

计算机组成原理 (2014级)

计算机组成原理课程组

(刘旭东、肖利民、牛建伟、栾钟治)

Tel : 82316285

Mail: liuxd@buaa.edu.cn

liuxd@act.buaa.edu.cn

第六讲 MIPS处理器设计

一 . 处理器设计概述

1. 处理器功能与组成
2. 处理器设计一般方法

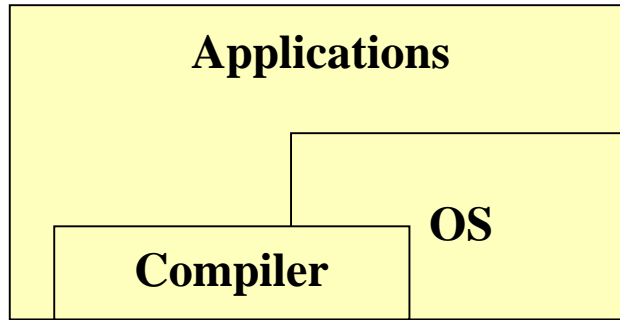
二 . MIPS模型机

三 . MIPS单周期处理器设计

四 . MIPS多周期处理器设计简介

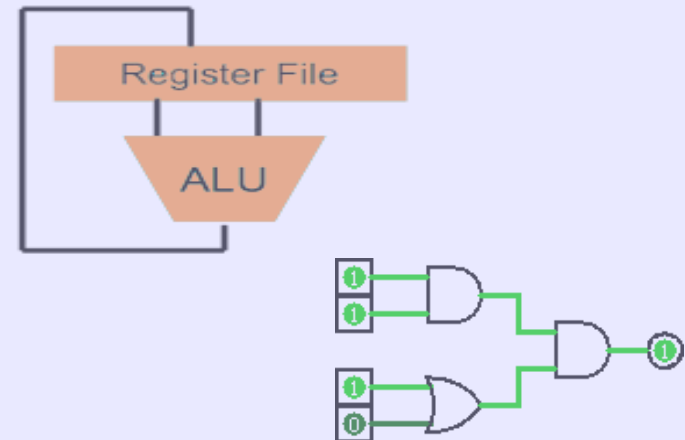
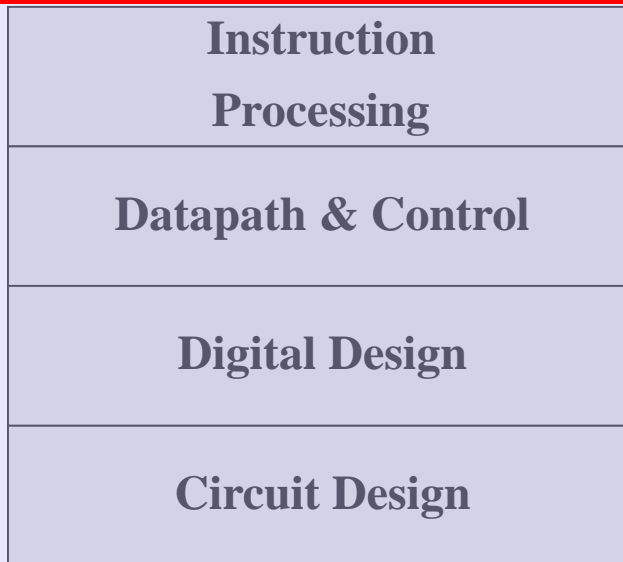
五 . MIPS流水线处理器设计

Software
layers of
abstraction



Instruction Set Architecture (**ISA**)

Hardware
layers for
design
abstraction



1.1 CPU的功能与组成

❖ CPU的功能：控制指令执行

❖ 指令的四种基本操作

- 取数：读取某主存单元的数据，并传送至某个寄存器；
- 存数：将某个寄存器中的数据存入主存某个单元之中；
- 传送：将某个寄存器中的数据传送至**ALU**或另一个寄存器；
- 运算：进行某种算术或逻辑运算，结果保存到某个寄存器中。

❖ 指令执行周期（一般性概念）：**CPU**从指令存储器中读出并执行指令功能的全部时间称为指令周期。包括：

- 取指周期：完成取指令操作和分析指令操作所需时间；
- 取数周期：从数据存储器读出操作数所需时间（包括计算操作数有效地址）；
- 执行周期：完成指令所规定的动作（运算）所需要时间，因指令不同而不同。

1.1 CPU的功能与组成

❖CPU所需的功能部件

- 取指令：从存储器中读出指令和分析指令（译码）
 - 指令地址部件：指明当前要读取的指令在存储器中的地址
 - 指令寄存部件：保存从存储器中取来的指令
 - 译码部件：对指令进行译码
- 执行指令：实现指令应该具有的操作功能（包括取数和执行）。
 - 执行部件：**ALU**、寄存器、数据存储器等等
 - 控制信号逻辑部件：根据指令的操作性质和操作对象的地址（译码结果），在时序信号配合下，产生一系列的微操作控制信号，从而控制计算机的运算器、存储器或输入输出接口等部件工作，实现指令所表示的功能。

1.1 CPU的功能与组成

➤ CPU内部结构（内部单总线结构）

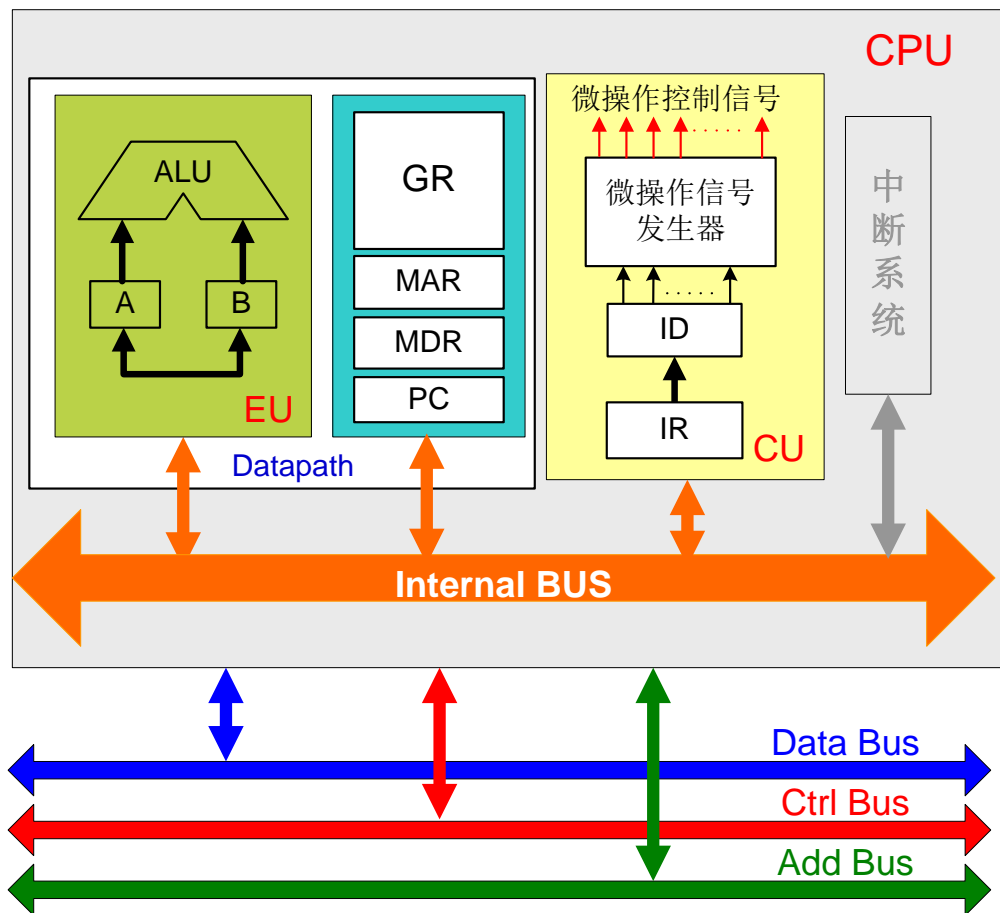
❑ 数据通路 (datapath)

- ◆ 运算单元
- ◆ 寄存器单元

❑ 控制器 (CU)

- ◆ 指令译码器ID
- ◆ 控制信号生成

❑ 内部总线



1.1 CPU的功能与组成

❖ CPU的组成

➤ 执行单元（数据通路，datapath）

- 运算单元：算术逻辑运算单元（ALU）
- 寄存器：通用寄存器组（GPRs），标志寄存器（FR，又称程序状态字PSW），临时寄存器（TR）

➤ 控制单元（控制器，control）：

- 指令地址部件：程序计数器（PC — Program Counter）
- 指令寄存部件：指令寄存器（IR — Instruction Register）
- 译码部件：指令译码器（ID — Instruction Decoder）
- 微操作控制信号产生部件：产生计算机其他部件所需要的所有微操作控制信号，有组合逻辑和微程序等实现方式。
- 时序部件：产生时序信号

1.1 CPU的功能与组成

❖ 数据通路

- 指令执行过程中，指令数据流所经过的部件和路径总称，用以实现数据的传送、处理和存储等功能，是**指令的执行部件**。
- 构成
 - 组合逻辑元件（操作元件）：**ALU**、译码器、多路选择器等
 - 存储元件（状态元件）：存储器、寄存器等
- 部件间连接方式
 - 总线连接方式（**CPU**内部总线）
 - 分散连接方式

