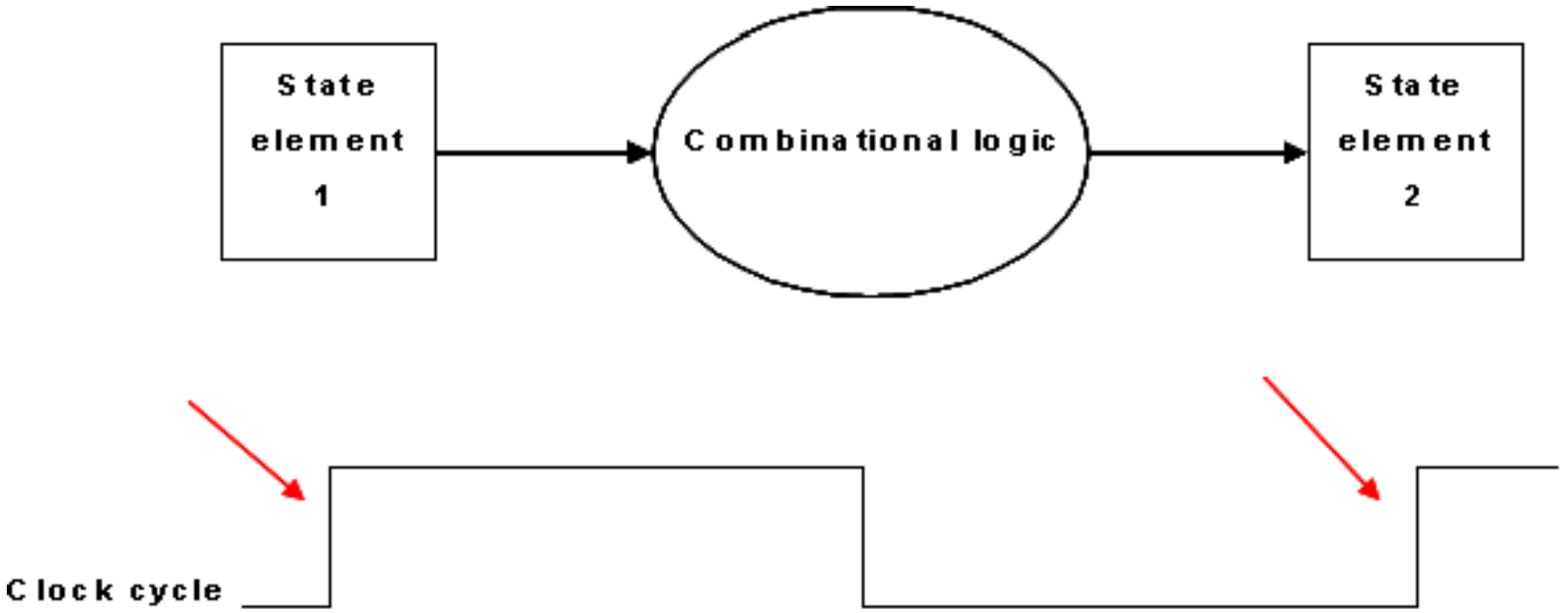


bit cell of a register file

- 例：2R/1W， 4R/2W
  - Register file designs are limited by routing and not by transistor density
  - Read access time of a register file grows approximately linearly with the number of ports

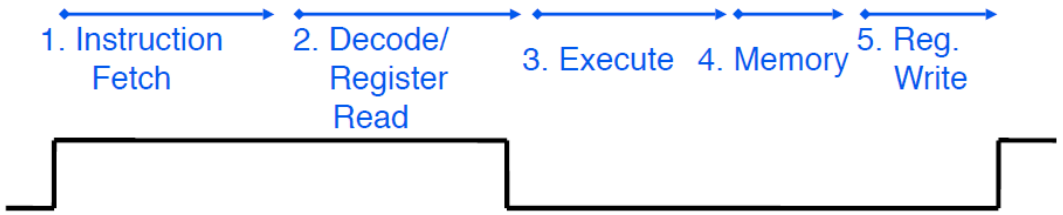


时钟周期

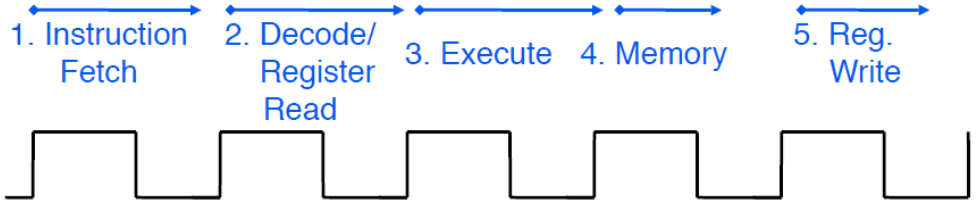

State element 1
Combinational logic
State element 2
Clock cycle

定时模式

- 单周期实现：
  - All stages of an instruction are completed within one ***long*** clock cycle.
  - 所需控制信号同时生成



- 多周期实现：
  - Only one stage of instruction per clock cycle
  - 按机器周期生成当前周期所需控制信号



单周期实现

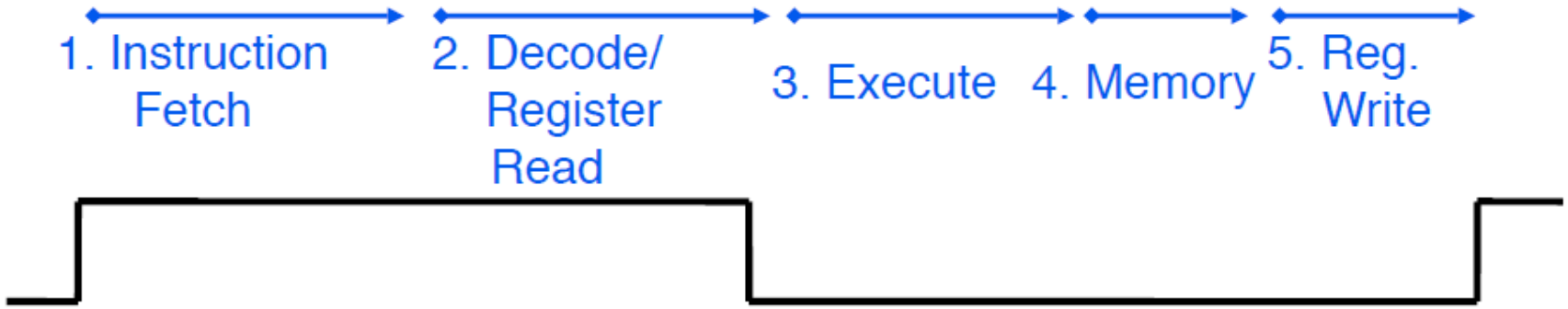
数据通路设计

ALU控制

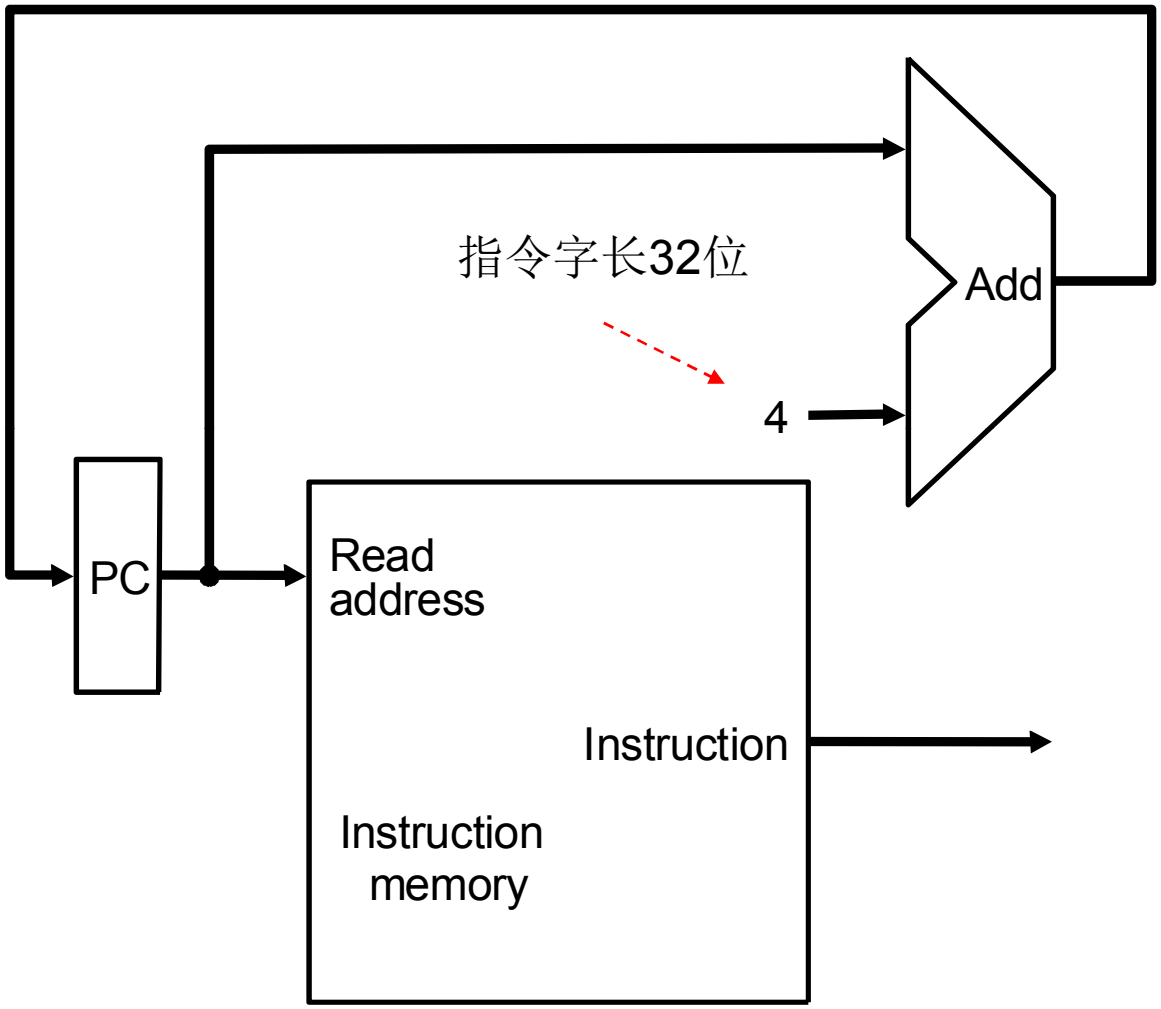
主控制部件

单周期

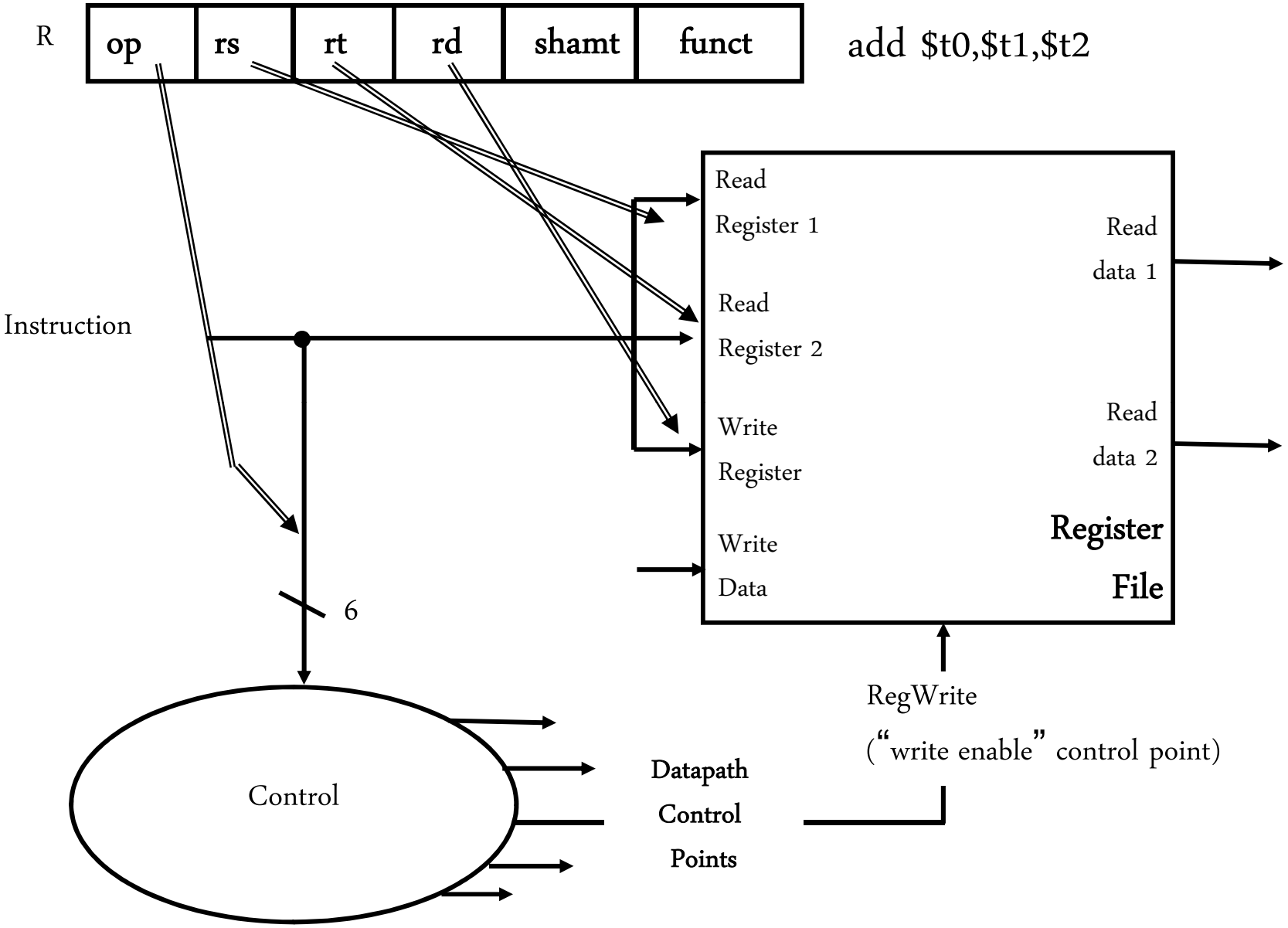
- 定长指令周期
  - 采用时钟边沿触发方式
    - 所有指令在时钟的一个边开始执行，在下一个边结束



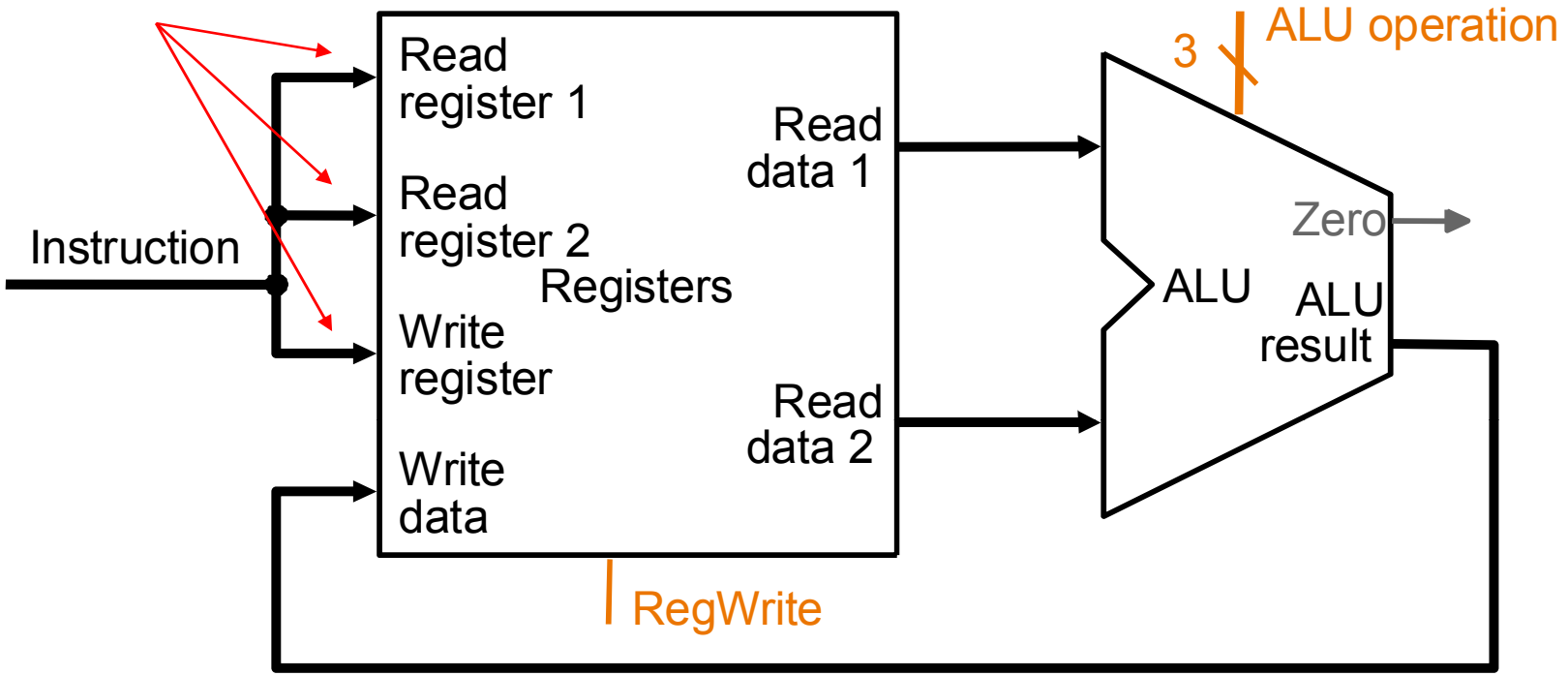
取指


指令字长32位
Add
4
PC
Read
address
Instruction
Instruction
memory

Decode


R
op
rs
rt
rd
shamt
funct
add $t0,$t1,$t2
Instruction
Read
Register 1
Read
Register 2
Write
Register
Write
Data
Read
data 1
Read
data 2
Register
File
6
RegWrite
("write enable" control point)
Control
Datapath
Control
Points