❖ 控制器

- 对指令进行译码并生成指令执行所需的控制信号，以实现数据通路中各件的功能控制，以及相应路径的开关控制等，是**指令的控制部件**。

1.1 CPU的功能与组成

❖ 简单的数据通路示例

➤ 取指令路径

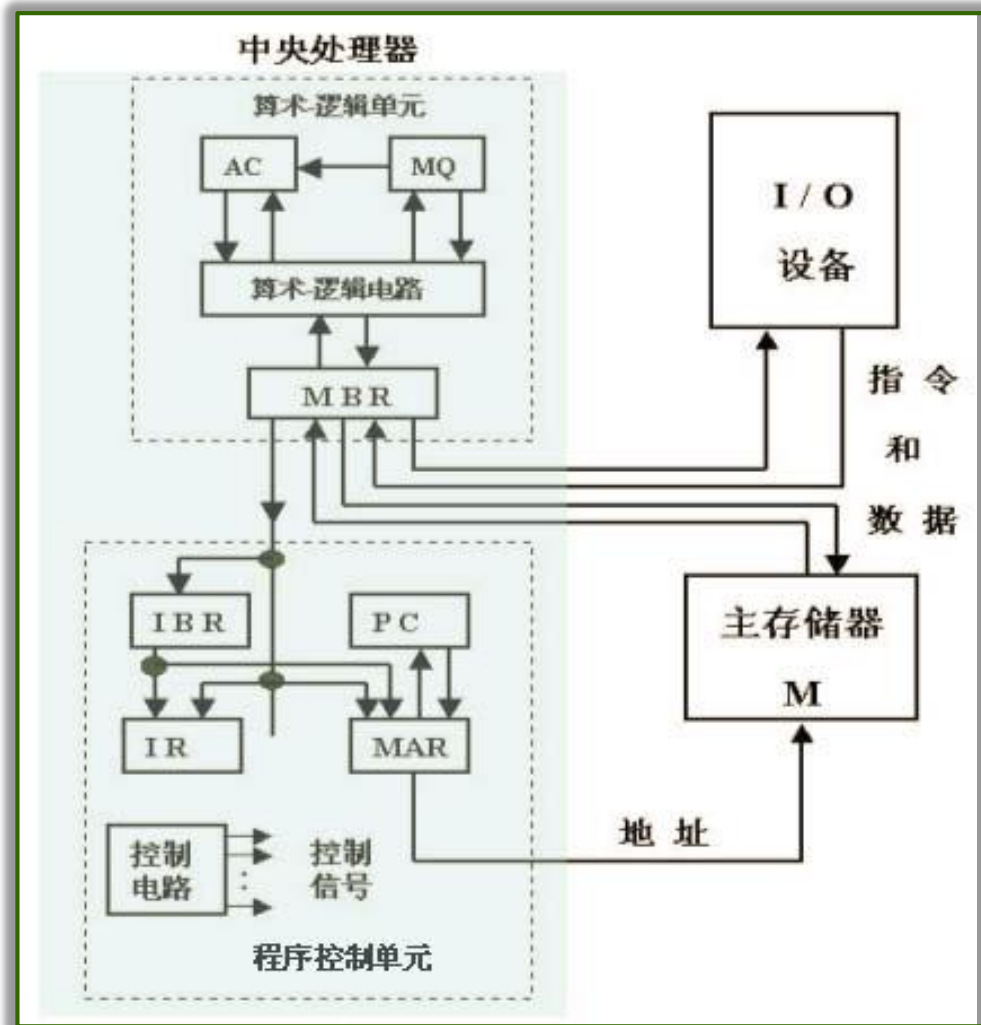
- PC→MAR
- Read Mem.
- M→MBR→IBR→IR

➤ 取操作数的路径

- 操作数地址→MAR
- Read Mem.
- M→MBR→ALU

➤ 运算结果保存路径

- ALU结果→MBR
- 结果地址→MAR
- Write Mem.



早期累加器型数据通路

1.1 CPU的功能与组成

❖ 单总线数据通路示例

➤ 取指令路径

- PC→IB→MAR
- MemR
- M→MER→IB→IR

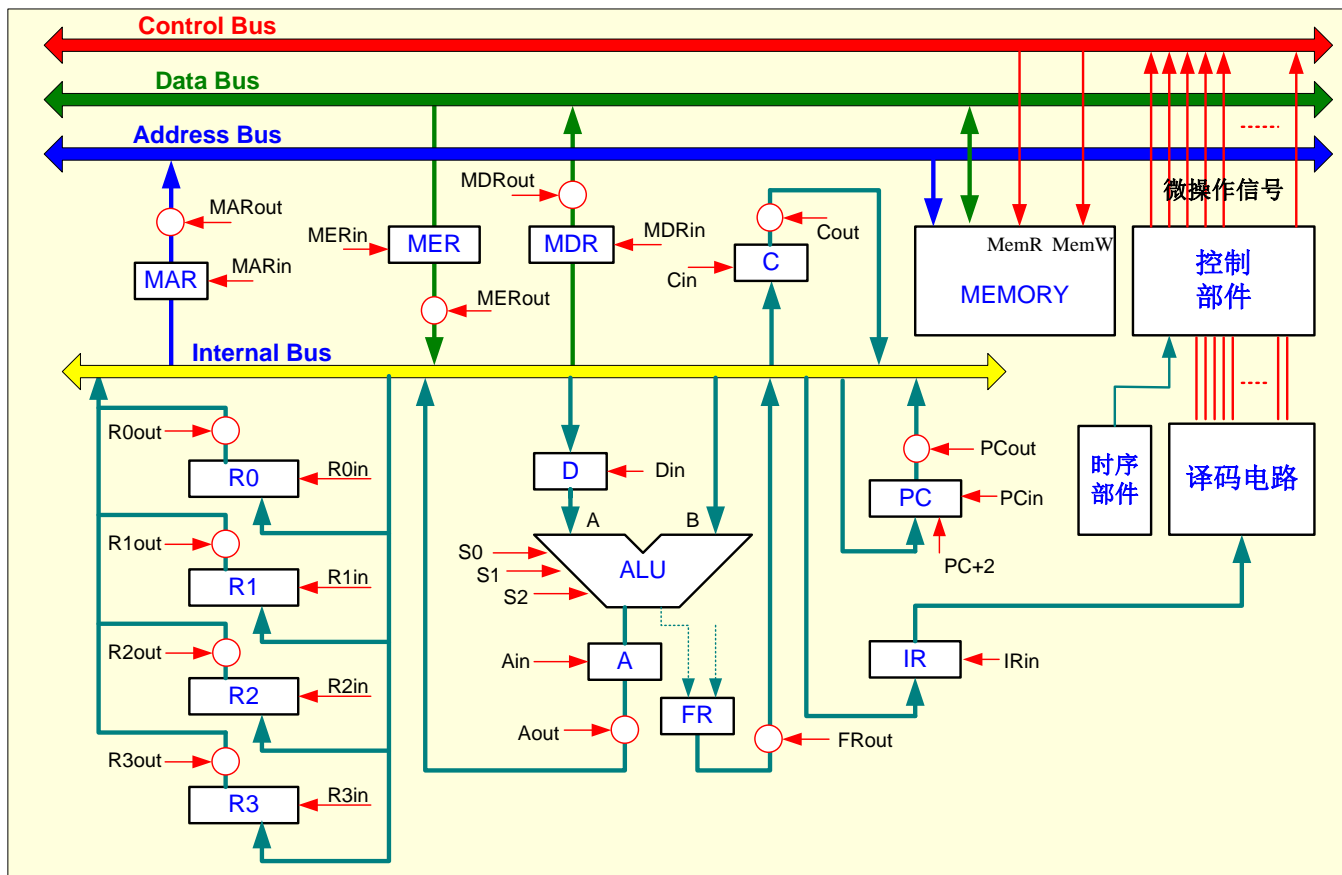
➤ 取操作数的路径

- 地址→IB→MAR
- MemR
- M→MER→IB→ALU

➤ 运算结果保存路径

- ALU结果→IB→MDR
- 结果地址→IB→MAR
- MemW

单总线
数据通路



1.1 CPU的功能与组成

❖ 指令功能的形式化描述：RTL（Register Transfer Language，寄存器传送语言）

- \leftarrow : 数据传送方向;
- $R[a]$: 寄存器 a ;
- $M[a]$: 主存中地址为 a 的单元;
- PC : 程序计数器
- $f(data)$: 表示对数据 $data$ 进行 f 操作

❖ 示例

- $R[c] \leftarrow R[a] + R[b]$ // 寄存器 a 加寄存器 b 的结果送寄存器 c
- $R[c] \leftarrow R[a] \text{ } op \text{ } R[b]$ // 寄存器 a 与寄存器 b 进行 op 运算结果送寄存器 c
- $Signext(imm16)$ // 对数 $imm16$ 进行 $Signext$ （符号扩展）运算
- $R[a] \leftarrow M[b]$ // 取数操作，读取主存单元 b 的数据传送至寄存器 a
- $M[a] \leftarrow R[b]$ // 存数操作，将寄存器 b 中的数据写入主存单元 a 中

1.3 处理器设计的一般方法

❖ 设计步骤

1. 分析指令系统需求：包括指令格式、指令类型、每种指令的功能、寻址方式等；
2. 数据通路构建
 - ① 根据指令需求选择数据通路部件，如**PC**、**ALU**、寄存器堆、指令/数据存储器、多路开关等等；
 - ② 根据指令执行流程构建每种类型指令的数据通路；
 - ③ 对所有类型指令执行数据通路综合形成综合数据通路。
3. 控制器设计
 - ① 确定控制器时序控制方式（单周期、多周期或其他）
 - ② 根据每种类型指令执行流程，确定该执行执行时各个数据通路部件所需要的控制信号与相应状态、条件；
 - ③ 对控制信号进行综合以得到每个控制信号的逻辑方程；
 - ④ 逻辑电路实现各个控制信号。

第六讲 MIPS处理器设计

一 . 处理器设计概述

二 . MIPS模型机

1. MIPS模型机指令集

2. MIPS模型机数据通路部件

3. 时钟同步方法

三 . MIPS单周期处理器设计

四 . MIPS多周期处理器设计简介

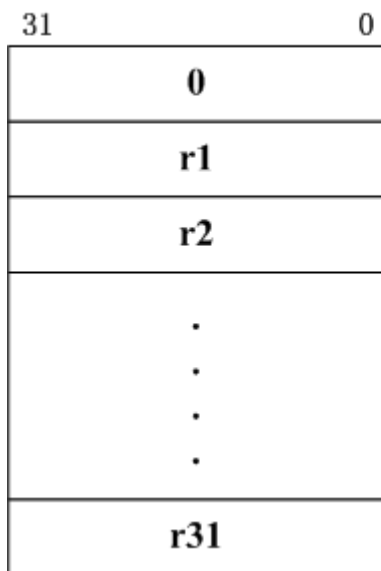
五 . MIPS流水线处理器设计

2.1 MIPS模型机指令集

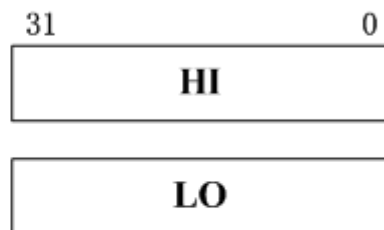
❖ MIPS 寄存器结构

- 32位虚拟地址空间
- 32个32位GPRs（通用寄存器）
- 32个32位FPRs（浮点数寄存器）
- HI, LO, PC

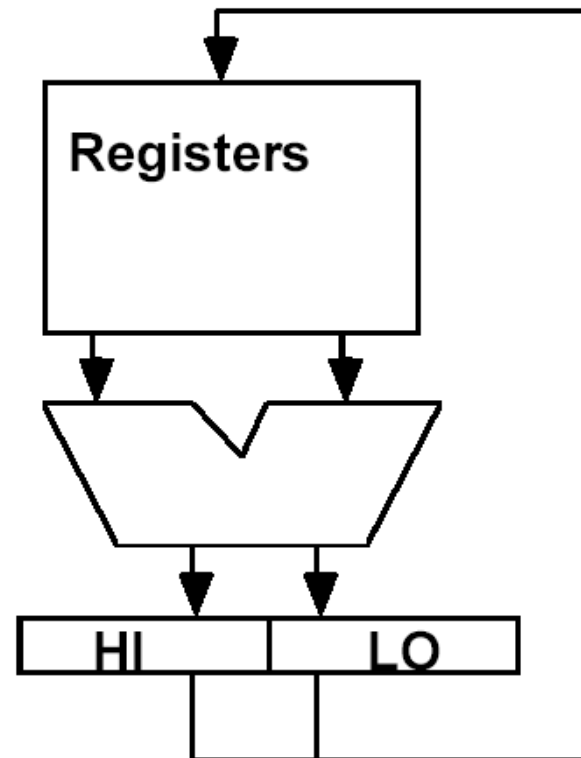
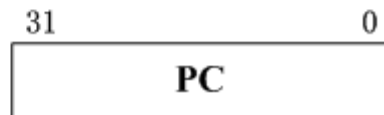
通用寄存器



乘/除寄存器



程序计数器



2.1 MIPS模型机指令集

MIPS 模型机寄存器使用约定

| 寄存器编号 | 寄存器名称 | 用途 |
|---------|-------------|-----------|
| 0 | \$zero | 常数0 |
| 1 | \$at | 保留给汇编器使用 |
| 2 ~ 3 | \$v0 ~ \$v1 | 结果值和表达式求值 |
| 4 ~ 7 | \$a0 ~ \$a3 | 参数 |
| 8 ~ 15 | \$t0 ~ \$t7 | 临时变量 |
| 16 ~ 23 | \$s0 ~ \$s7 | 数据寄存器 |
| 24 ~ 25 | \$t8 ~ \$t9 | 其他临时变量 |
| 26 ~ 27 | \$k0 ~ \$k1 | 保留给操作系统使用 |
| 28 | \$gp | 全局指针 |
| 29 | \$sp | 栈指针 |
| 30 | \$fp | 帧指针 |
| 31 | \$ra | 返回地址 |

2.1 MIPS模型机指令集

❖ MIPS 指令格式

- Op: 6 bits, Opcdoe
- Rs: 5 bits, The first register source operand
- Rt: 5 bits, The second register source operand
- Rd: 5 bits, The register destination operand
- Shamt: 5 bits, Shift amount (shift instruction)
- Func: 6 bits, function code (another Opcode)

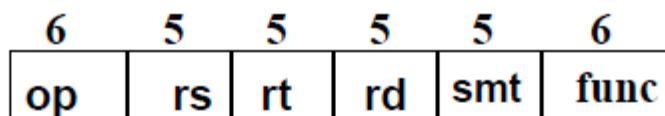
| | | | | | | |
|-----|---------------|--|---------------|---------------------------------------|-----------------|---------------|
| | 6 | 5 | 5 | 5 | 5 | 6 |
| R类型 | Op (31-26) | Rs (25-21) | Rt (20-16) | Rd (15-11) | Shamt (10-6) | Func (5-0) |
| I类型 | Op (31-26) | Rs (25-21) | Rt (20-16) | 16 bit Address or Immediate (15-0) | | |
| J类型 | Op (31-26) | 26 bit Address (for Jump Instruction) (25-0) | | | | |

2.1 MIPS模型机指令集

❖ MIPS 寻址方式

R-format:

Register (direct)



register

I-format:

Immediate



Base或index



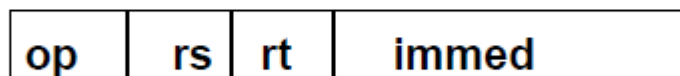
register

+

Memory

B/HW/W

PC-relative



PC + 4

+

Memory

J-format:

Pseudodirect



Memory

2.1 MIPS模型机指令集

模型机指令集（8条指令）

| 指令格式 | 指令 | 功能 | 说明 |
|---------|-------------------|---|--|
| R 类型 | add rd, rs, rt | $R[rd] \leftarrow R[rs] + R[rt]$ | 加运算： 寄存器 rs 和寄存器 rt 相加，结果送寄存器 rd |
| | sub rd, rs, rt | $R[rd] \leftarrow R[rs] - R[rt]$ | 减运算： 寄存器 rs 和寄存器 rt 相减，结果送寄存器 rd |
| | and rd, rs, rt | $R[rd] \leftarrow R[rs] \& R[rt]$ | 与运算： 寄存器 rs 和寄存器 rt 按位与，结果送寄存器 rd |
| | or rd, rs, rt | $R[rd] \leftarrow R[rs] R[rt]$ | 或运算： 寄存器 rs 和寄存器 rt 按位或，结果送寄存器 rd |
| I 类型 | lw rt, rs, imm16 | $Add = R[rs] + \text{Signext}(imm16)$ $R[rt] \leftarrow M[Add]$ | 取字： 寄存器 rs 和立即数 imm16（符号扩展至 32 位）相加得到内存地址，从内存该地址单元读取数据送 rt |
| | sw rt, rs, imm16 | $Add = R[rs] + \text{Signext}(imm16)$ $M[Add] \leftarrow R[rt]$ | 存字： 寄存器 rs 和立即数 imm16（符号扩展至 32 位）相加得到内存地址，寄存器 rt 数据写入内存该地址单元 |
| | beq rs, rt, imm16 | If ($R[rs] - R[rt] = 0$) then $PC \leftarrow PC + \text{Signext}(imm16) \ll 2$ | 分支： 如果寄存器 rs 与 rt 相等，则转移（imm16 符号扩展至 32 位），否则顺序执行。（取指令后，PC+4） |
| J 类型 | j target | $PC(31:2) \leftarrow \{ PC[31..28], \text{target address}, 00 \}$ | 跳转： 当前 PC 的高 4 位与 target（26 位）拼接成 30 位目标地址送 PC（31:2）。（取指令后，PC+4）。 |

2.1 MIPS模型机指令集

模型机指令编码

| R 类型格式 | OP (31 ~ 26) | Rs (25 ~ 21) | Rt (20 ~ 16) | Rd (15 ~ 11) | Shamt (10 ~ 6) | Funct (5 ~ 0) |
|----------------|-----------------|-----------------|-----------------|-----------------|-------------------|------------------|
| add rd, rs, rt | 000000 | rs | rt | rd | XXXXXX | 100000 |
| sub rd, rs, rt | 000000 | rs | rt | rd | XXXXXX | 100010 |
| and rd, rs, rt | 000000 | rs | rt | rd | XXXXXX | 100100 |
| or rd, rs, rt | 000000 | rs | rt | rd | XXXXXX | 100101 |

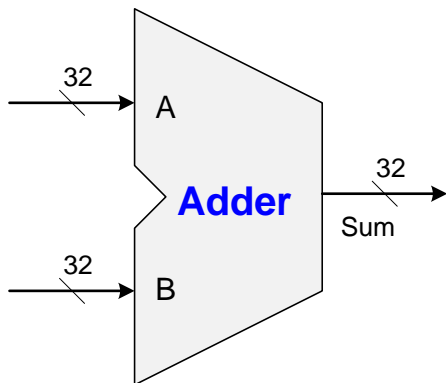
| I 类型格式 | OP (31 ~ 26) | Rs (25 ~ 21) | Rt (20 ~ 16) | 16 bits immediate or address (15 ~ 0) |
|-------------------|-----------------|-----------------|-----------------|--|
| lw rt, rs, imm16 | 100011 | rs | rt | imm16 |
| sw rt, rs, imm16 | 101011 | rs | rt | imm16 |
| beq rs, rt, imm16 | 000100 | rs | rt | imm16 |

| J 类型格式 | OP (31 ~ 26) | 26 bits address |
|----------|-----------------|-----------------|
| j target | 000010 | target |

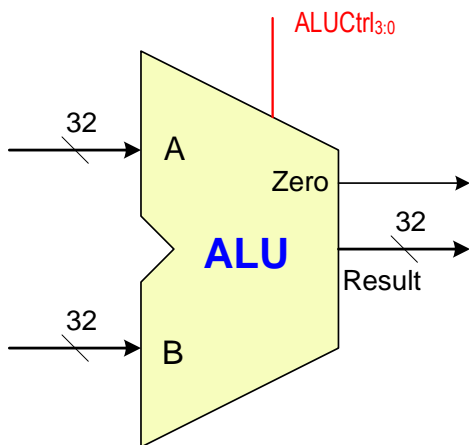
2.2 数据通路部件

❖ 组合部件:

➤ 加法器 (Adder) 、算术逻辑运算单元 (ALU)



| 输入 (32位) | | 输出 (32位) |
|----------|---|----------|
| A | B | Sum=A+B |

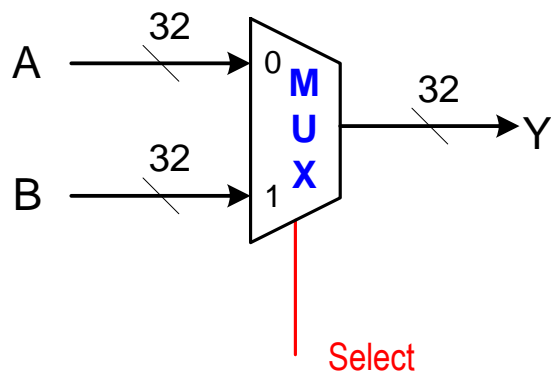


| 数据输入 (32位) | | 控制输入 ALUCtrl (4位) | 输出 |
|------------|---|-------------------|----------------------|
| A | B | 0000 | Result = A&B |
| A | B | 0001 | Result = A B |
| A | B | 0010 | Result = A+B |
| A | B | 0110 | Result = A-B |
| A | B | 0111 | If A=B then Result=1 |