R-type指令的执行



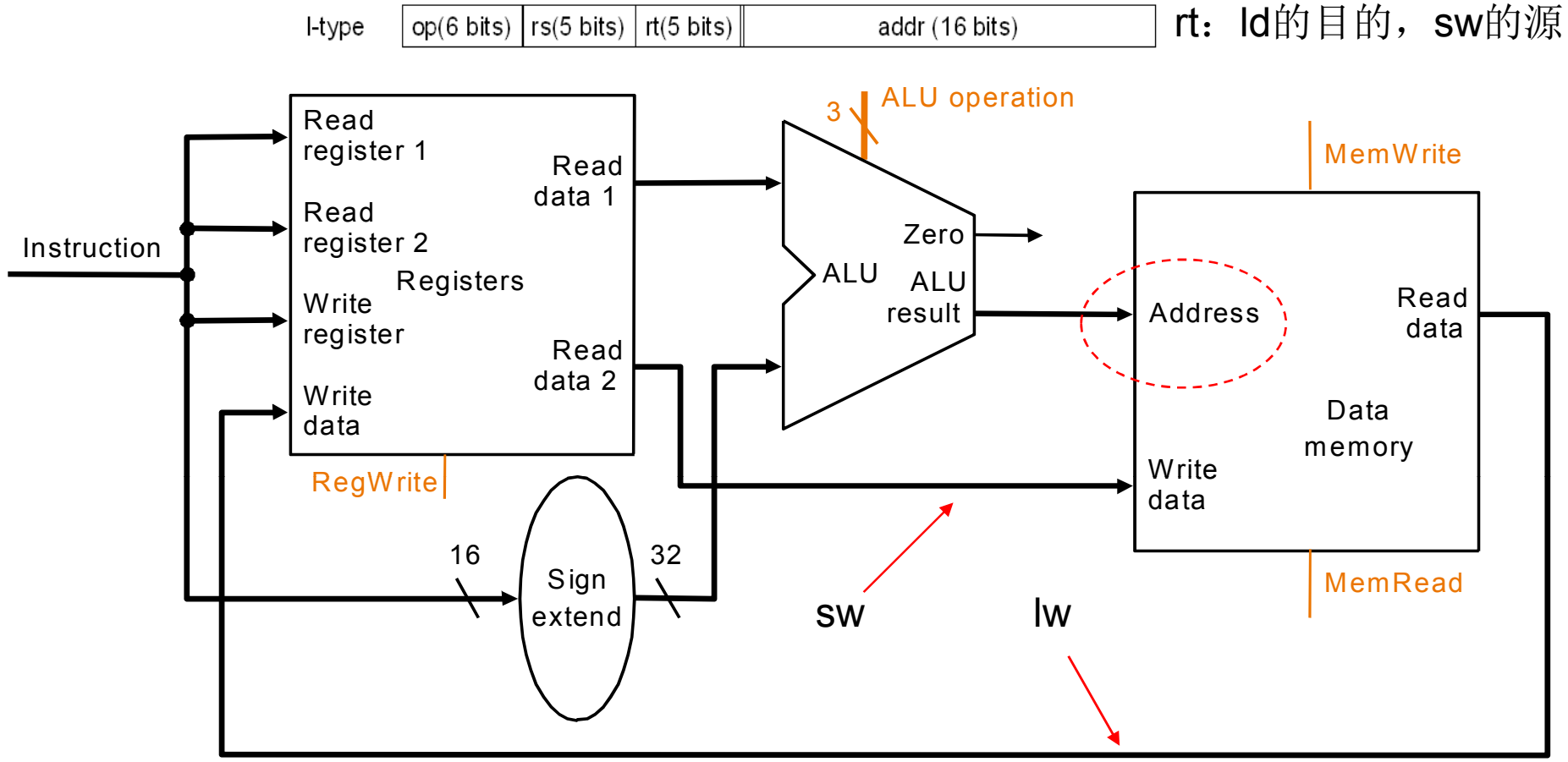
R-type	op(6 bits)	rs(5 bits)	rt(5 bits)	rd(5 bits)	shamt(5 bits)	funct(6 bits)

- 寄存器堆操作
  - 读：给出寄存器编号，则寄存器的值自动送到输出端口
  - 写：需要寄存器编号和控制信号RegWrite，时钟边沿触发
  - 在一个周期内，REG可以同时完成读写操作，但读出的是上一个周期写入的值

访存指令的执行

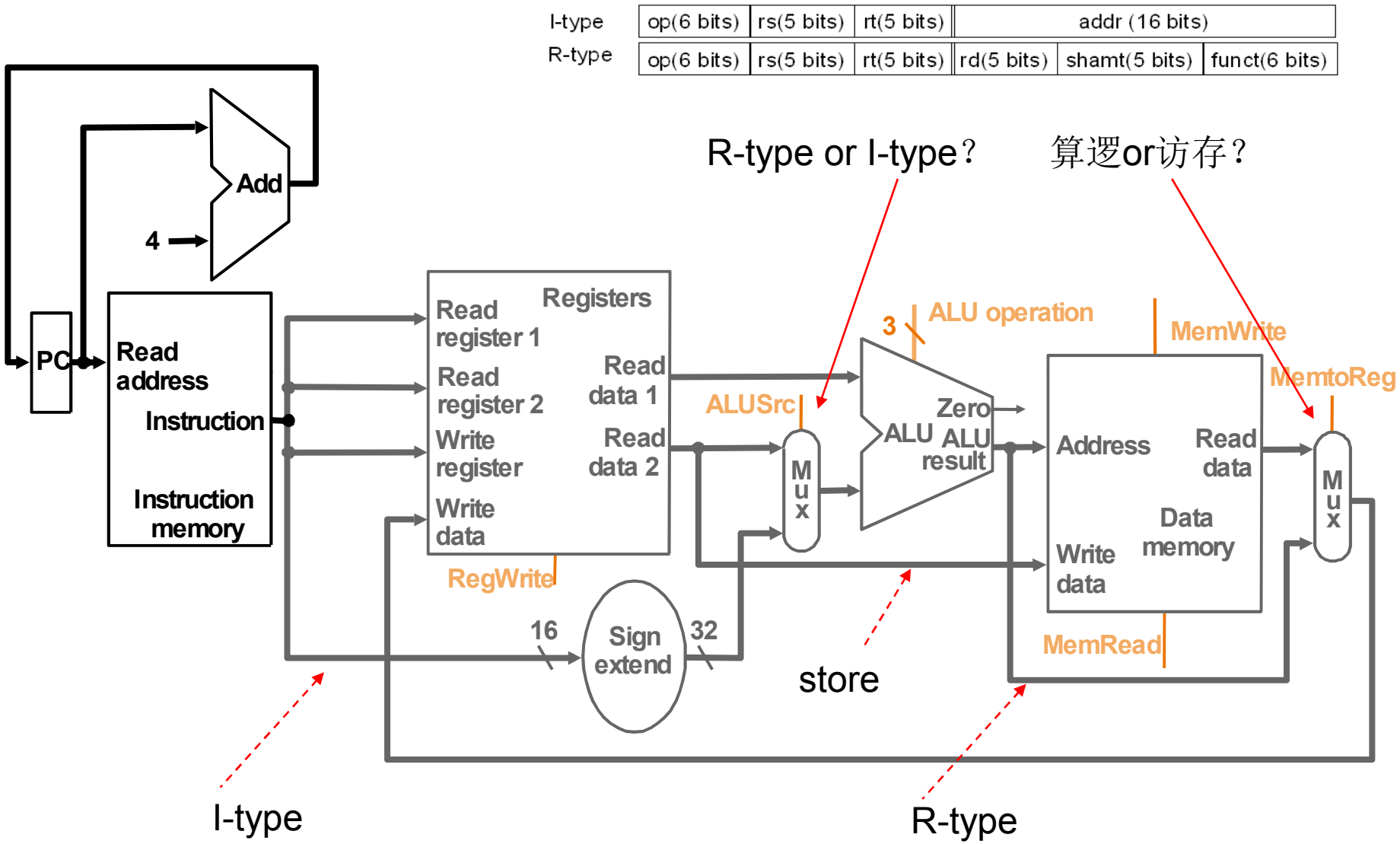
rs：基址

rt：ld的目的，sw的源

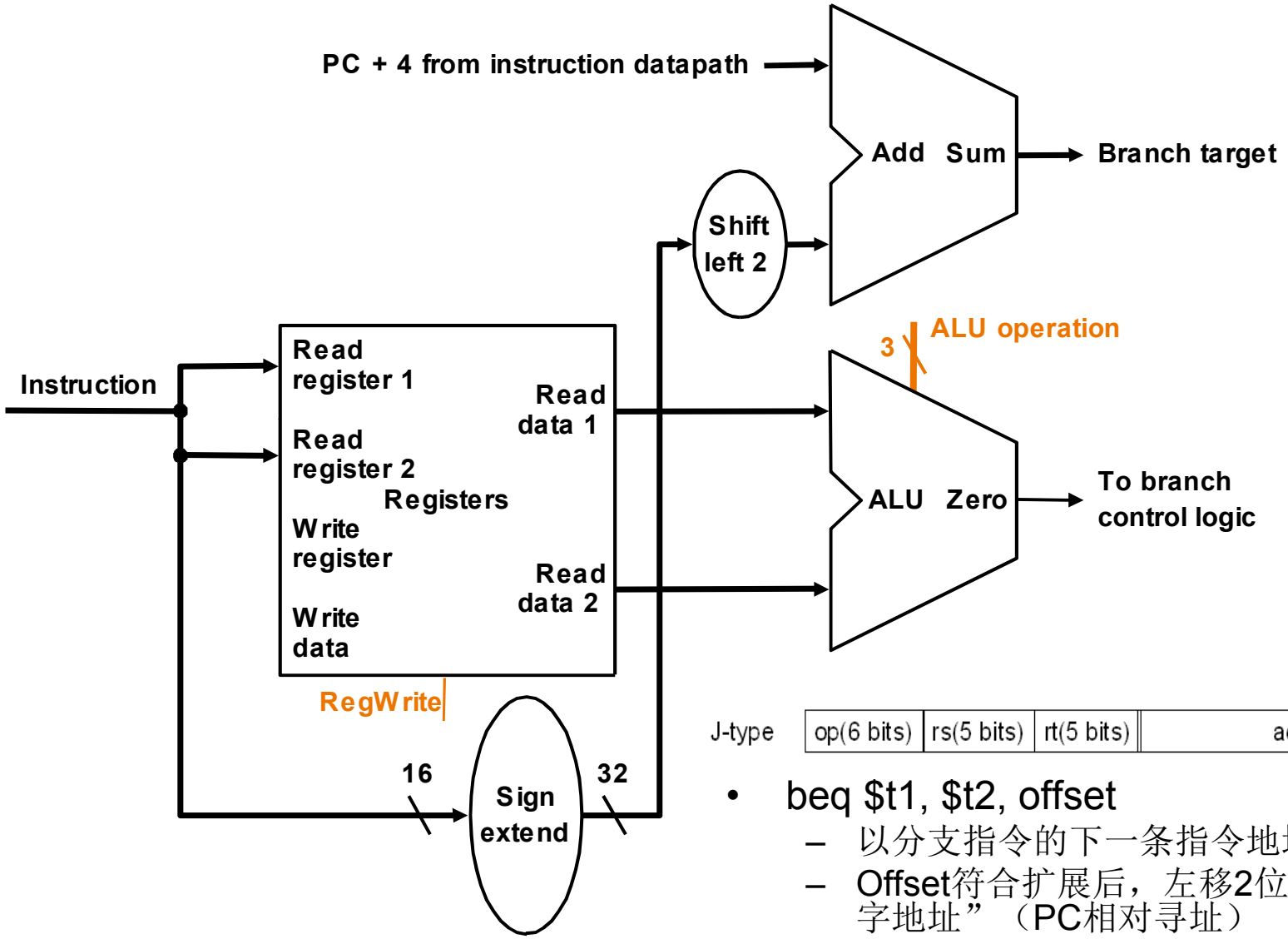


- lw $t1, offset($t2)；M($t2+offset) -> $t1
- sw $t1, offset($t2)； $t1 -> M($t2+offset)
- 需要对指令字中的16位偏移进行32位带符号扩展

访存指令和算逻指令的数据通路综合



条件转移beq

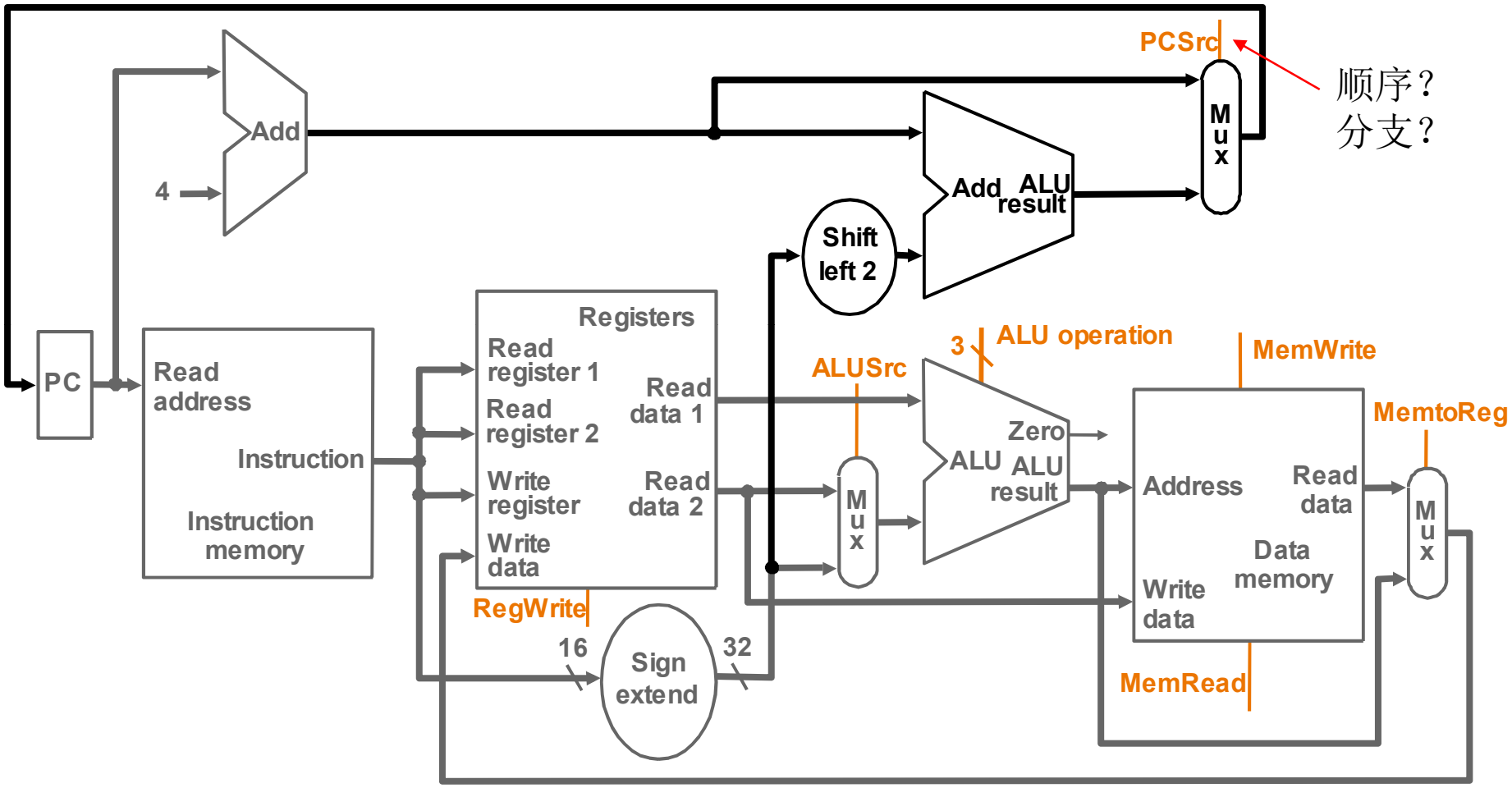


J-type

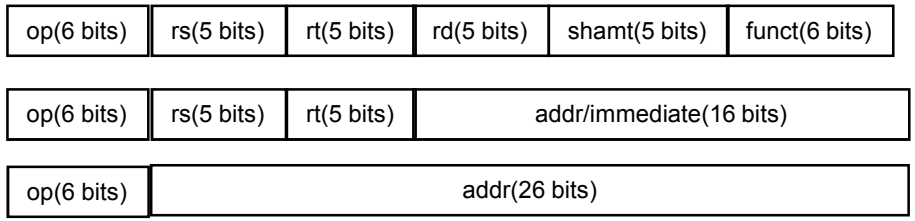
op(6 bits)	rs(5 bits)	rt(5 bits)	addr (16 bits)

- beq $t1, $t2, offset
  - 以分支指令的下一条指令地址为基址
  - Offset符合扩展后，左移2位形成“32位字地址”（PC相对寻址）

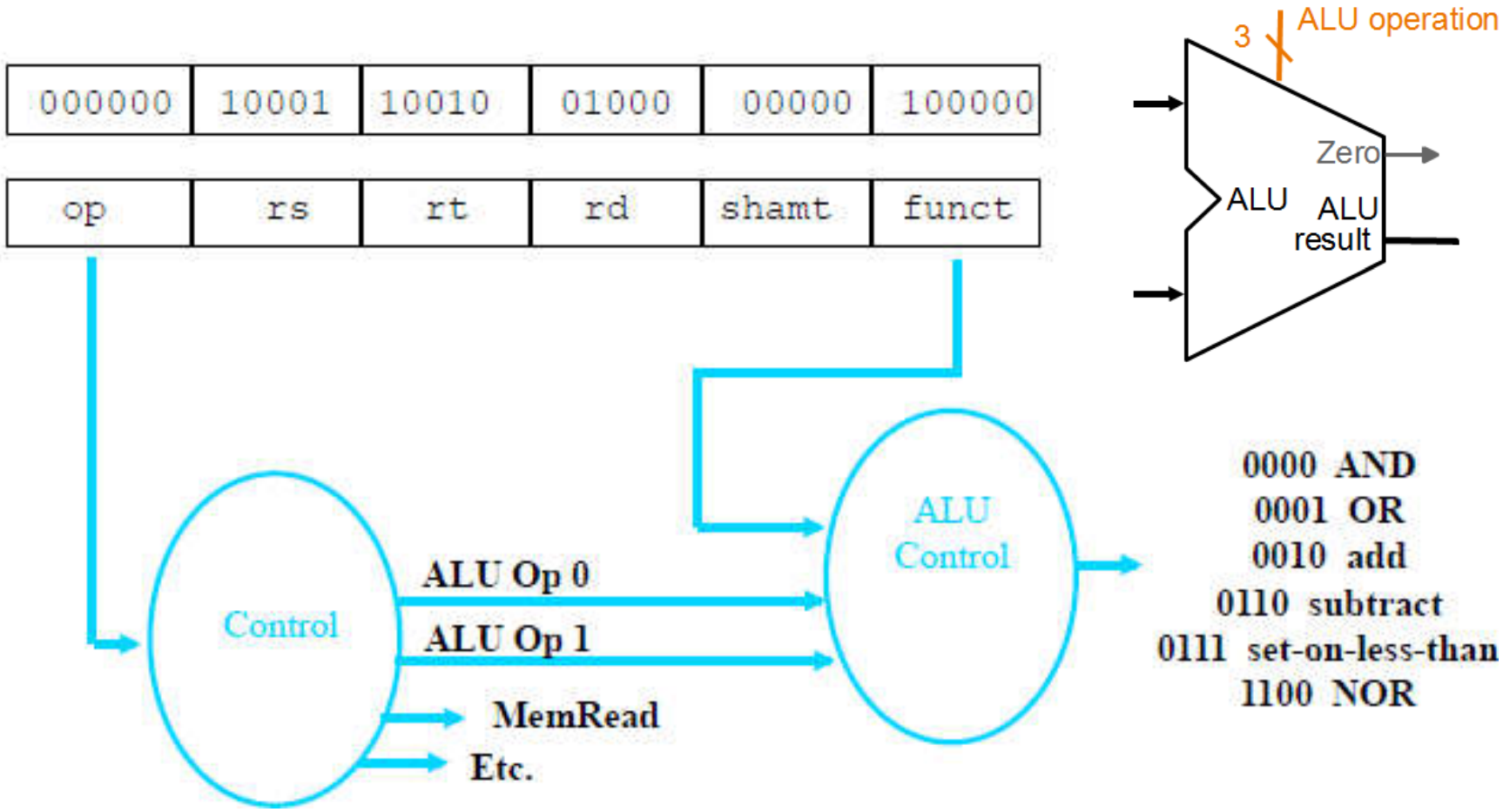
R-/I-/J-type操作数据通路总图



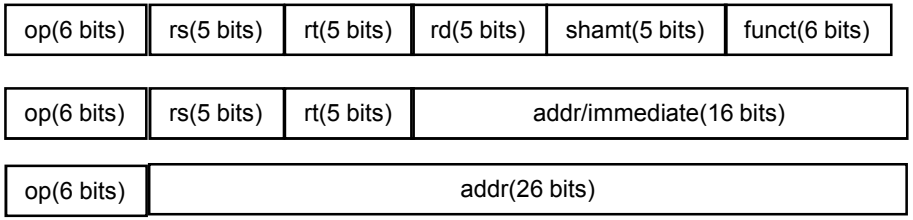
ALU控制信号



Example: add $8, $17, $18



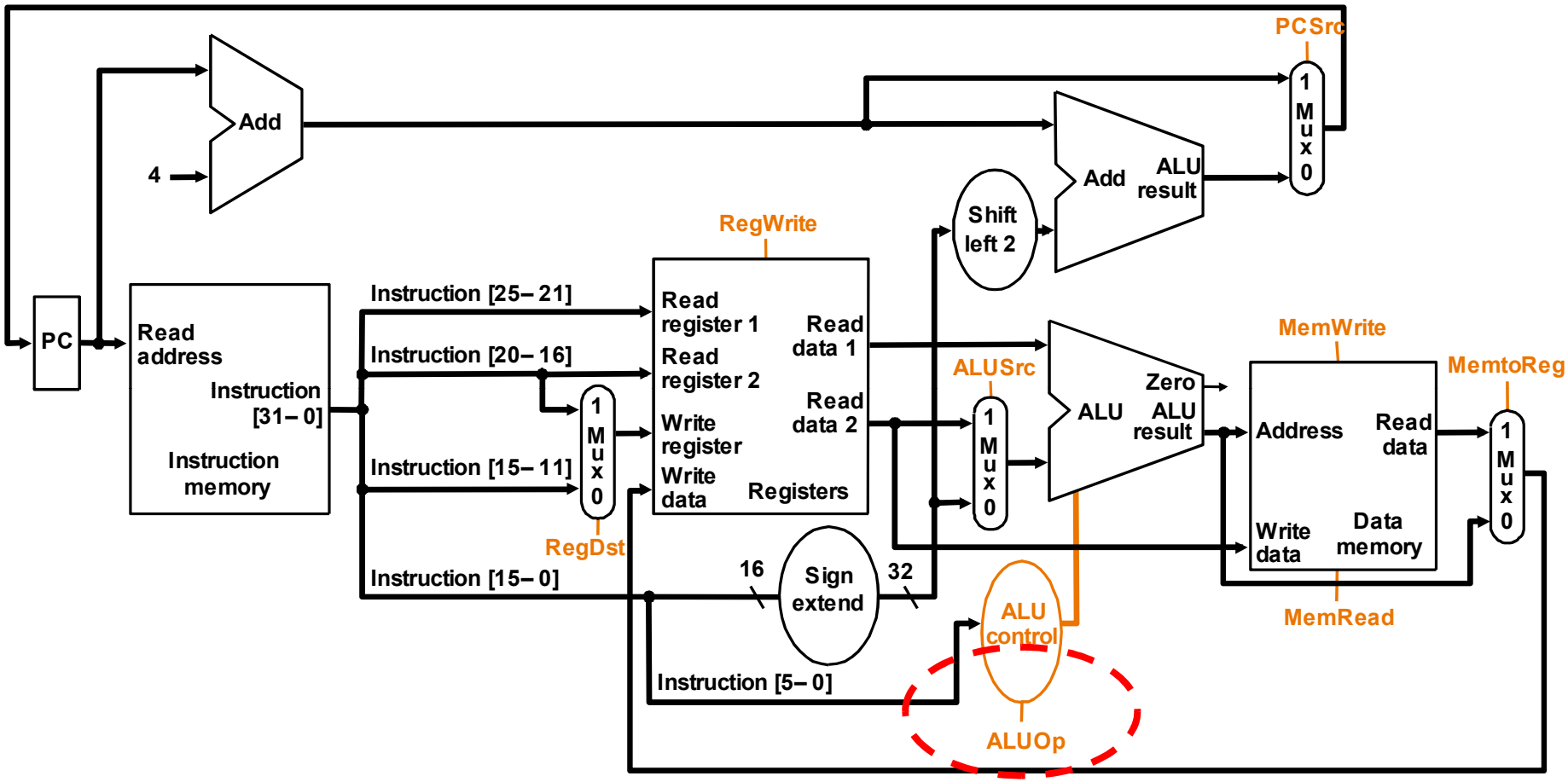
ALU控制信号



Instruction opcode	ALUop	Instruction operation	Funct field	desired ALU action	ALU ctrl input
LW	00	Load word	xxxxxx	add	010
sw	00	store word	xxxxxx	add	010
beq	01	Branch eq	xxxxxx	subtract	110
R-type	10	Add	100000	Add	010
R-type	10	Substract	100010	Substract	110
R-type	10	And	100100	And	000
R-type	10	Or	100101	Or	001
R-type	10	**Set on less than**	101010	**Set on less than**	111

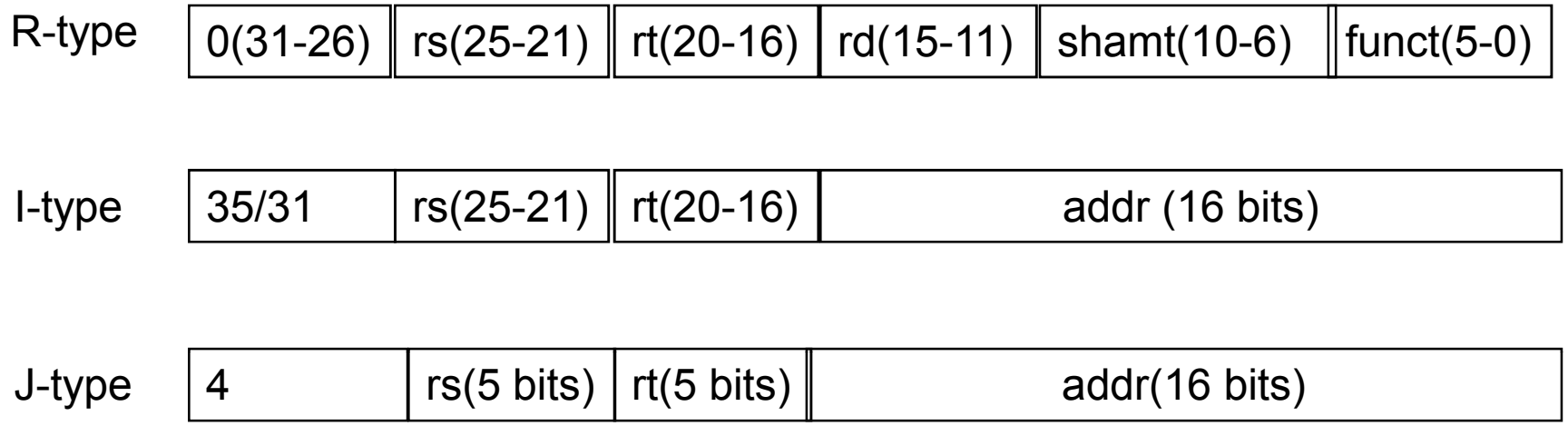
两位ALUop和五位func组合产生ALU_ctrl_input（即3位ALU operation）

ALU控制选择



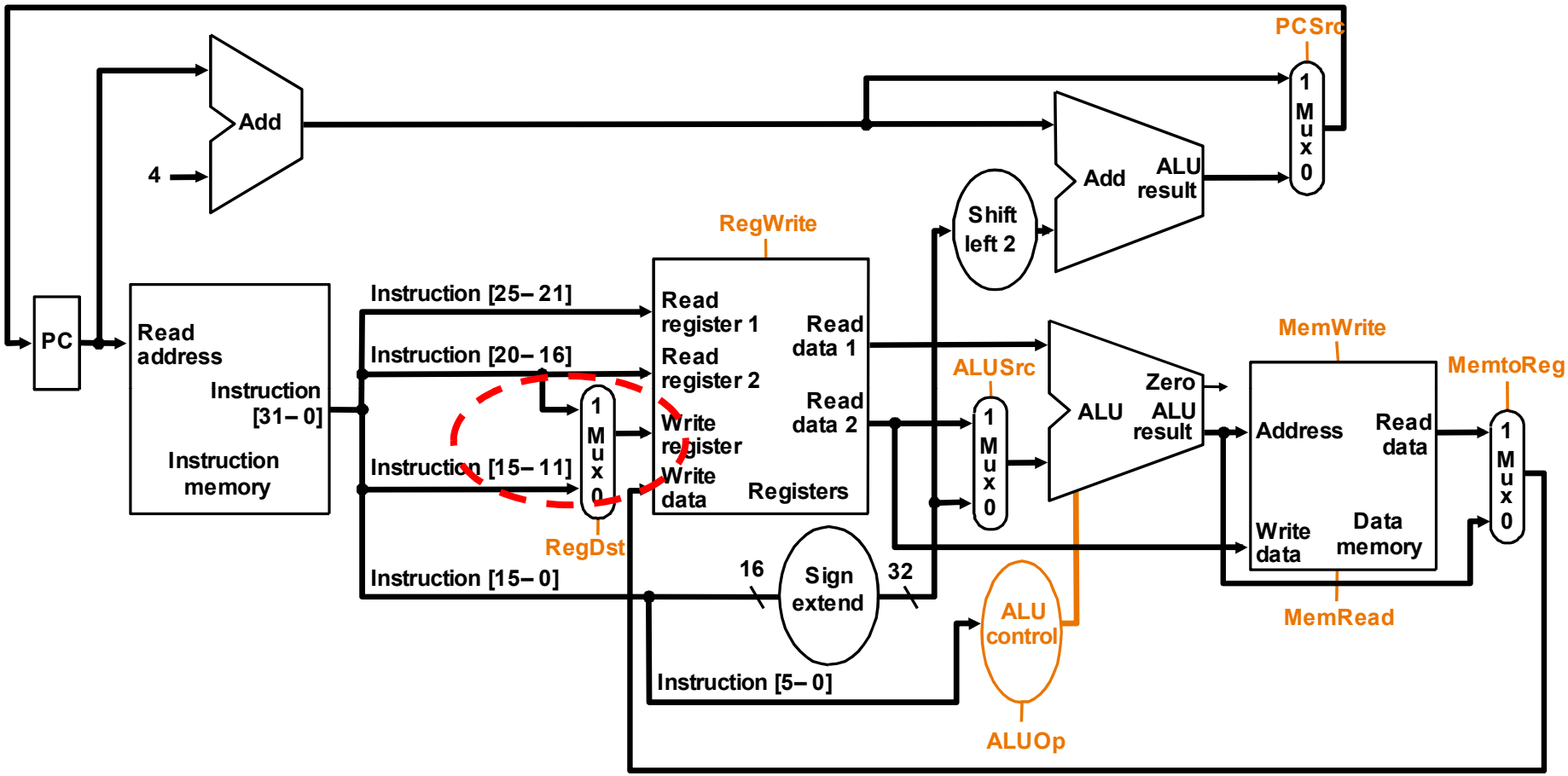
- 两位ALUop和func组合产生ALU控制选择

控制器：指令译码



- 指令格式分析
  - 操作码：在31-26
  - R-type：需要参考5-0
  - 目的地址：需要对目的寄存器进行选择控制
    - 对R-type指令，在rd
    - 对load，在rt