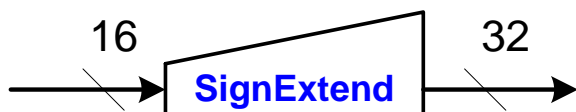
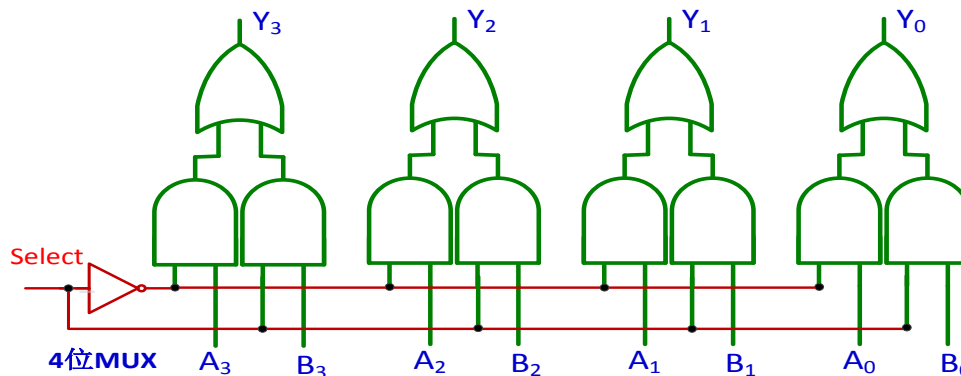
2.2 数据通路部件

❖ 组合部件

- 多路选择器 (Mux)、符号扩展器 (Signext)
- 多路选择器有二选一，三选一，四选一等。



| 输入 (32位) | | 控制输入Select | 输出 (32位) |
|----------|---|------------|----------|
| A | B | 无效 (低电平) | $Y=A$ |
| A | B | 有效 (高电平) | $Y=B$ |

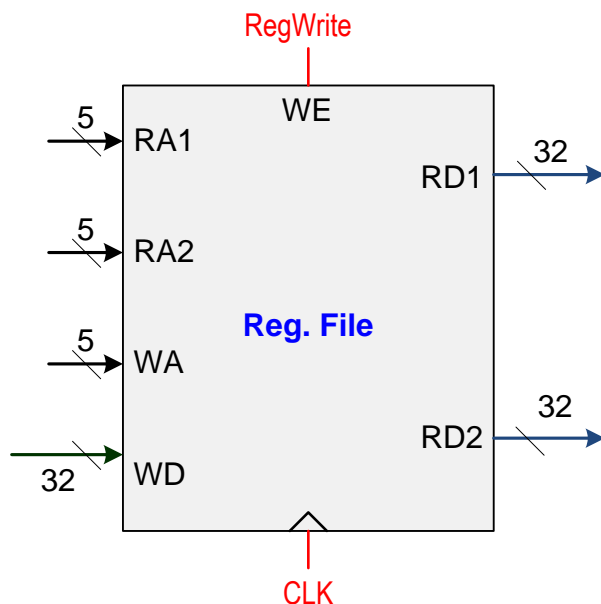


16位二进制数按符号位扩展至32位
如: $\text{Signext}(8000\text{H}) = \text{FFFF}8000\text{H}$
 $\text{Signext}(7800\text{H}) = 00007800\text{H}$

2.2 数据通路部件

❖ 寄存器堆（32个寄存器）

- 两个32位输出端口
- 一个32位输入端口



寄存器堆读操作（输出）

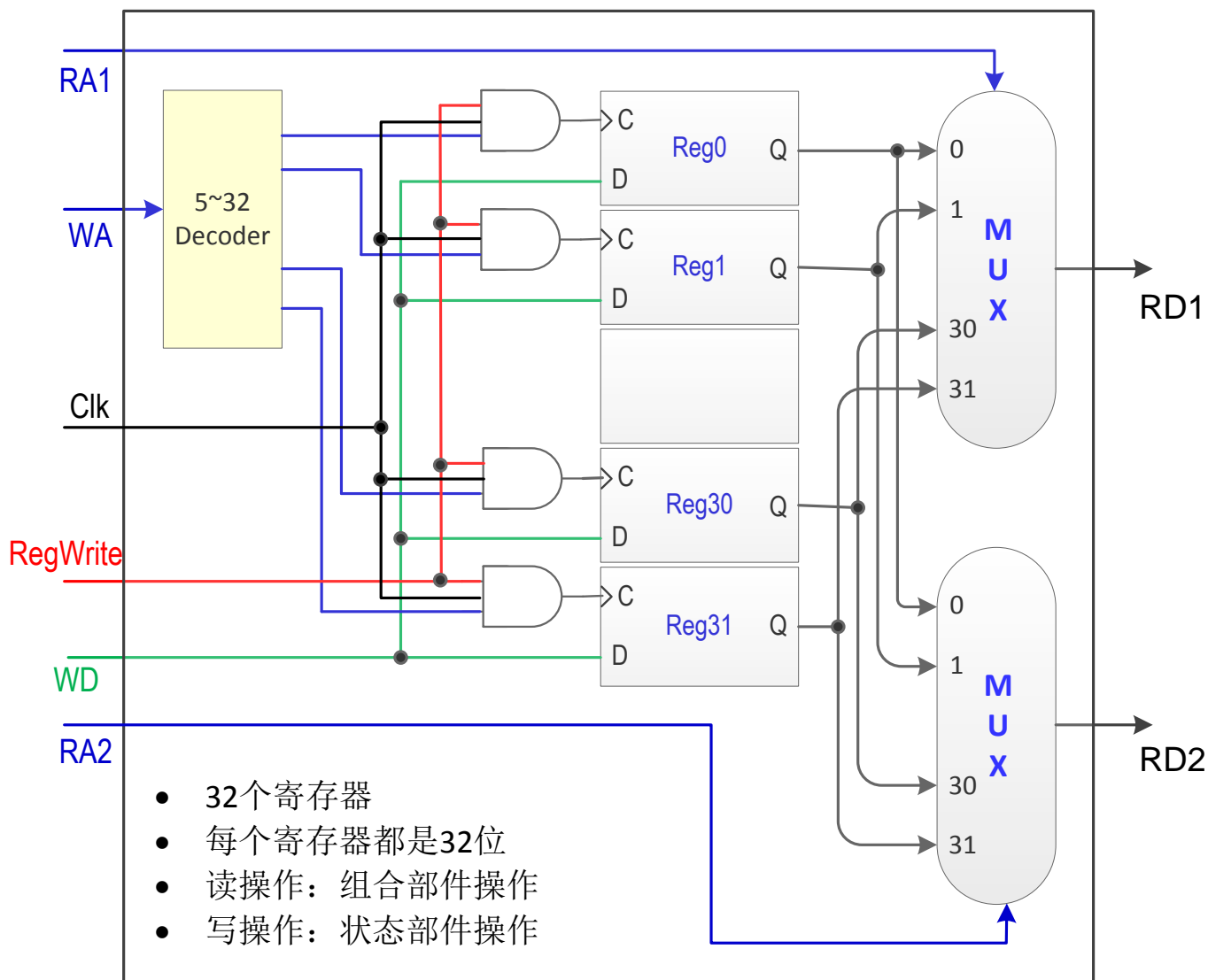
| 地址输入（5位） | | 数据输出 |
|----------|-----|----------------------------|
| RA1 | RA2 | RD1=RF[RA1] RD2=RF[RA2] |

寄存器堆写操作（输入）

| 地址输入（5位） | 数据输入（32位） | 写入使能控制 RegWrite | 操作 |
|----------|-----------|--------------------------|------------------------|
| WA | WD | 无效 | 无 |
| WA | WD | 有效 (Clk 时钟跳沿) | $RF[WA] \leftarrow WD$ |

2.2 数据通路部件

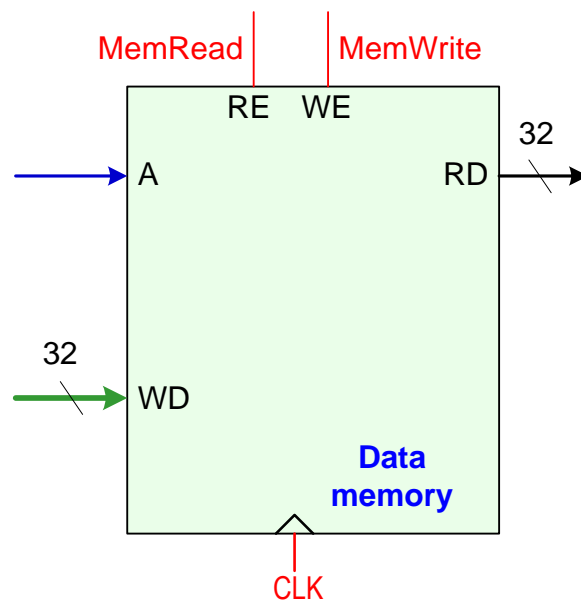
❖ 寄存器堆内部结构



2.2 数据通路部件

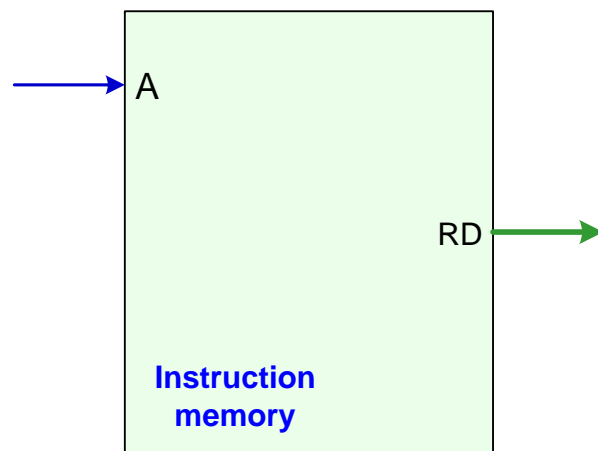
❖ 数据存储器DM（理想存储器）

- 单输入总线：WD（Write Data）
- 单输出总线：RD（Read data）
- 读：MemRead控制信号有效时，地址线A（Address）选择的存储字被放在RD输出总线上（组合元件操作）
- 写：MemWrite控制信号有效且时钟信号CLK跳沿时，WD总线上的数据被写入地址A选择的存储单元中（状态元件操作）



❖ 指令存储器IM

- 读取指令地址A（Address）选择的指令送数据输出线RD上（组合元件操作）

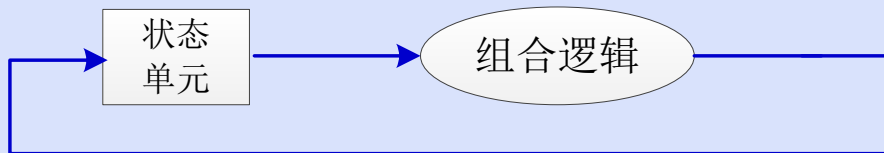


2.3 时钟同步方法

❖ 时钟同步方法

- 以时钟周期信号为基准，确定数据读出和写入的时刻。
- 采用边沿触发的时钟同步方法，如下跳沿触发，意味着所有状态元件（寄存器、存储器）的数据写入都发生在时钟周期的下跳沿时刻。

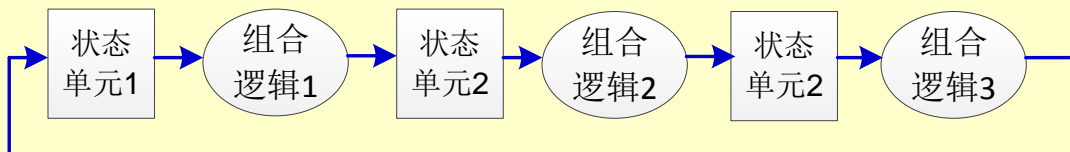
时钟



单周期通路同步

- 组合逻辑操作时钟周期内完成；
- 信号在时钟周期内从状态单元输出经组合逻辑再回到状态单元输入。
- 时钟信号上跳沿同步

时钟

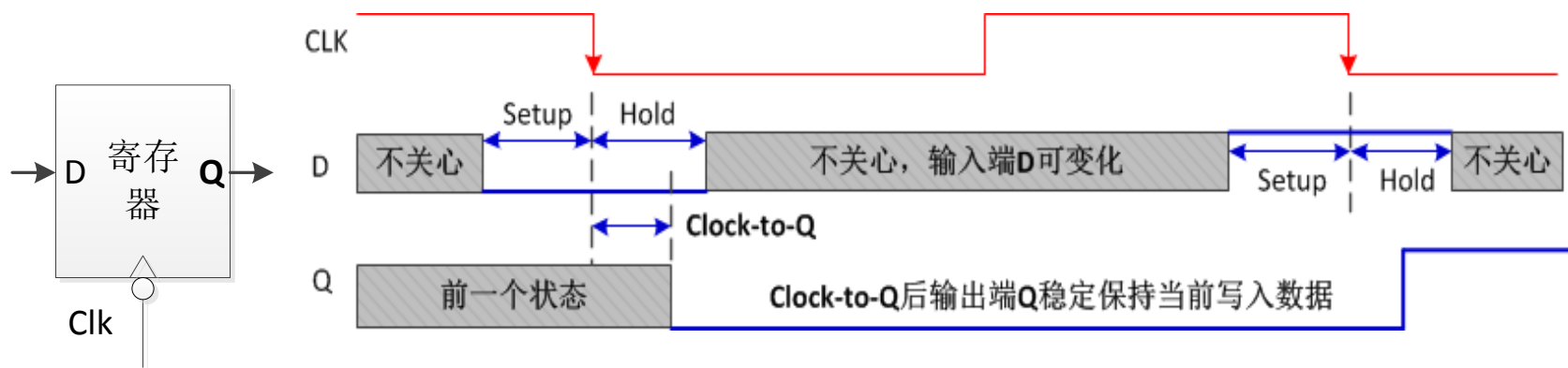


多周期通路同步

- 组合逻辑操作时钟周期内完成；
- 所有信号在时钟周期内从状态单元1经组合逻辑传送到状态单元2。
- 时钟信号上跳沿同步

2.3 时钟同步方法

❖ 寄存器（状态单元）的时序



建立时间 T_{setup} (Setup Time)：触发时钟边沿之前输入必须稳定的时间；

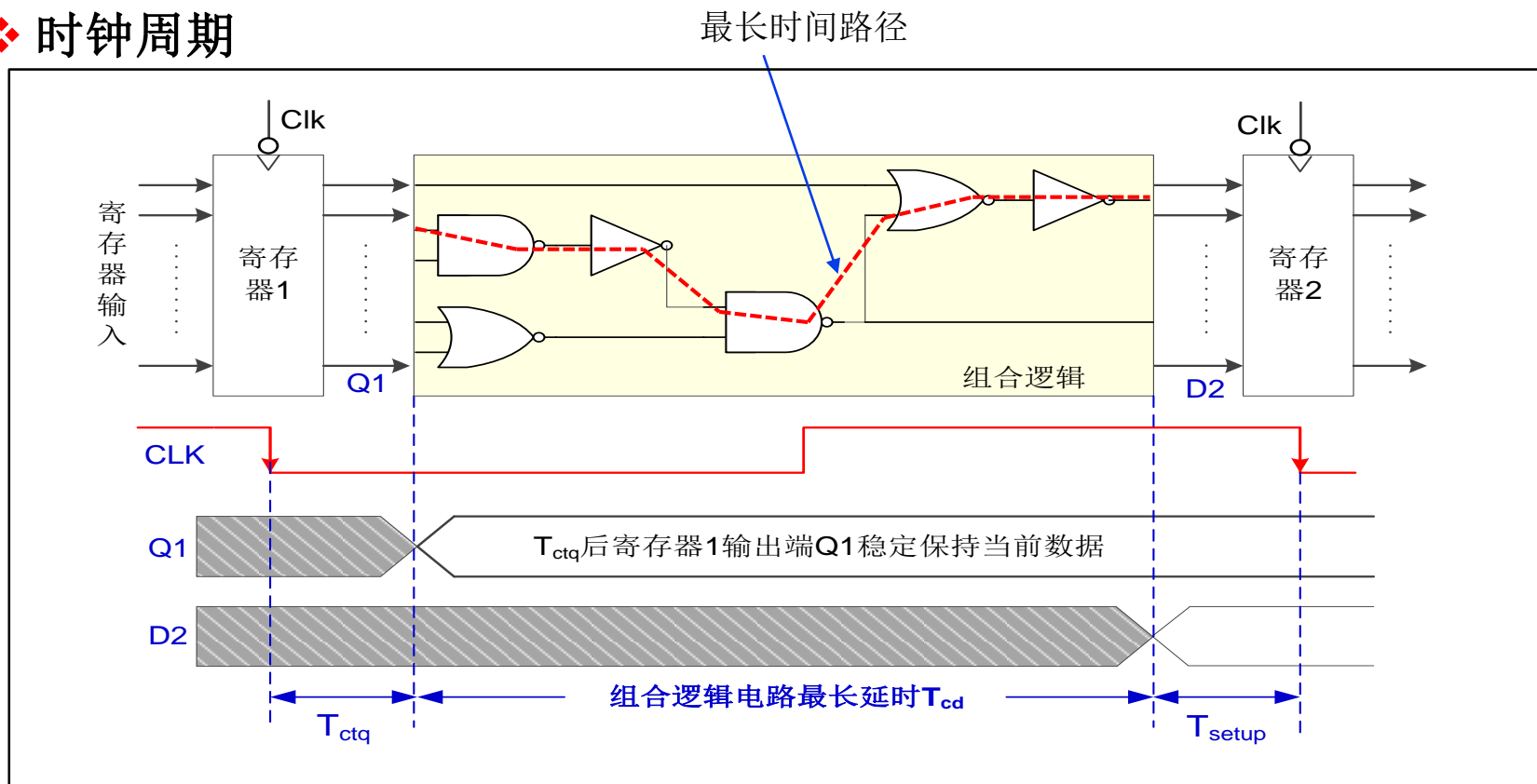
保持时间 T_{hold} (Hold Time)：触发时钟边沿之后输入仍需稳定的时间；

Clock-to-Q时间 T_{ctq} ：从触发时钟边沿到输出稳定的时间。

孔径时间 $= T_{\text{setup}} + T_{\text{hold}}$ ，输入信号在孔径时间内必须稳定不变。

2.3 时钟同步方法

❖ 时钟周期



- 时序电路由“寄存器元件+ 操作元件(组合电路) + 寄存器元件”组成
- 时钟周期 $T_C \geq T_{ctq} + T_{cd} + T_{setup} + \text{时钟偏移}$
- 时钟偏移：由于时钟信号源到各个寄存器部件的连线长度不同等原因所引起的各个寄存器时钟信号达到的细微时间差异，一般忽略。

第六讲 MIPS处理器设计

- 一 . 处理器设计概述
- 二 . MIPS模型机
- 三 . MIPS单周期处理器设计**
 - 1. 单周期数据通路设计**
 - 2. 单周期控制器设计**
 - 3. 单周期性能分析**
- 四 . MIPS多周期处理器设计简介
- 五 . MIPS流水线处理器设计

3.1 单周期数据通路设计

❖ 单周期

- 所有指令执行周期固定为单一时钟周期，**CPI=1**。

❖ 通路设计考虑

- **哈佛体系结构**：使用指令存储区（IM）和数据存储区（DM）分别保存指令和数据
- 先为每类指令设计独立的数据通路，然后再考虑数据通路合并。

❖ 指令执行的共性

- 根据PC从指令存储器读取指令，取指令后，**PC+4**；
- 模型机7条指令在读取寄存器后，都要使用ALU
 - **LW/SW**（存储访问）指令用ALU计算数据地址
 - **ADD/SUB/AND/OR**(算术逻辑)指令用ALU完成算术逻辑运算
 - **BEQ**（分支）指令用ALU进行比较（减法运算）

3.1 单周期数据通路设计

❖ 分析指令执行步骤，确定数据通路所需部件和部件间连接

➤ 模型机指令执行过程一般会分为如下几个步骤：

- 取指令：根据PC访问指令存储器获得指令，然后PC+4；
- 读寄存器：根据指令格式读取相应寄存器操作数
- ALU运算：在ALU完成相应的算术逻辑运算
- 数据存取：LW/SW指令的数据存储器访问
- 写寄存器：运算类指令和LW指令要把数据写入寄存器

❖ 使用数据通路设计表格

- 表格记录数据通路部件的
- 暂不考虑控制信号

| 指令 | Adder | | PC | IM.A | Reg. File | | | | ALU | | DM | |
|----|-------|---|----|------|-----------|-----|----|----|-----|---|----|----|
| | A | B | | | RA1 | RA2 | WA | WD | A | B | A | WD |
| | | | | | | | | | | | | |
| | | | | | | | | | | | | |

3.1 单周期数据通路设计——取指与PC自增

1. 取指和PC自增数据通路（所有指令）

➤ 功能描述

■ 取指: $\text{IM Address} \leftarrow \text{PC}$, $\text{instruction} = \text{IM}[\text{PC}]$