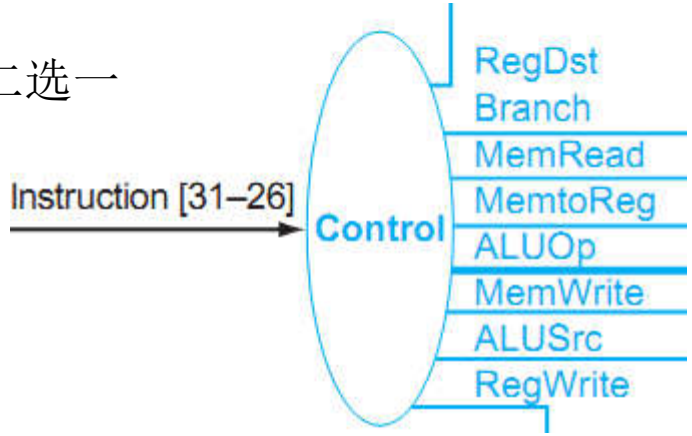
目的地址选择



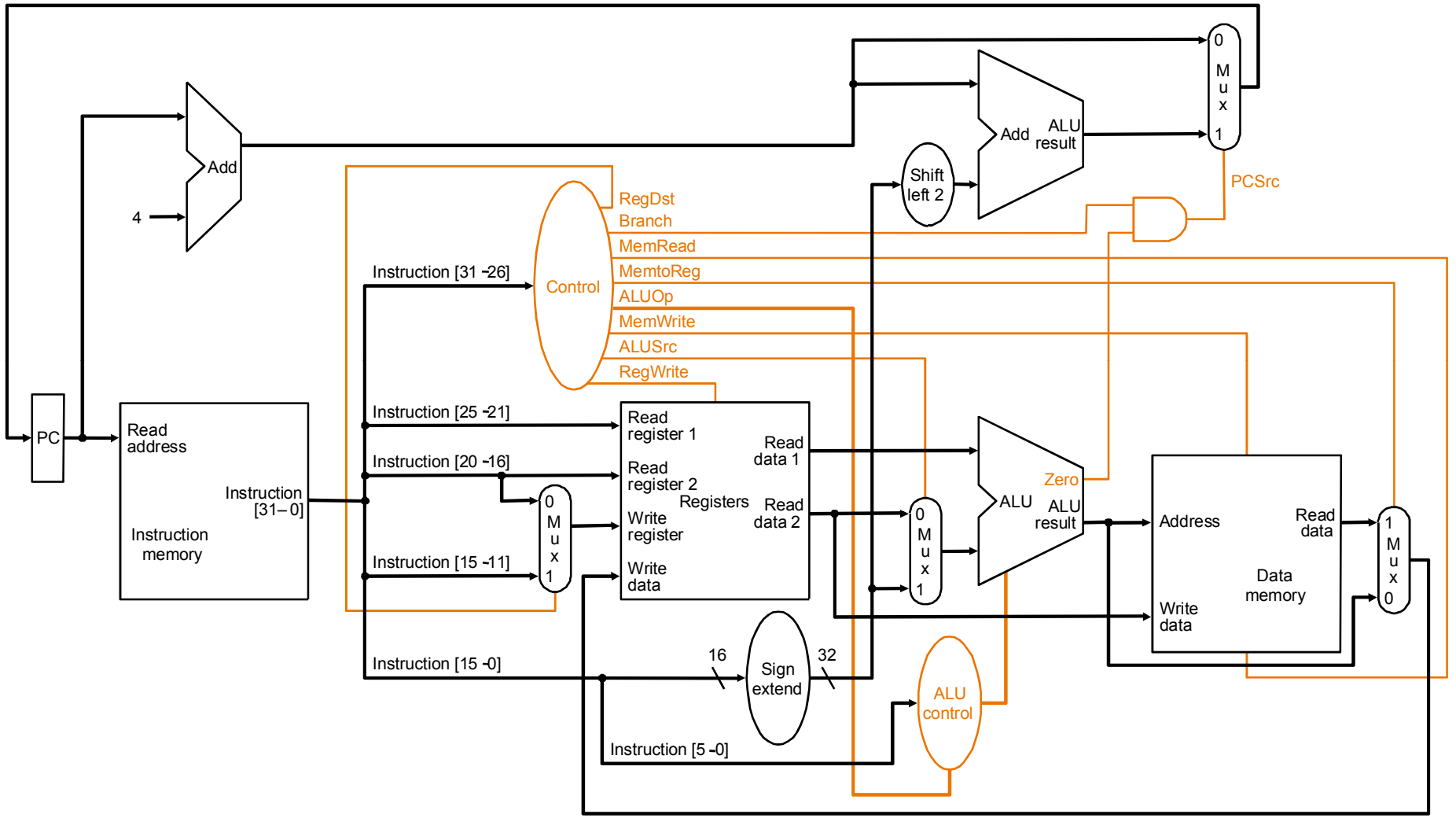
I-type	op(6 bits)	rs(5 bits)	rt(5 bits)	addr (16 bits)		
R-type	op(6 bits)	rs(5 bits)	rt(5 bits)	rd(5 bits)	shamt(5 bits)	funct(6 bits)

控制器：控制信号生成

- 所需控制信号：8个
  - RegDst：选择rt或rd作为写操作的目的寄存器
    - R-type指令与 load指令二选一
  - RegWrite：寄存器写操作控制
    - R-type指令与 load指令二选一
  - ALUSrc：ALU的第二个操作数来源
    - R-type指令（含branch指令）与 I-type指令二选一
  - ALUOp：R-type指令
  - MemRead：存储器读控制，load指令
  - MemWrite：存储器写控制，store指令
  - MemtoReg：目的寄存器数据来源
    - R-type指令与load指令二选一
  - PCSrc：nPC控制
    - 顺序执行与分支二选一
- 所有信号（除PCSrc）都可以根据op域译码产生
  - PCSrc依据两个条件
    - 指令是否是beq——增加一个“branch”控制信号指示
    - ALU的Zero状态



主控制部件

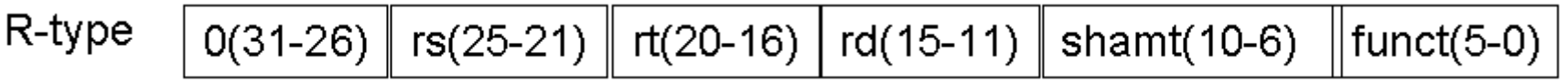

Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [31–26]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
0 M u x 1
Read register 1
Read register 2
Registers
Write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

控制部件真值表

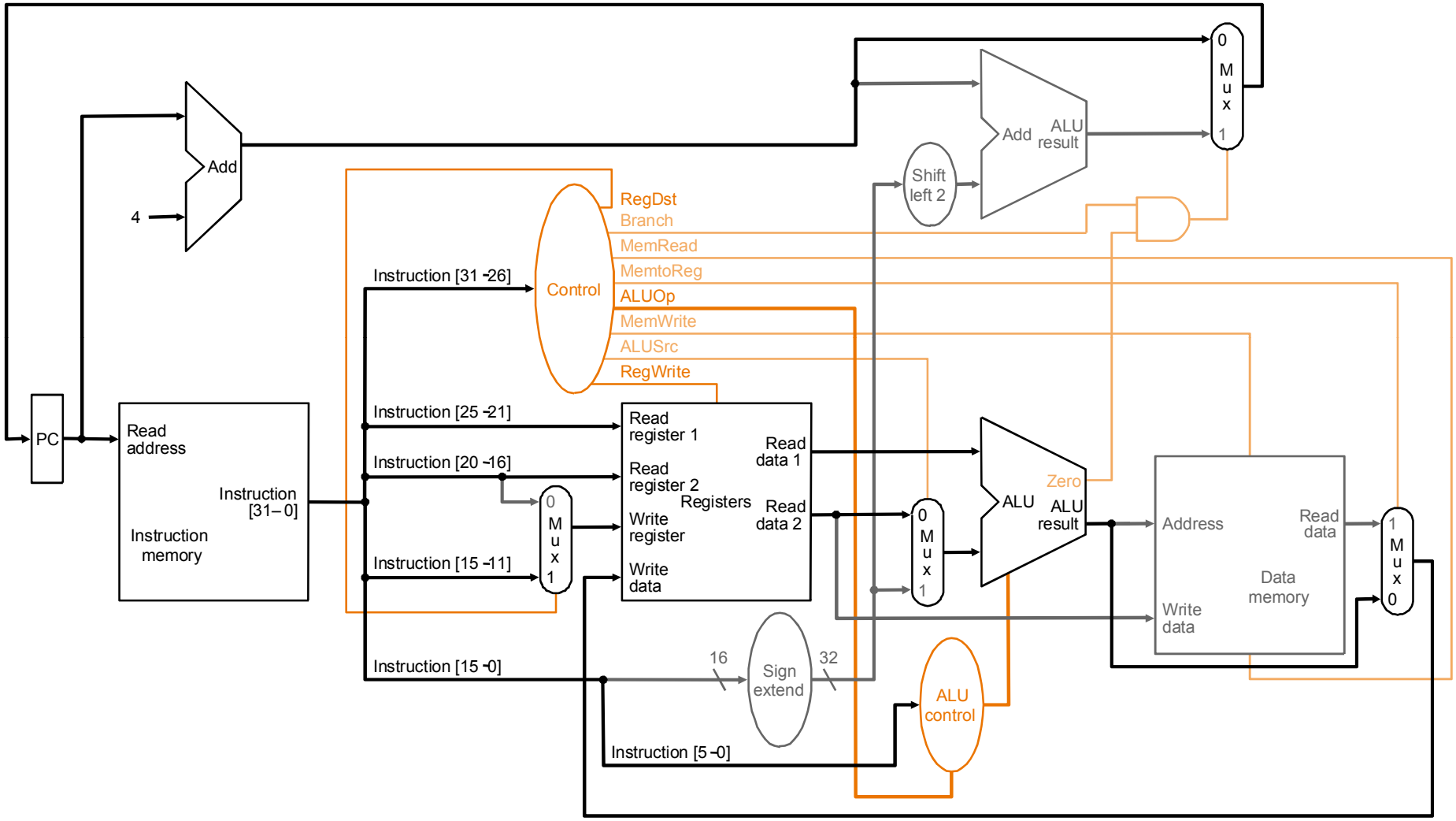
	Signal name	R-type	lw （31）	sw （35）	beq
inputs	op5（26）	0	1	1	0
	op4	0	0	0	0
	op3	0	0	1	0
	op2	0	0	0	1
	op1	0	1	1	0
	op0（31）	0	1	1	0
outputs	RegDst	1	0	x	x
	ALUSrc	0	1	1	0
	MemtoReg	0	1	x	x
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUop1	1	0	0	0
	ALUop0	0	0	0	1

R-type指令的执行过程

- add $t1, $t2, $t3; $t2+$t3->$t1
- 在一个周期内完成如下动作
  - 第一步：取指和PC＋1
  - 第二步：读两个源操作数寄存器$t2和$t3
  - 第三步：ALU操作
  - 第四步：结果写回目的寄存器$t1

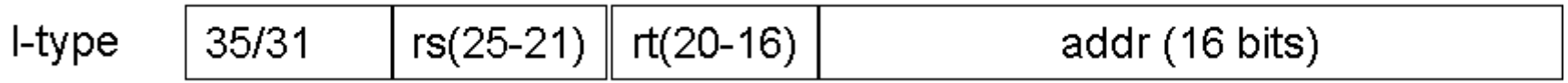


R-type指令的执行路径

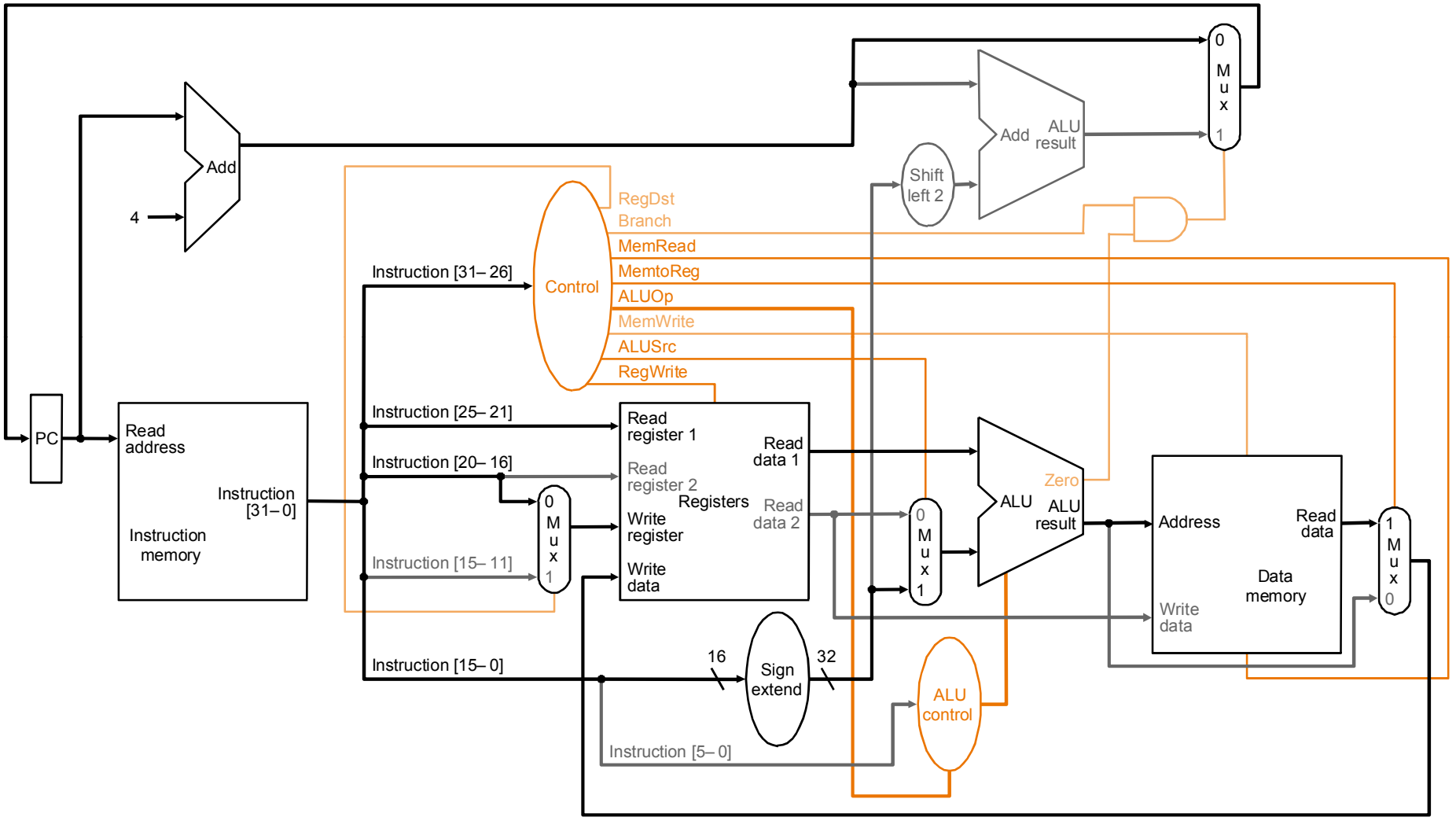


lw指令的执行过程

- lw $t1, offset($t2); M($t2+offset) -> $t1
- 第一步：取指和PC＋1
- 第二步：读寄存器$t2
- 第三步：ALU操作完成$t2与符号扩展后的16位offset加
- 第四步：ALU的结果作为访存地址，送往数据MEM
- 第五步：内存中的数据送往$t1

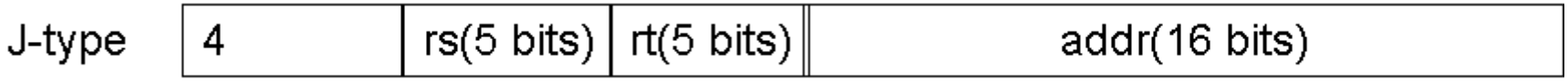


lw指令的执行路径

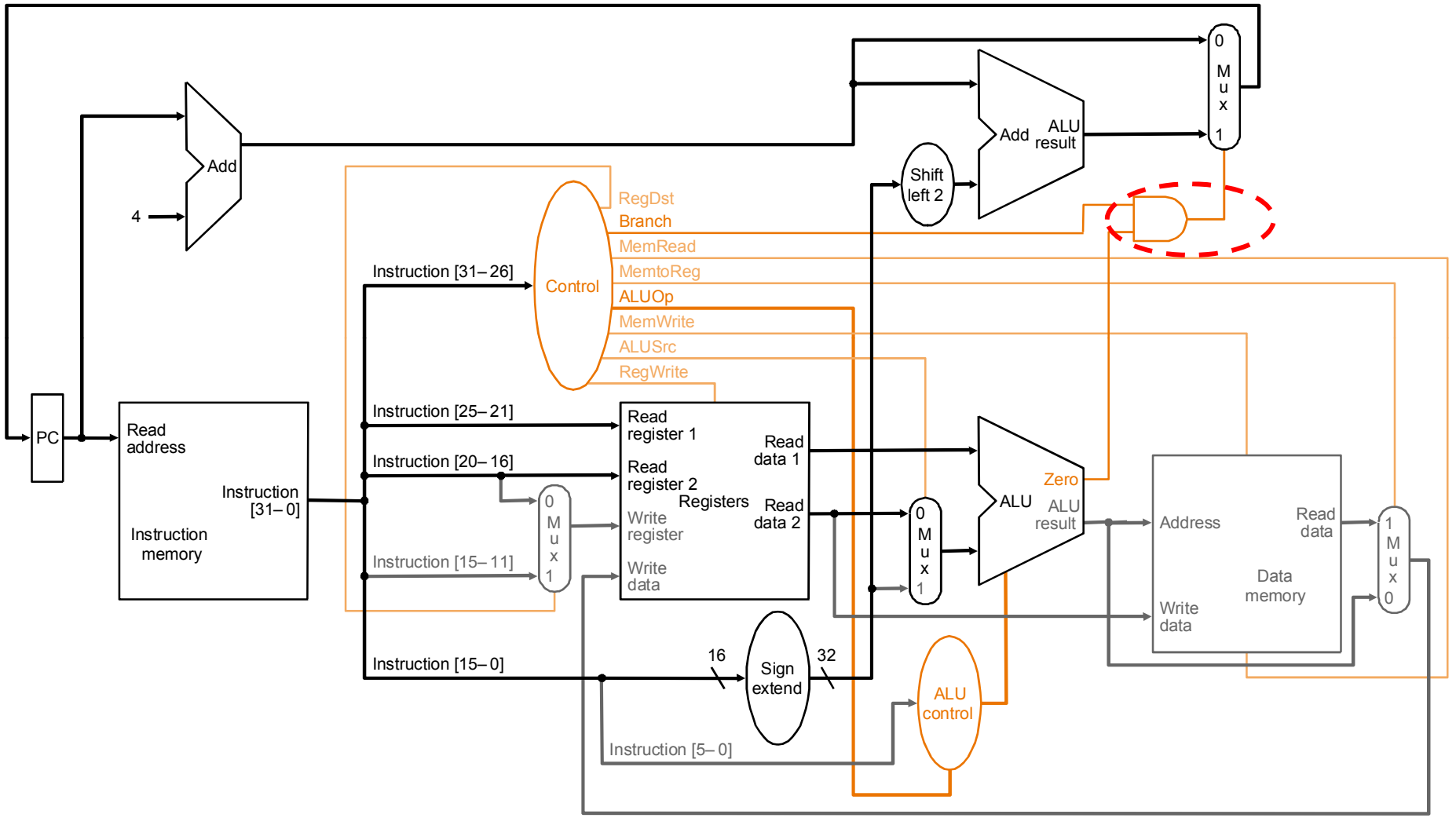

Add
4
0
M
u
x
1
Add
ALU result
Shift left 2
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Read register 1
Read register 2
Registers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

beq指令的执行过程

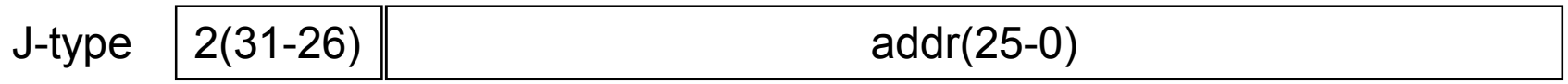
- beq $t1, $t2, offset
- 第一步：取指和PC＋1
- 第二步：读寄存器$t1，$t2
- 第三步：ALU将$t1和$t2相减；PC+4与被左移两位并进行符号扩展后的16位offset相加，作为分支目标地址
- 第四步：ALU的Zero确定应送往PC的值



beq的执行路径

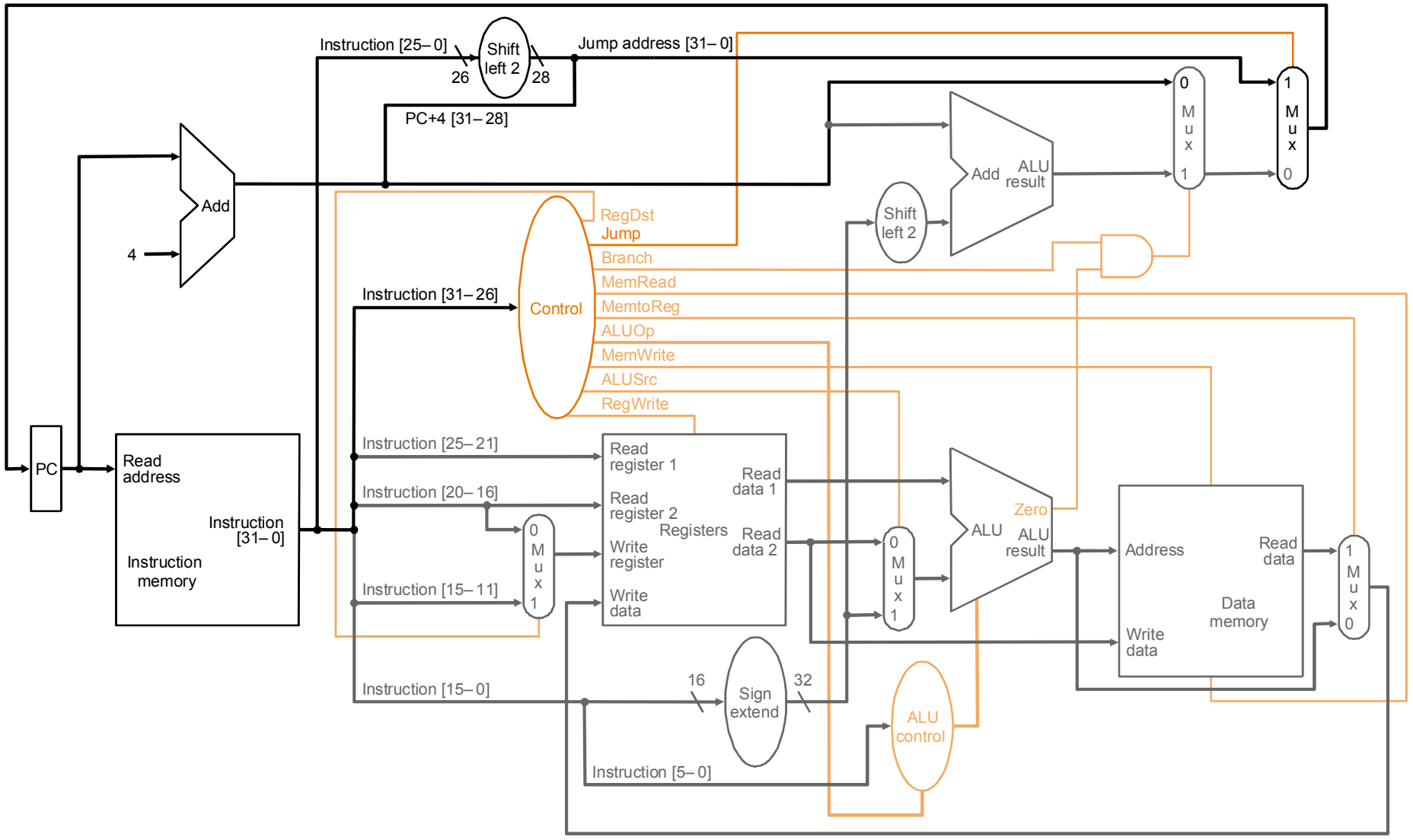

Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [31–26]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Registers
Write register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

jump指令的实现



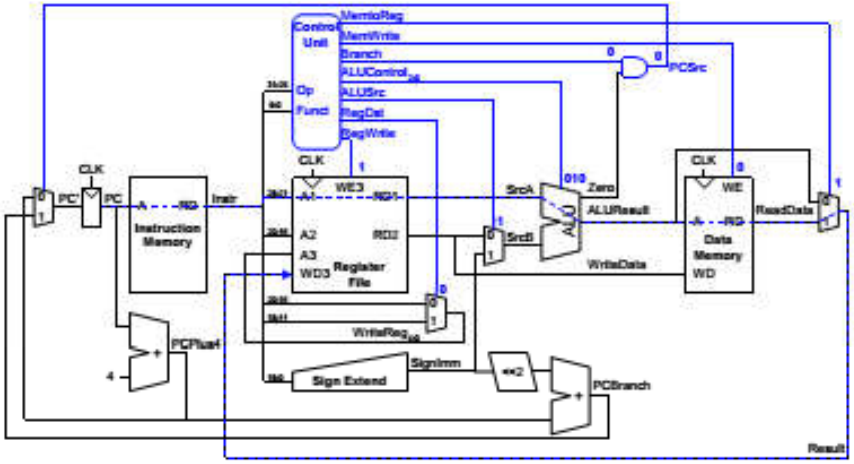
- 无条件转移，关键在于目标地址的拼装
  - PC+4的最高4位
  - 指令字中的26位地址
  - 最低两位补00
- “拼装”：只需合并地址总线
- 增加一个jump指令识别控制

jump指令的实现


Instruction [25–0]
Shift left 2
26
28
Jump address [31–0]
PC+4 [31–28]
Add
4
0
M
u
x
1
1
M
u
x
0
Add
ALU result
Shift left 2
RegDst
Jump
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
Registers
Write register
Write data
Read data 1
Read data 2
0
M
u
x
1
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign extend
32
ALU control

The Critical Path Dictates the Clock Period

Element		Delay
Register clk-to-Q	$t_{\text{pcq-PC}}$	30 ps
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory Read	t_{mem}	250
Register file read	t_{RFread}	150
Register file setup	t_{RFsetup}	20



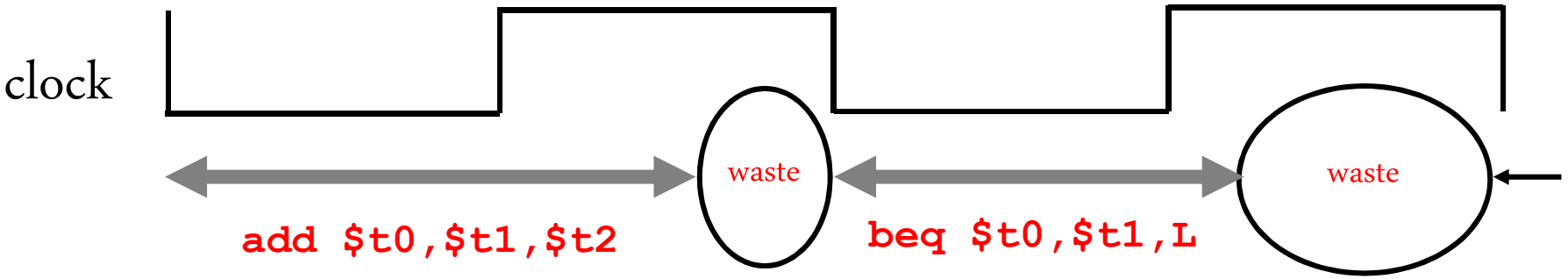
$$
\begin{aligned}
T_C &= t_{\text{pcq-PC}} + t_{\text{mem-I}} + t_{\text{RFread}} + t_{\text{ALU}} + t_{\text{mem-D}} + t_{\text{mux}} + t_{\text{RFsetup}} \\
&= (30 + 250 + 150 + 200 + 250 + 25 + 20)\ \text{ps} \\
&= 925\ \text{ps} \\
&= 1.08\ \text{GHz}
\end{aligned}
$$

定长or不定长单周期？

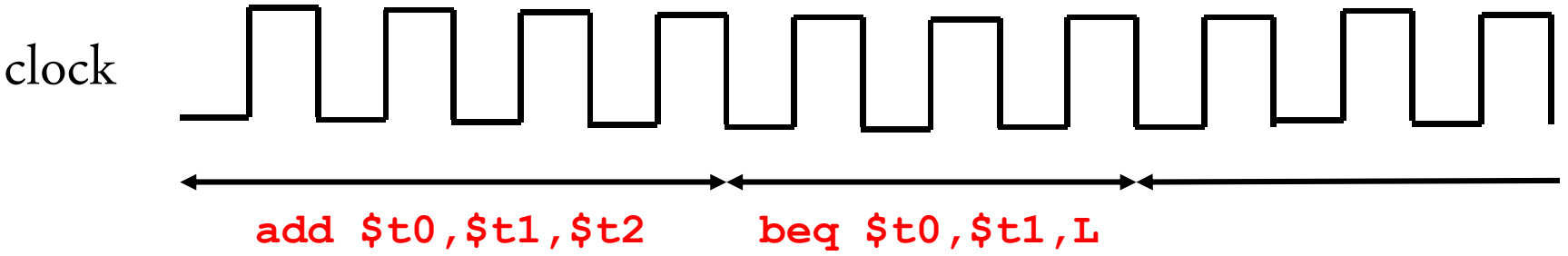
- 设程序中load有24%，store有12%，R-type有44%，beq有18%，jump有2%。试比较时钟定长单周期实现和不定长单周期实现的性能。
  - 程序执行时间＝指令数×CPI×时钟宽度
  - 定长单周期的时钟为8ns
  - 不定长单周期的时钟可以是2ns～8ns。其平均指令执行时间＝8×24%＋7×12%＋6×44%＋5×18%＋2×2%＝6.3ns
  - 因此，变长实现较定长实现快8/6.3=1.27倍

Clocking: single-cycle vs. multicycle

Single-cycle Implementation



Multicycle Implementation

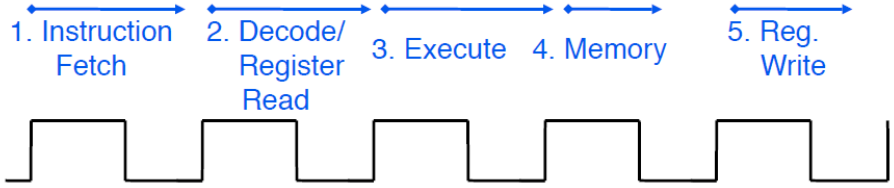


- Multicycle Implementation:

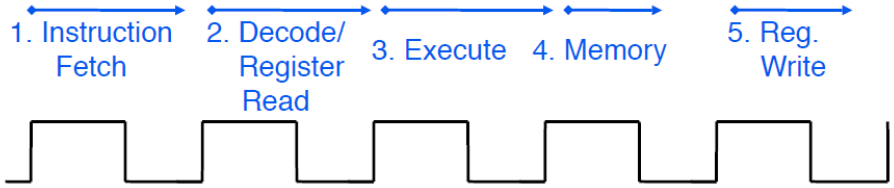
  less waste = higher performance

多周期实现

- 根据指令执行所使用的功能部件，将执行过程划分成多个阶段，每个阶段一个周期
  - 在一个周期内的各个部件并行工作
  - 功能部件可以在不同的阶段（周期）复用
    - 有利于降低硬件实现复杂度
- 时钟周期确定
  - 每个周期的工作尽量平衡
  - 假设一个周期内可以完成
    - 一次MEM访问，or
    - 一次寄存器访问（2 reads or one write），or
    - 一次ALU操作