■ PC自增: $\text{PC} \leftarrow \text{PC} + 4$

➤ 所需部件: PC, Adder（实现PC加4），指令存储器IM

| 指令 | Adder | | PC | IM.A | Reg. File | | | | ALU | | DM | |
|------|-------|---|-------|------|-----------|-----|----|----|-----|---|----|----|
| | A | B | | | RA1 | RA2 | WA | WD | A | B | A | WD |
| R型指令 | PC | 4 | Adder | PC | | | | | | | | |
| Lw | PC | 4 | Adder | PC | | | | | | | | |
| Sw | PC | 4 | Adder | PC | | | | | | | | |
| Beq | PC | 4 | Adder | PC | | | | | | | | |
| | | | | | | | | | | | | |

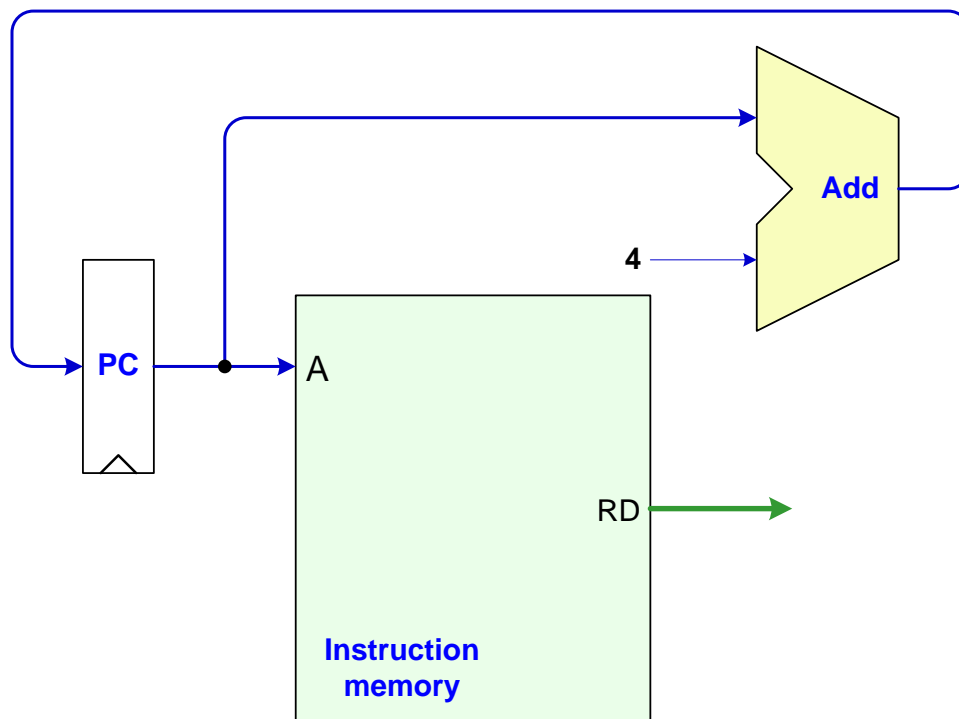
3.1 单周期数据通路设计——取指与PC自增

1. 取指和PC自增数据通路图

➤ 功能描述

- 取指: $\text{instruction} = \text{IM}[\text{PC}]$
- PC自增: $\text{PC} \leftarrow \text{PC} + 4$

➤ 所需部件: PC, Adder (实现PC加4), IM



3.1 单周期数据通路设计——R型指令数据通路

2. R型指令数据通路（add,sub,and,or指令，以add为例）

➤ add rd, rs, rt

➤ 功能描述

$$\text{RF}[\text{rd}] \leftarrow \text{RF}[\text{rs}] + \text{RF}[\text{rt}]$$

➤ 通路部件：寄存器堆、ALU

| | | | | | |
|---------------|---------------|---------------|---------------|-----------------|---------------|
| Op (31-26) | Rs (25-21) | Rt (20-16) | Rd (15-11) | Shamt (10-6) | Func (5-0) |
|---------------|---------------|---------------|---------------|-----------------|---------------|

| 指令 | Adder | | PC | IM.A | Reg. File | | | | ALU | | DM | |
|-------------|-------|---|-------|------|-----------|-----|----|-----|--------|--------|----|----|
| | A | B | | | RA1 | RA2 | WA | WD | A | B | A | WD |
| R型指令 | PC | 4 | Adder | PC | Rs | Rt | Rd | ALU | RF.RD1 | RF.RD2 | | |
| Lw | PC | 4 | Adder | PC | | | | | | | | |
| Sw | PC | 4 | Adder | PC | | | | | | | | |
| Beq | PC | 4 | Adder | PC | | | | | | | | |

- Rs=Instr(25:21)
- Rt=Instr(20:16)
- Rd=instr(15:11)

3.1 单周期数据通路设计——R型指令数据通路

2. R型指令数据通路（add,sub,and,or指令，以add为例）

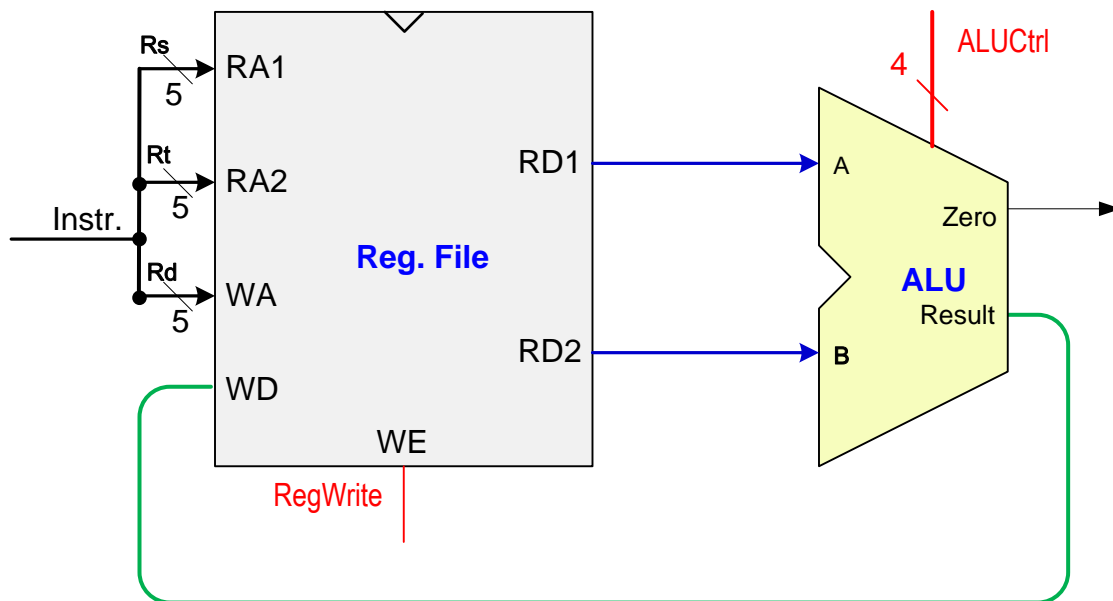
➤ **add rd, rs, rt**

➤ 功能描述

▪ $RF[rd] \leftarrow RF[rs] + RF[rt]$

➤ 通路部件：寄存器堆、ALU

| Op (31-26) | Rs (25-21) | Rt (20-16) | Rd (15-11) | Shamt (10-6) | Func (5-0) |
|---------------|---------------|---------------|---------------|-----------------|---------------|
|---------------|---------------|---------------|---------------|-----------------|---------------|



| 输入 | | ALU Ctrl | Result |
|----|---|----------|--------|
| A | B | 0000 | A & B |
| A | B | 0001 | A B |
| A | B | 0010 | A + B |
| A | B | 0110 | A - B |

3.1 单周期数据通路设计——LW指令数据通路

3. 取数指令（lw）数据通路

➤ lw rt, rs, imm16

➤ 功能描述：

▪ $RF[rt] \leftarrow DM[RF[rs] + \text{Signext}(imm16)]$

➤ 通路部件：寄存器堆，ALU，符号扩展单元Signext，数据存储器DM

| | | | |
|---------------|---------------|---------------|---------------------------------------|
| Op (31-26) | Rs (25-21) | Rt (20-16) | 16 bit Address or Immediate (15-0) |
|---------------|---------------|---------------|---------------------------------------|

| 指令 | Adder | | PC | IM.A | Reg. File | | | | ALU | | DM | | Sign-ext |
|------|-------|---|-------|------|-----------|-----|----|-------|--------|----------|-----|----|----------|
| | A | B | | | RA1 | RA2 | WA | WD | A | B | A | WD | |
| R型指令 | PC | 4 | Adder | PC | Rs | Rt | Rd | ALU | RF.RD1 | RF.RD2 | | | |
| Lw | PC | 4 | Adder | PC | Rs | | Rt | DM.RD | RF.RD1 | Sign-ext | ALU | | imm16 |
| Sw | PC | 4 | Adder | PC | | | | | | | | | |
| Beq | PC | 4 | Adder | PC | | | | | | | | | |

- Imm16=Instr(15:0)