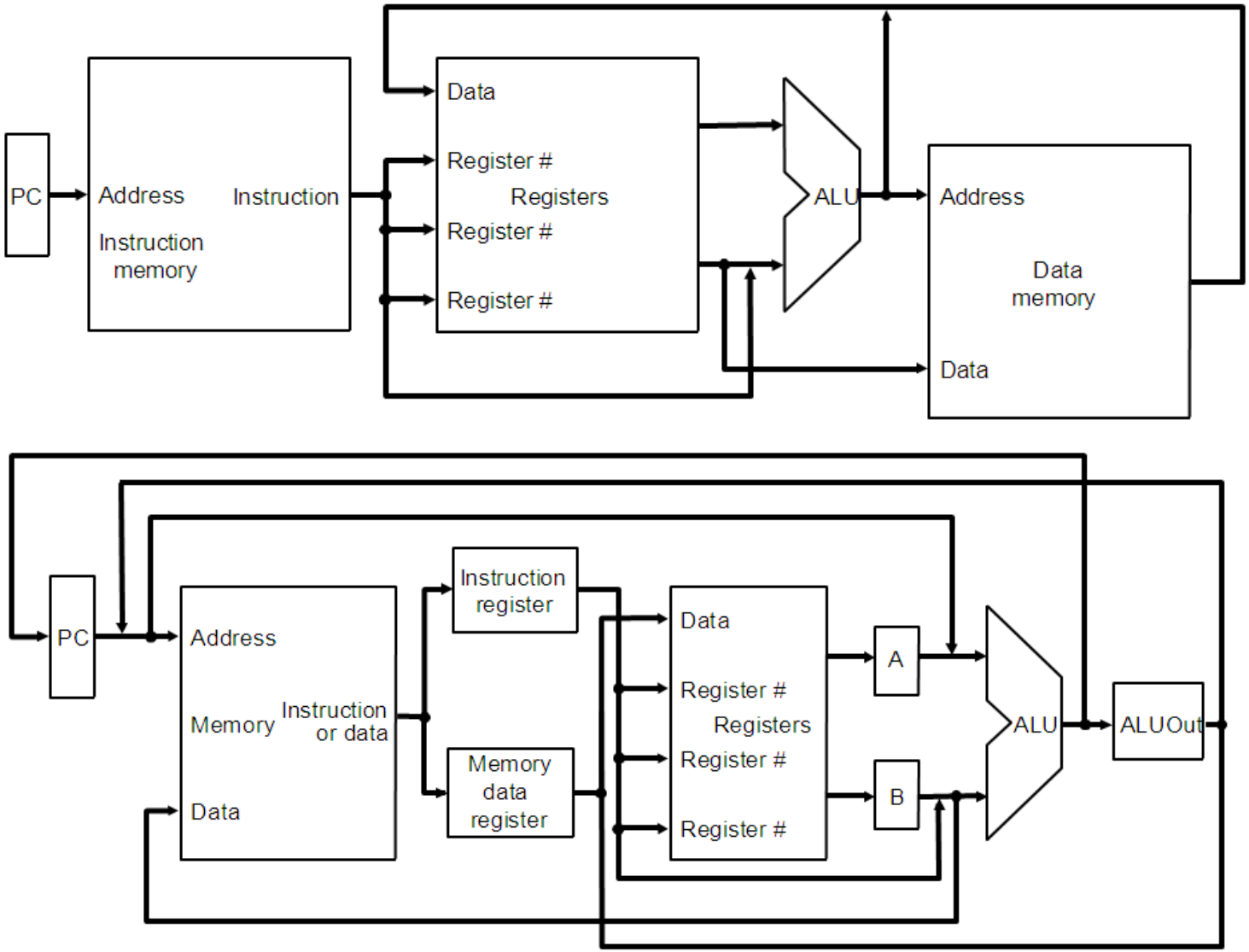


指令执行的阶段划分

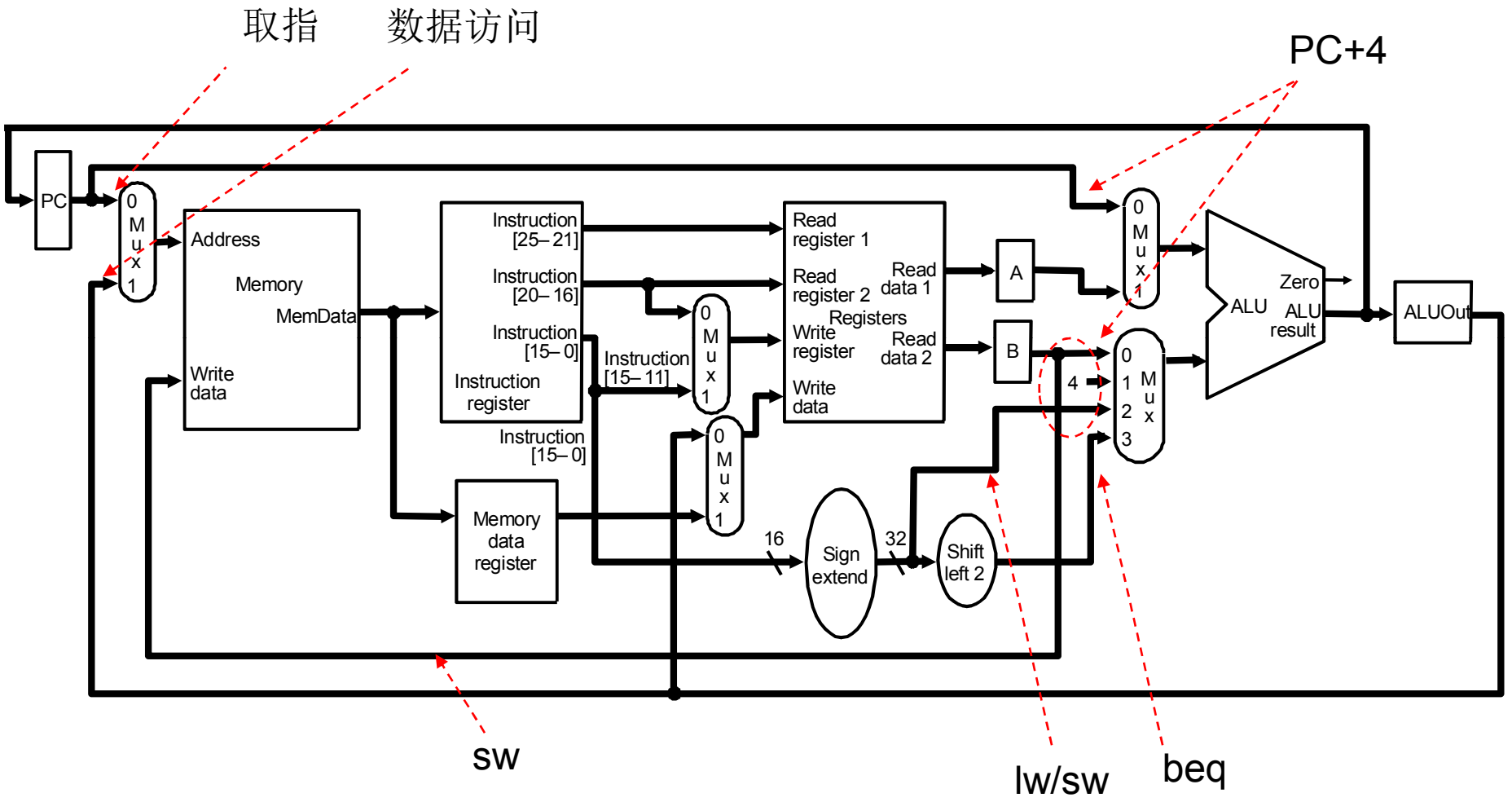


- 共5个阶段
  - 取指
  - 译码阶段、计算beq目标地址
  - R-type指令执行、访存地址计算，分支完成阶段
  - lw读，store和R-type指令完成阶段
  - lw完成阶段
- 注意
  - 定长机器周期：每个指令周期含5个机器周期
    - 等于时钟节拍
  - 不定长指令周期：分别为3、4、5个机器周期
- 控制器根据机器周期标识发出控制信号

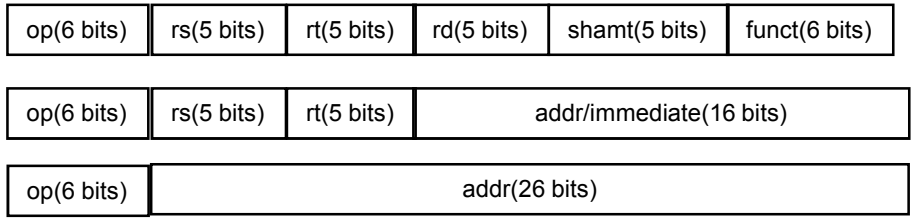
Overview

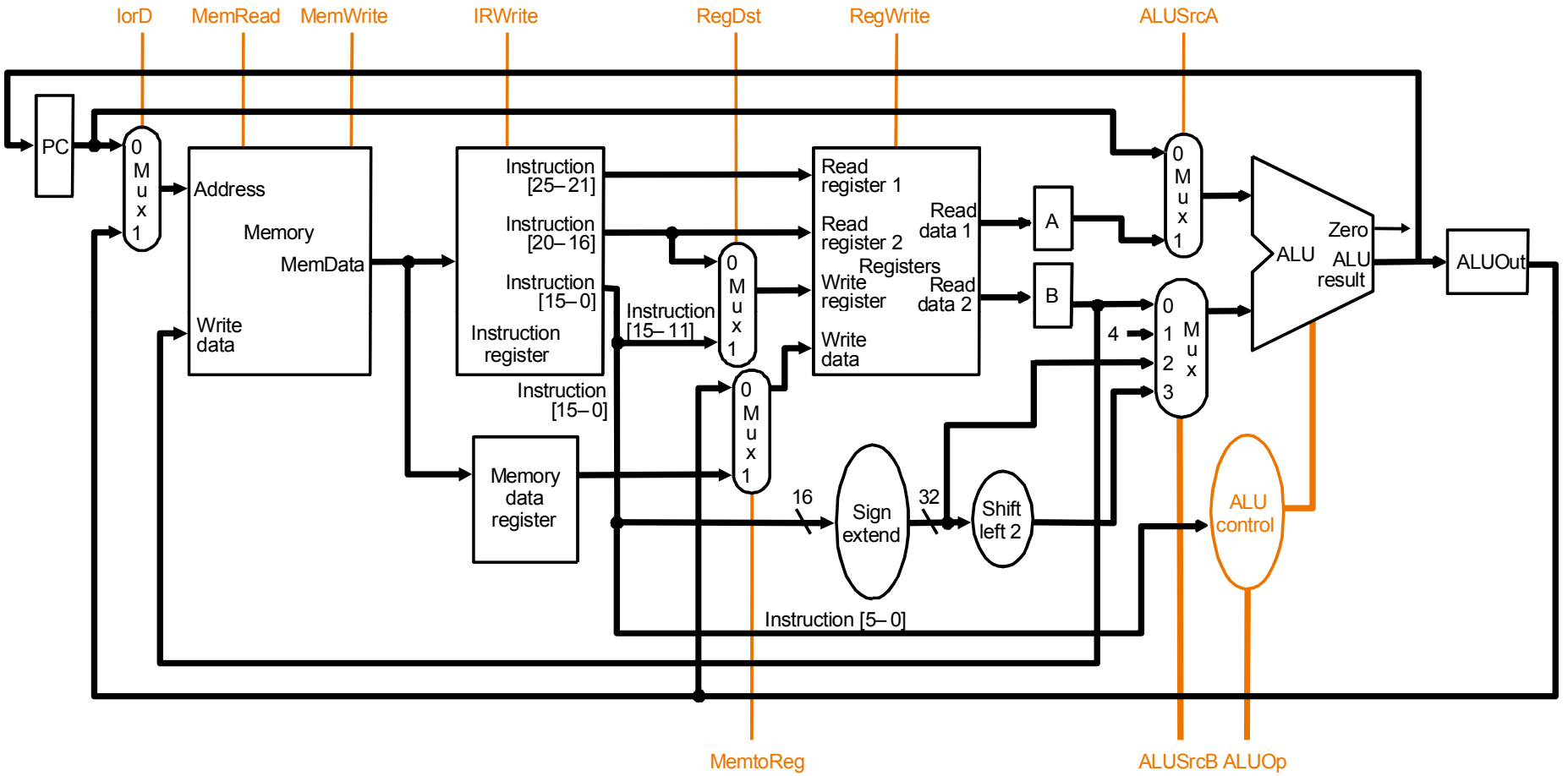

PC
Address
Instruction
Instruction memory
Data
Register #
Registers
Register #
Register #
ALU
Address
Data memory
Data
PC
Address
Memory
Instruction or data
Data
Instruction register
Memory data register
Data
Register #
Registers
Register #
Register #
A
B
ALU
ALUOut

多周期数据通路


取指
数据访问
PC+4
PC
0
M
u
x
1
Address
Memory
MemData
Write
data
Instruction
[25–21]
Instruction
[20–16]
Instruction
[15–0]
Instruction
register
Instruction
[15–0]
Instruction
[15–11]
Memory
data
register
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
ALU
Zero
ALU
result
ALUOut
16
Sign
extend
32
Shift
left 2
sw
lw/sw
beq

多周期控制信号

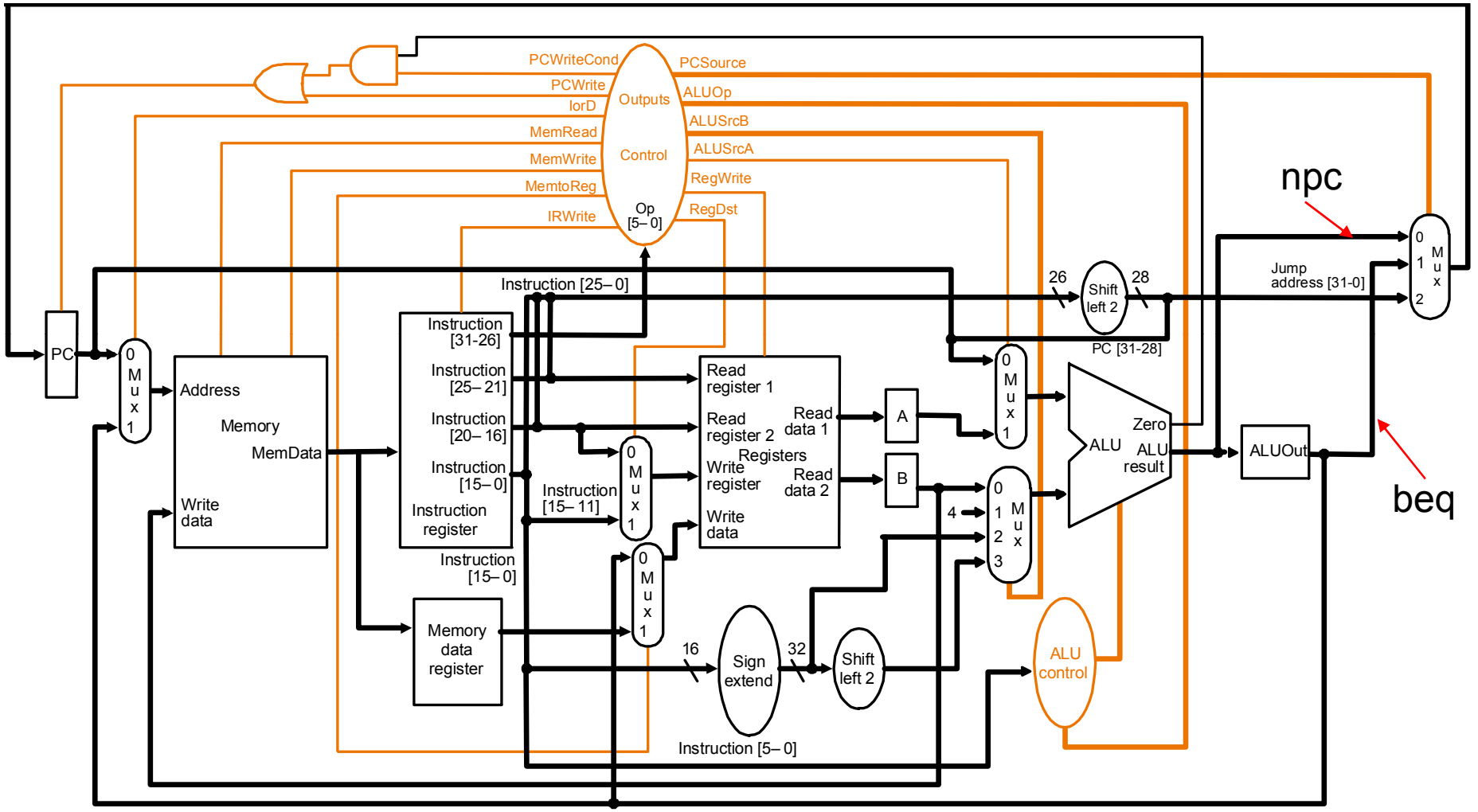

op(6 bits)
rs(5 bits)
rt(5 bits)
rd(5 bits)
shamt(5 bits)
funct(6 bits)
op(6 bits)
rs(5 bits)
rt(5 bits)
addr/immediate(16 bits)
op(6 bits)
addr(26 bits)


IorD
MemRead
MemWrite
IRWrite
RegDst
RegWrite
ALUSrcA
PC
0
M
u
x
1
Address
Memory
MemData
Write
data
Instruction
[25–21]
Instruction
[20–16]
Instruction
[15–0]
Instruction
register
Instruction
[15–0]
Instruction
[15–11]
Memory
data
register
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
ALU
Zero
ALU
result
ALUOut
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5–0]
MemtoReg
ALUSrcB
ALUOp

PC的写控制

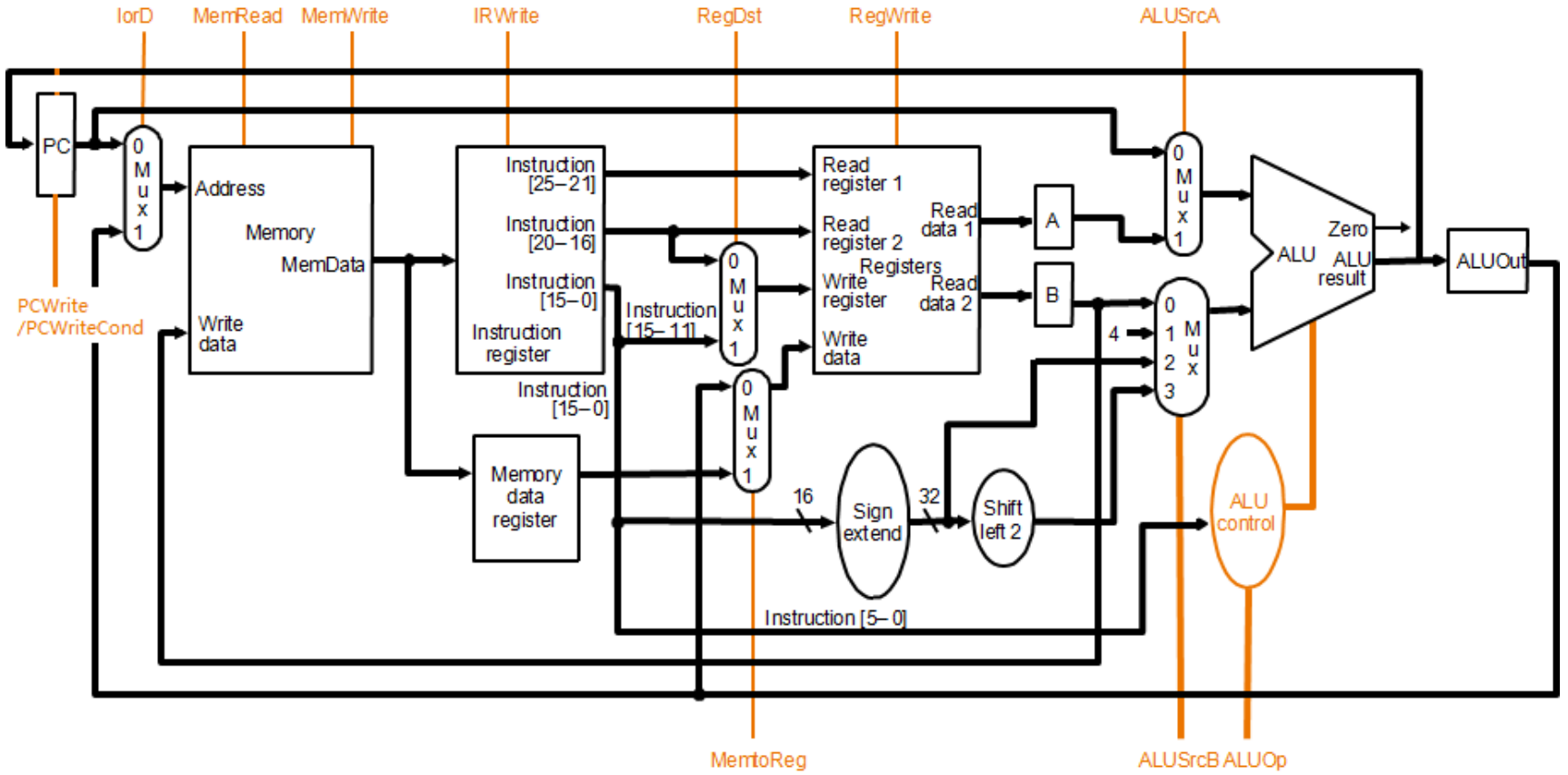
- 在一个指令周期内，PC不能变，因此需要写控制信号
- 3种情况
  - ALU：PC＋4的输出直接存入PC
  - ALUOut：beq指令的目标地址
  - jump指令
- 需要两个写控制
  - 无条件写PCWrite：PC+4，jump
  - 有条件写PCWriteCond：beq

主控制部件


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

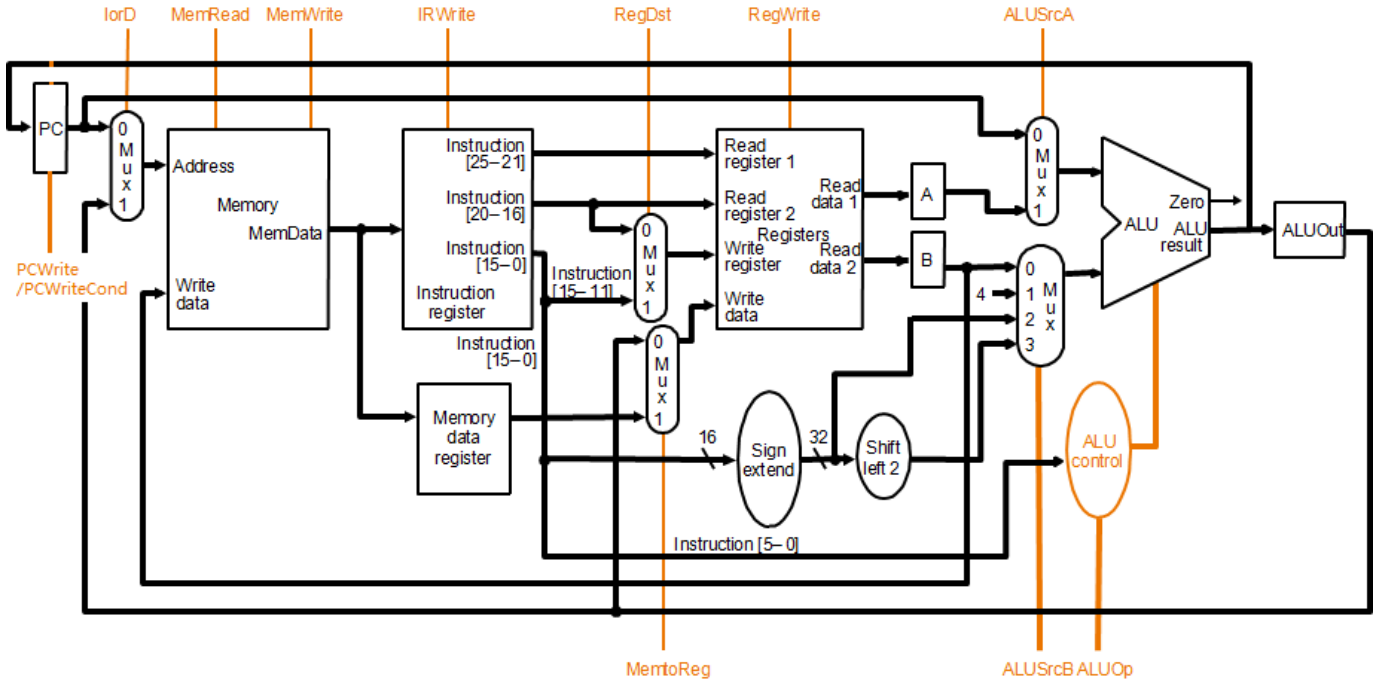
取指阶段

- 取指
  - 根据PC从MEM中取指，IR=MEM[PC]
  - 计算NPC，PC=PC+4
  - 控制信号：MemRead, IRWrite, IorD, ALUSrcA, ALUSrcB, ALUOp, PCWrite



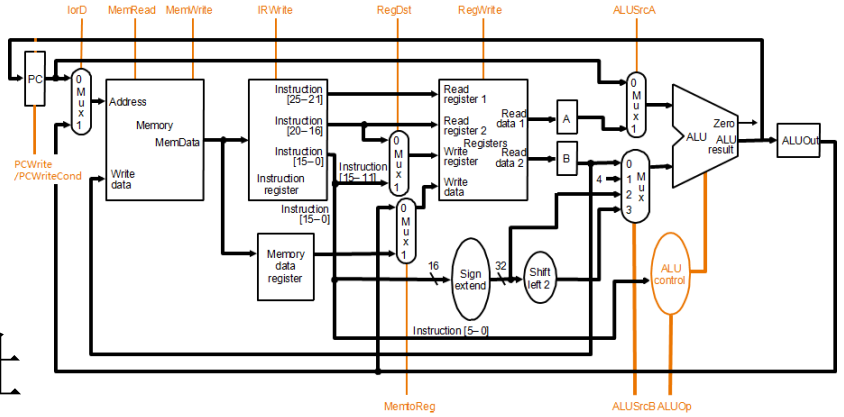
译码阶段

- 指令译码和读寄存器
  - 将rs和rt送往A和B：A=Reg[IR[25-21]], B=Reg[IR[20-16]]
- 计算beq目标地址
  - ALUOut=PC+(sign-extend(IR[15-0] <<2)
    - 由于此时尚不知是何指令，所以读寄存器和计算分支地址可能无效，但亦无害，可以节省后面的操作
- 控制信号：ALUSrcA, ALUSrcB, ALUOp



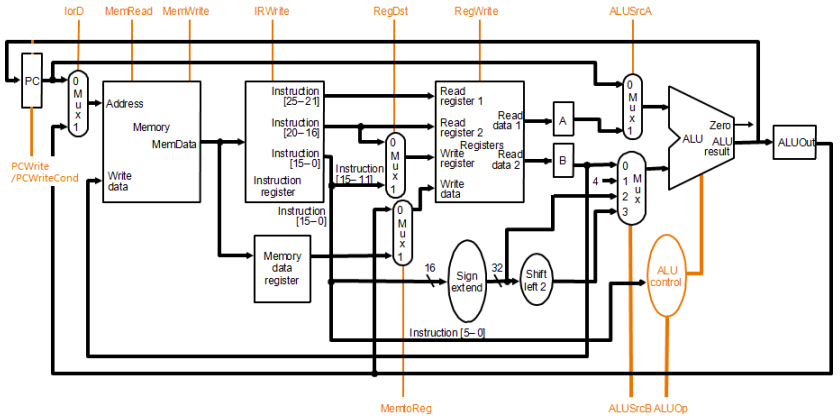
R-type执行、访存地址计算、分支完成阶段

- 依赖于指令类型
  - R-type指令
    - ALUOut= A op B
  - 访存指令：计算访存地址
    - ALUOut=A+(sign-extend(IR[15-0])
  - beq指令
    - if (A == B) PC=ALUOut；路径？
  - jump指令
    - PC = PC[31-28] || (IR[25-0] << 2)
- 需要的控制信号



R-type和sw完成、lw读阶段；lw写阶段

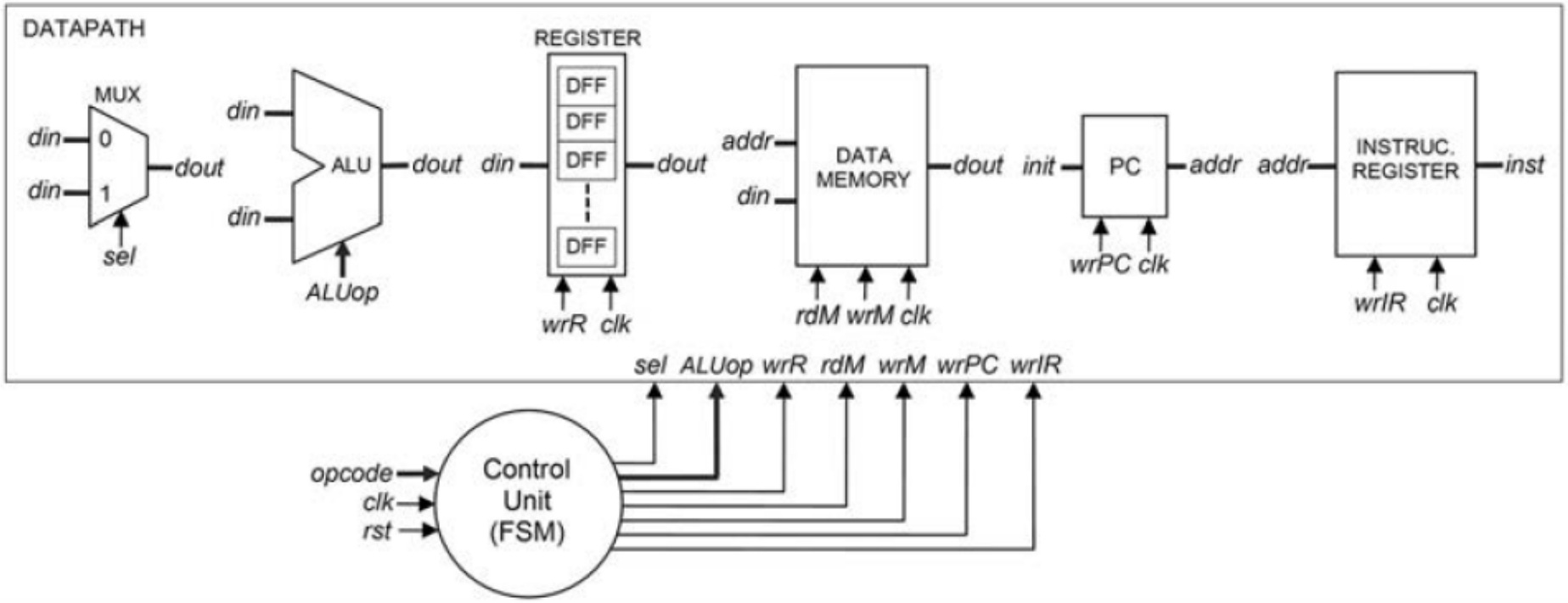
- 第四阶段
  - R-type完成：结果写回
    - Reg[IR[15-11]] = ALUOut
  - sw完成：写入MEM
    - MEM[ALUOut] = B
  - lw读：
    - MDR = MEM[ALUOut]
- 第五阶段
  - lw写回：Reg[IR[15-11]] = MDR
- 所需的控制信号



Multicycle RTL

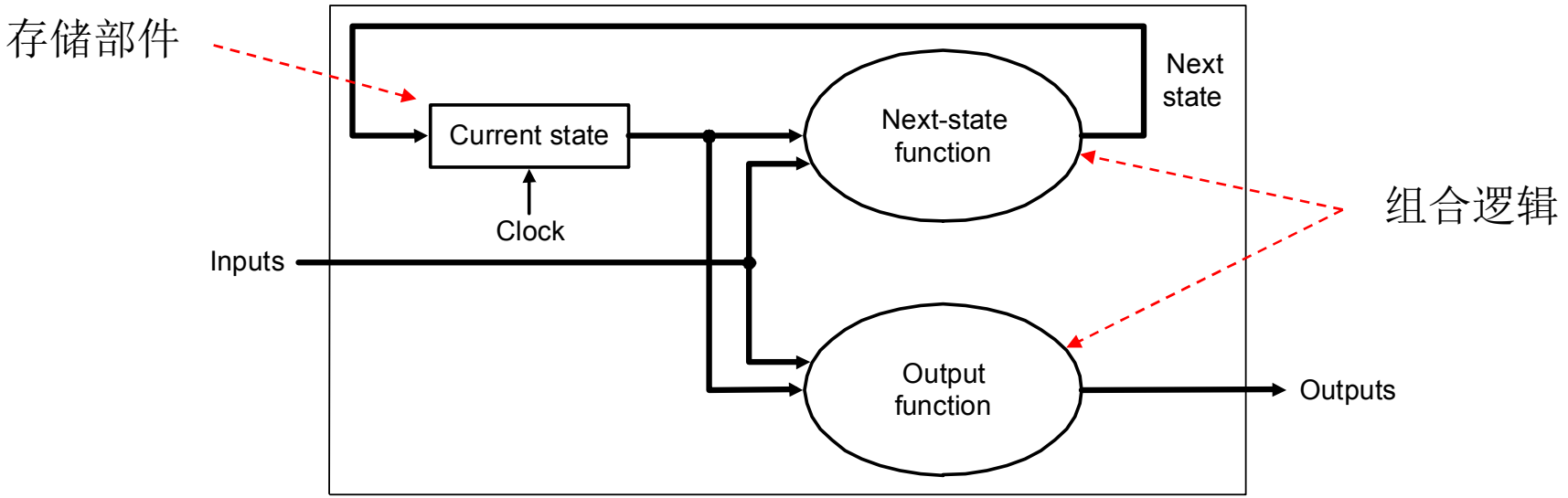
Step	R-Type	lw/sw	beq/bne	j
IF	IR = Mem[PC] PC = PC + 4			
ID	A = Reg[IR[25-21]] B = Reg[IR[20-16]] ALUOut = PC + (SE(IR[15-0]) << 2)			
EX	ALUOut = A op B	ALUOut = A +SE(IR[15-0])	If (A==B) then PC = ALUOut	PC = PC[31-28] || (IR[25-0]<<2
MEM	Reg[IR[15-11]] = ALUOut	MDR=Mem[ALUOut] Mem[ALUOut] = B		
WB		Reg[IR[20-16]] = MDR		