3.1 单周期数据通路设计——LW指令数据通路

3. 取数指令（lw）数据通路

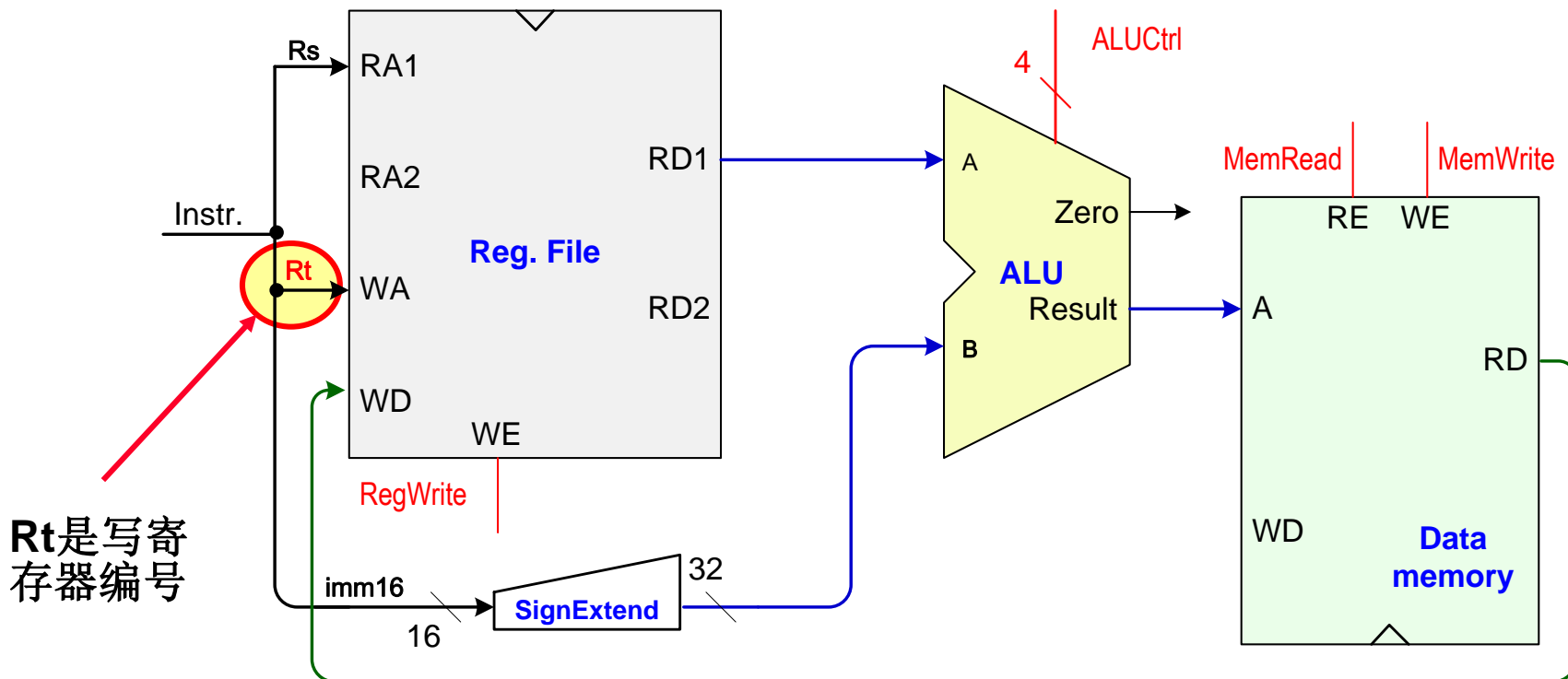
➤ **lw** *rt*, *rs*, *imm16*

➤ 功能描述：

▪ $RF[rt] \leftarrow DM[RF[rs] + \text{Signext}(imm16)]$

➤ 通路部件：寄存器堆，ALU，符号扩展单元Signext，数据存储器DM

| Op (31-26) | Rs (25-21) | Rt (20-16) | 16 bit Address or Immediate (15-0) |
|---------------|---------------|---------------|---------------------------------------|
|---------------|---------------|---------------|---------------------------------------|



Rt是写寄存器编号

3.1 单周期数据通路设计——SW指令数据通路

4. 存数指令（sw）数据通路

➤ **sw** rt, rs, imm16

| | | | |
|---------------|---------------|---------------|---------------------------------------|
| Op (31-26) | Rs (25-21) | Rt (20-16) | 16 bit Address or Immediate (15-0) |
|---------------|---------------|---------------|---------------------------------------|

➤ 功能描述：

▪ $DM[RF[rs] + Signext(imm16)] \leftarrow RF[rt]$

➤ 通路部件：寄存器堆，ALU，符号扩展单元Signext，数据存储器DM

| 指令 | Adder | | PC | IM.A | Reg. File | | | | ALU | | DM | | Sign-ext |
|-------------|-------|---|-------|------|-----------|-----------|----|-------|---------------|-----------------|------------|---------------|--------------|
| | A | B | | | RA1 | RA2 | WA | WD | A | B | A | WD | |
| R型指令 | PC | 4 | Adder | PC | Rs | Rt | Rd | ALU | RF.RD1 | RF.RD2 | | | |
| Lw | PC | 4 | Adder | PC | Rs | | Rt | DM.RD | RF.RD1 | Signext | ALU | | imm16 |
| Sw | PC | 4 | Adder | PC | Rs | Rt | | | RF.RD1 | Sign-ext | ALU | RF.RD2 | imm16 |
| Beq | PC | 4 | Adder | PC | | | | | | | | | |

3.1 单周期数据通路设计——SW指令数据通路

4. 存数指令（sw）数据通路

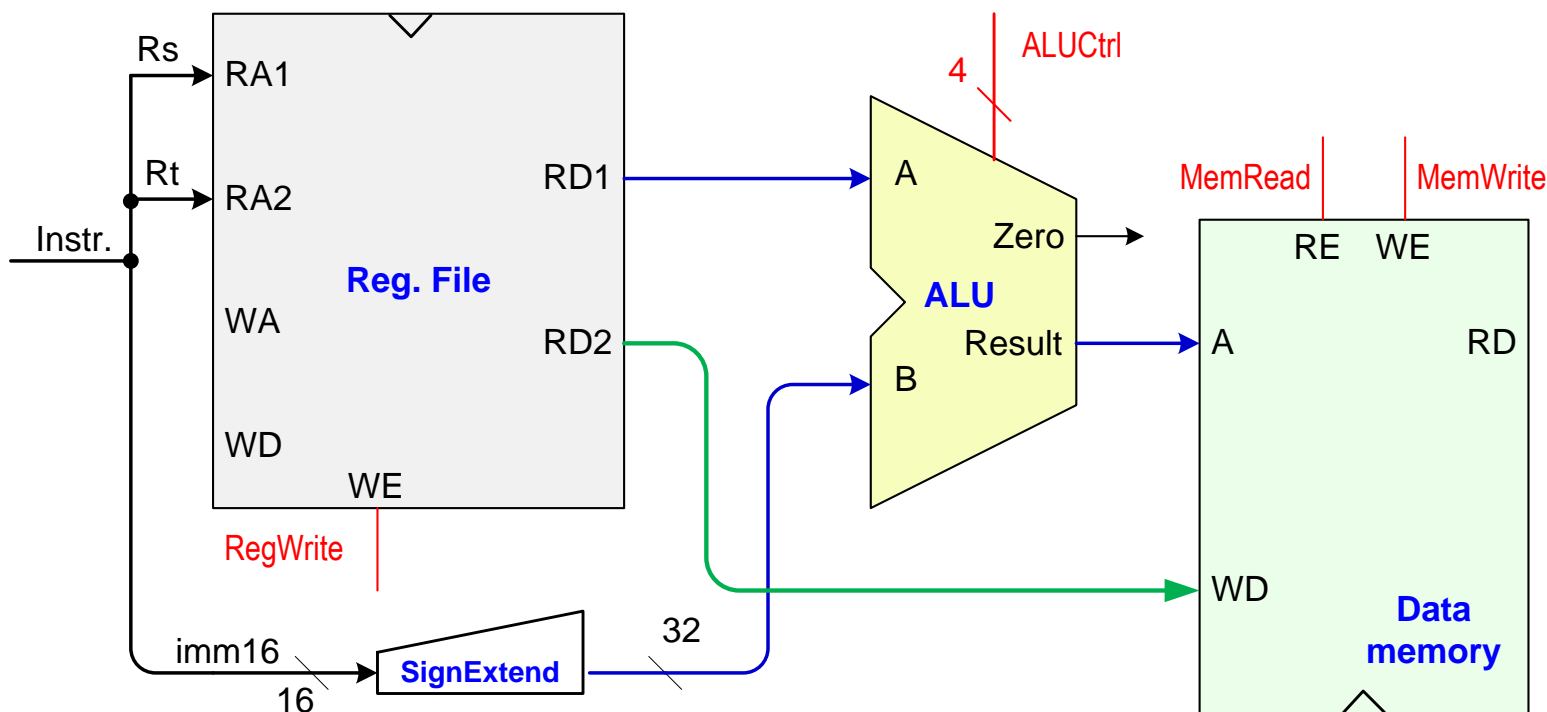
➤ **sw** rt, rs, imm16

➤ 功能描述：

■ $DM[RF[rs] + \text{Signext}(imm16)] \leftarrow RF[rt]$

➤ 通路部件：寄存器堆，ALU，符号扩展单元Signext，数据存储器DM

| Op (31-26) | Rs (25-21) | Rt (20-16) | 16 bit Address or Immediate (15-0) |
|---------------|---------------|---------------|---------------------------------------|
|---------------|---------------|---------------|---------------------------------------|



3.1 单周期数据通路设计——R型指令与访存指令通路合并

5. R型指令与访存指令数据通路合并：增加多路选择Mux

| | | | | | |
|---------------|---------------|---------------|---------------------------------------|-----------------|---------------|
| Op (31-26) | Rs (25-21) | Rt (20-16) | Rd (15-11) | Shamt (10-6) | Func (5-0) |
| Op (31-26) | Rs (25-21) | Rt (20-16) | 16 bit Address or Immediate (15-0) | | |

R型指令格式

LW/SW指令格式

| 指令 | Adder | | PC | IM.A | Reg. File | | | | ALU | | DM | | Sign-ext |
|-------------|-------|---|-------|------|-----------|-----|---------|-------------|--------|------------------|-----|--------|----------|
| | A | B | | | RA1 | RA2 | WA | WD | A | B | A | WD | |
| R型指令 | PC | 4 | Adder | PC | Rs | Rt | Rd | ALU | RF.RD1 | RF.RD2 | | | |
| Lw | PC | 4 | Adder | PC | Rs | | Rt | DM.RD | RF.RD1 | Signext | ALU | | imm16 |
| Sw | PC | 4 | Adder | PC | Rs | Rt | | | RF.RD1 | Signext | ALU | RF.RD2 | imm16 |
| 合并 | PC | 4 | Adder | PC | Rs | Rt | Rd Rt | ALU DM.RD | RF.RD1 | RF.RD2 Signext | ALU | RF.RD2 | imm16 |