控制部件实现

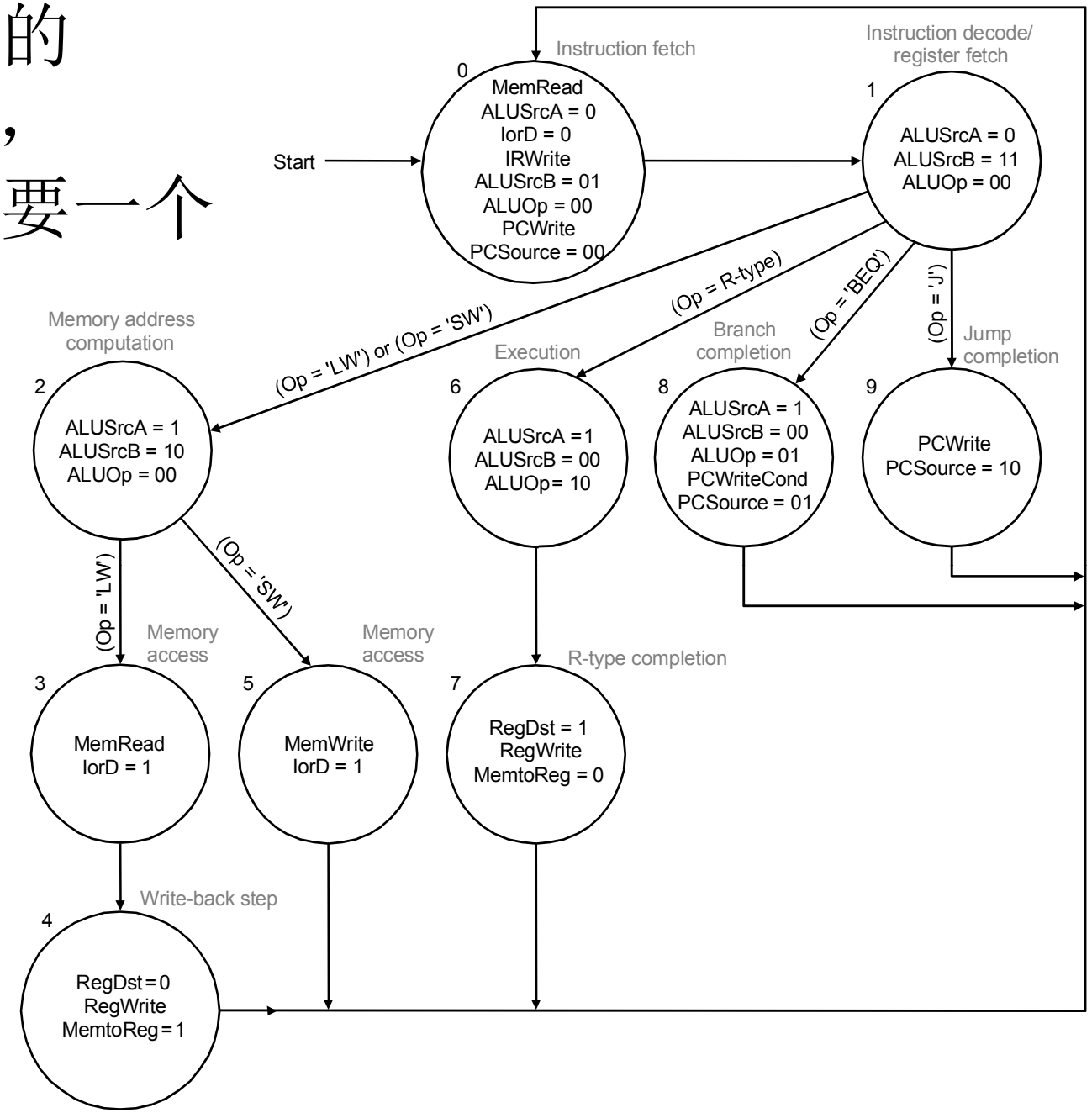

DATAPATH
MUX
din
0
din
1
dout
sel
din
ALU
din
dout
ALUop
REGISTER
DFF
DFF
DFF
DFF
din
dout
wrR clk
addr
din
DATA MEMORY
dout
rdM wrM clk
init
PC
addr
wrPC clk
addr
INSTRUC. REGISTER
inst
wrIR clk
sel ALUop wrR rdM wrM wrPC wrIR
opcode
clk
rst
Control Unit (FSM)

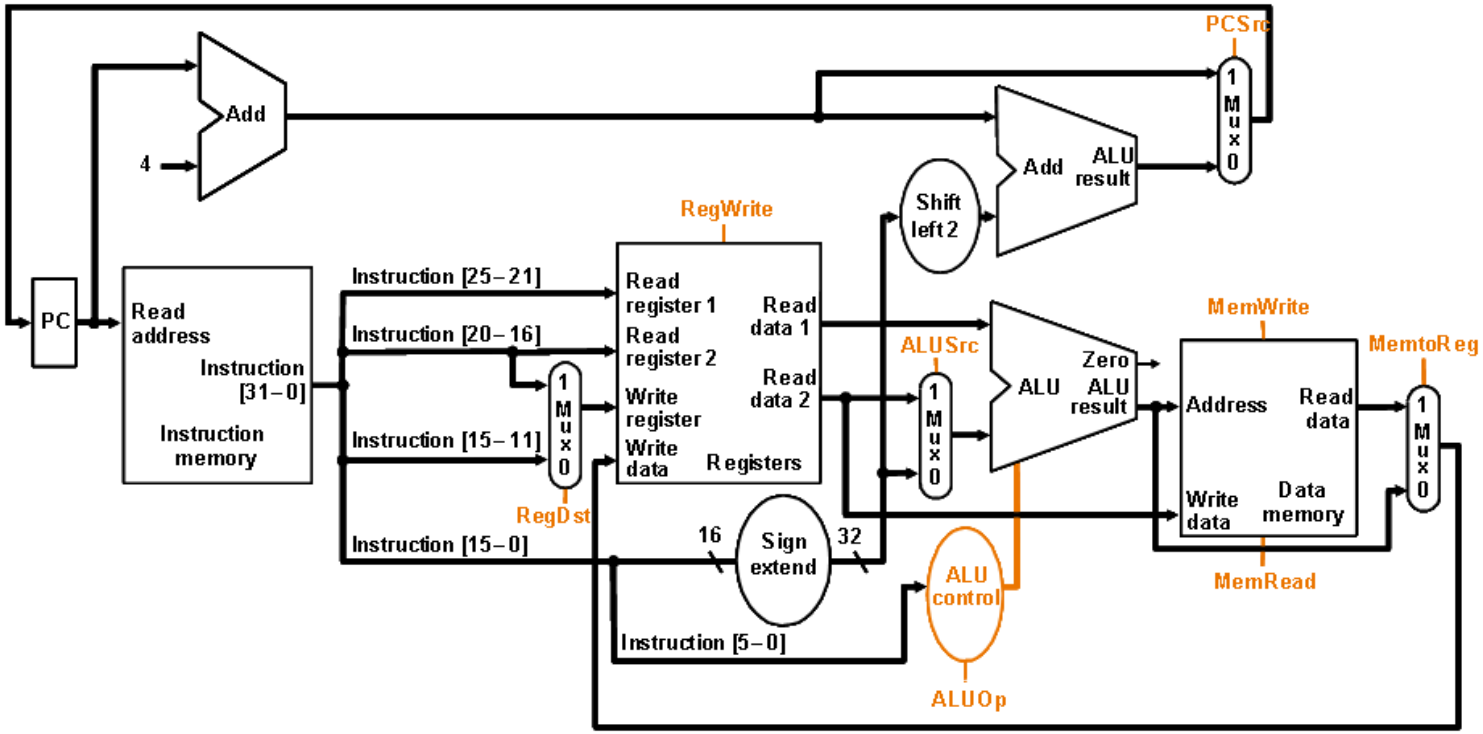
FSM控制部件实现

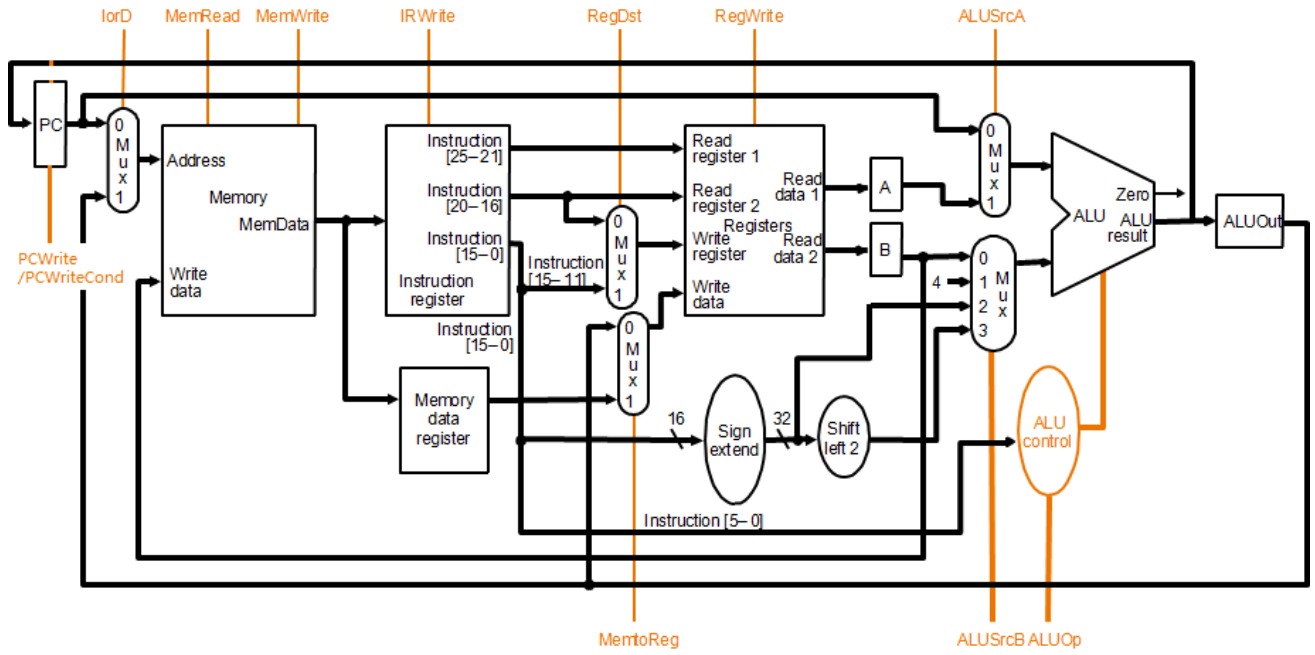
- Moore型（Edward Moore），Mealy型（George Mealy）
  - Moore型速度快（输出与输入无关，可以在周期一开始就提供控制信号）。一步延迟。输出与时钟完全同步。
  - Mealy型电路较小。输出与时钟不完全同步（多值，一个tick内随输入而变）。
  - 两种状态机可以相互转换。
- EDA工具可以根据FSM自动综合生成控制器



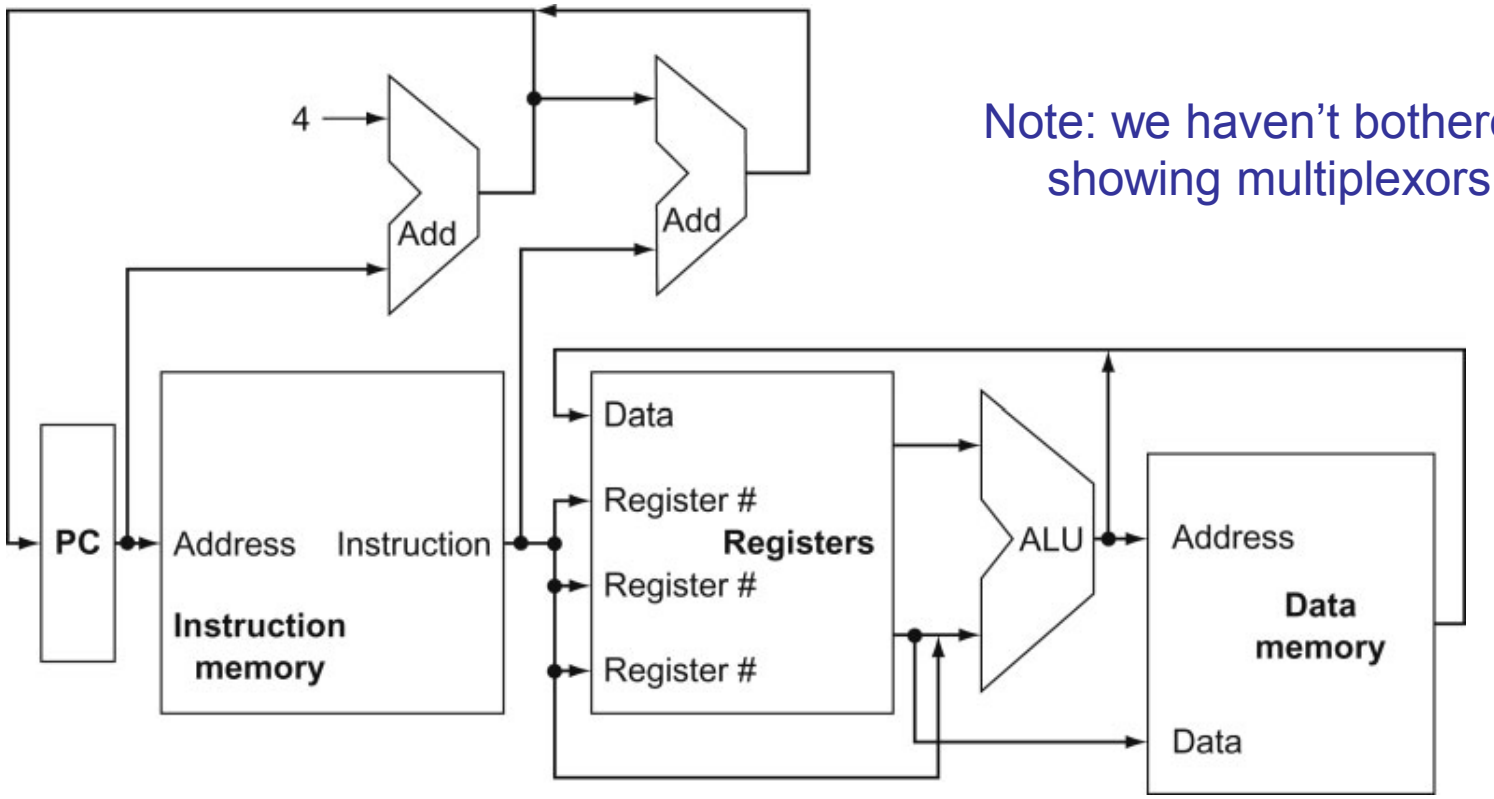
多周期控制的MooreFSM，每个状态需要一个周期。




PCSrc
Add
4
1
M
u
x
0
Add
ALU
result
Shift
left 2
RegWrite
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
RegDst
ALUSrc
ALU
Zero
ALU
result
MemWrite
MemtoReg
Address
Read
data
Write
data
Data
memory
MemRead
16
Sign
extend
32
ALU
control
ALUOp


IorD
MemRead
MemWrite
IRWrite
RegDst
RegWrite
ALUSrcA
PC
PCWrite
/PCWriteCond
0
M
u
x
1
Address
Memory
MemData
Write
data
Instruction
[25–21]
Instruction
[20–16]
Instruction
[15–0]
Instruction
register
Instruction
[15–0]
Memory
data
register
Instruction
[15–11]
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
0
1
2
3
M
u
x
ALU
Zero
ALU
result
ALUOut
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5–0]
MemtoReg
ALUSrcB
ALUOp

View from 30,000 Feet



Note: we haven't bothered showing multiplexors

Source: H&P textbook

- What is the role of the Add units?
- Explain the inputs to the data memory unit
- Explain the inputs to the ALU
- Explain the inputs to the register unit

小结

- 单周期与多周期
  - 单周期：在一个周期内完成指令的所有操作
    - 周期宽度如何确定？
  - 多周期：一个周期完成指令的一步
    - 与单周期的区别：功能部件复用，中间结果保存
    - 何时刷新PC：指令周期中PC保持不变如何实现？

- 作业：

  1. 4.1，4.9
  2. 分析MIPS三种类型指令的多周期设计方案中每个周期所用到的功能部件。


Thank You