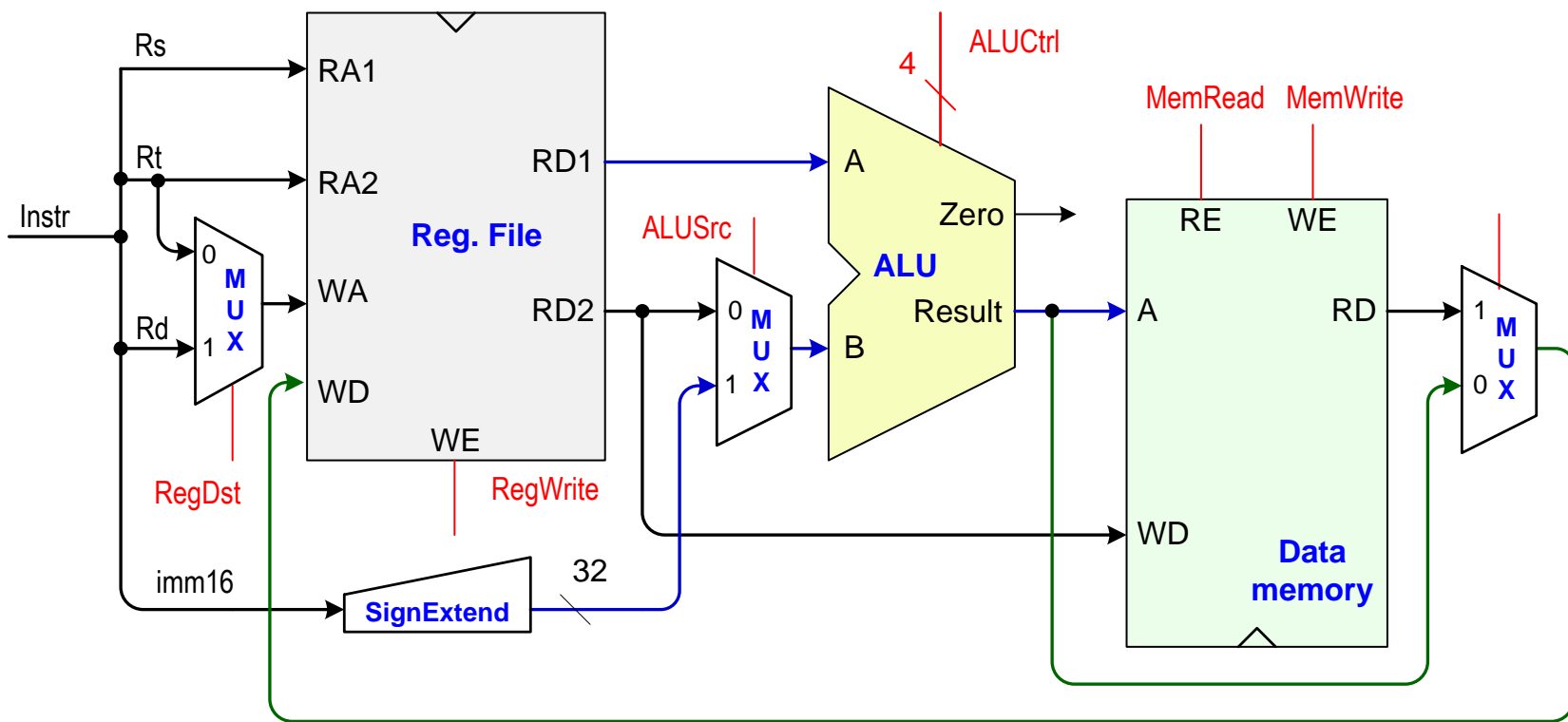
输入端数据源出现多个选择，
需要加入多路选择器MUX

3.1 单周期数据通路设计——R型指令与访存指令通路合并

5. R型指令与访存指令数据通路合并

➤ 增加3个二选一多路选择器MUX

- 寄存器堆写入端地址选择MUX，选择控制信号 **RegDst**
- ALU输入端B数据源选择MUX，选择控制信号 **ALUSrc**
- 寄存器堆写入端数据源选择MUX，选择控制信号 **MemtoReg**



3.1 MIPS的数据通路设计——Beq指令数据通路

6. 分支指令数据通路

➤ beq rs, rt, imm16

➤ 功能描述:

- If ($RF[rs] - RF[rt] = 0$) then $PC \leftarrow (PC + 4) + \text{Signext}(\text{imm16}) \ll 2$
else $PC \leftarrow PC + 4$

➤ 通路部件: 寄存器堆, ALU, 增加一加法器Nadd, 符号扩展
Signext, 移位器

| | | | |
|---------------|---------------|---------------|---------------------------------------|
| Op (31-26) | Rs (25-21) | Rt (20-16) | 16 bit Address or Immediate (15-0) |
|---------------|---------------|---------------|---------------------------------------|

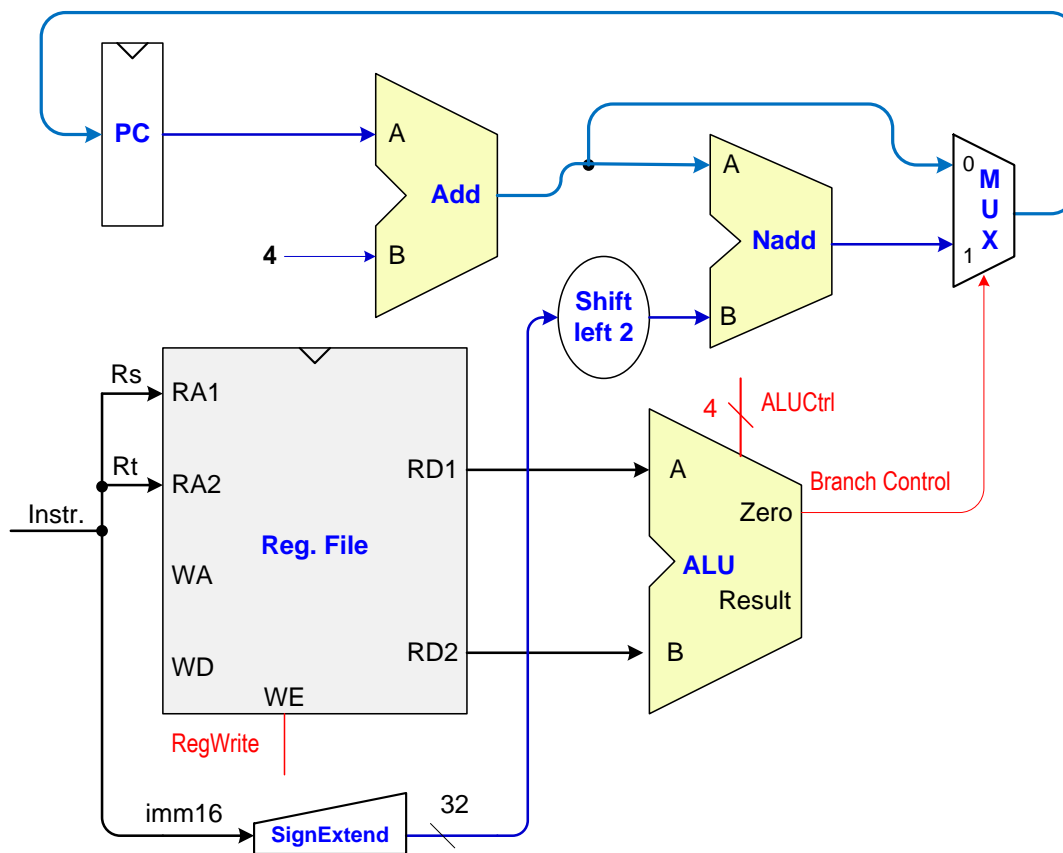
| 指令 | Adder | | PC | IM. A | Reg. File | | | | ALU | | DM | | Sign- ext | Nadd | |
|-------|-------|---|--------------|----------|-----------|-----|---------|-------------|--------|------------------|-----|--------|--------------|-------|-------|
| | A | B | | | RA1 | RA2 | WA | WD | A | B | A | WD | | A | B |
| R型与访存 | PC | 4 | Adder | PC | Rs | Rt | Rd Rt | ALU DM.RD | RF.RD1 | RF.RD2 Signext | ALU | Rdata2 | imm16 | | |
| Beq | PC | 4 | Adder Nadd | PC | Rs | Rt | | | RF.RD1 | RF.RD2 | | | imm16 | Adder | Shift |
| | | | | | | | | | | | | | | | |

需要一个MUX, ALU的判零输出端Zero可直接作为该MUX的选择控制

3.1 MIPS的数据通路设计——Beq指令数据通路

6. 分支指令数据通路

- beq rs, rt, imm16
- 通路部件：寄存器堆，ALU，增加一Adder，符号扩展Signext，移位器



3.1 单周期数据通路设计

7. MIPS数据通路再合并

➤ 支持：R类型指令、内存访问指令（lw/sw）、beq指令

| 指令 | Adder | | PC | IM. A | Reg. File | | | | ALU | | DM | | Sign- ext | Nadd | |
|-------|-------|---|--------------|----------|-----------|-----|---------|-------------|--------|------------------|-----|--------|--------------|-------|-------|
| | A | B | | | RA1 | RA2 | WA | WD | A | B | A | WD | | A | B |
| R型与访存 | PC | 4 | Adder | PC | Rs | Rt | Rd Rt | ALU DM.RD | RF.RD1 | RF.RD2 Signext | ALU | RF.RD2 | imm16 | | |
| Beq | PC | 4 | Adder Nadd | PC | Rs | Rt | | | RF.RD1 | RF.RD2 | | | imm16 | Adder | Shift |
| 合并 | PC | 4 | Adder Nadd | PC | Rs | Rt | Rd Rt | ALU DM.RD | RF.RD1 | RF.RD2 Signext | ALU | RF.RD2 | imm16 | Adder | Shift |

需要4个二选一多路选择器MUX

- PC输入端数据源选择MUX，选择控制信号 PCSrc
- 寄存器堆写入端地址WA选择MUX，选择控制信号 RegDst
- ALU输入端B数据源选择MUX，选择控制信号 ALUSrc
- 寄存器堆写入端数据WD选择MUX，选择控制信号 MemtoReg

7. MIPS数据通路合并

The diagram illustrates the internal structure and data flow of a processor, likely a MIPS-style architecture. Key components and their interactions are as follows:

- PC (Program Counter):** Receives the next instruction address from the MUX (Multiplexer) and outputs to the Instruction memory.
- Instruction memory:** Provides the instruction to the ALU and the Register File (RA1, RA2, WA, WD, WE, RD1, RD2).
- Register File:** Contains registers RA1, RA2, WA, WD, WE, RD1, and RD2. It receives register addresses from the ALU and the MUX, and outputs register values to the ALU and the MUX.
- ALU (Arithmetic Logic Unit):** Performs arithmetic and logical operations on data from the Register File and the MUX. It outputs the result to the MUX and the Data memory.
- Nadd (Next Addition):** A specialized ALU unit that performs the next addition operation, outputting the result to the MUX.
- Shift left 2:** A shift operation that shifts the result of the ALU operation left by 2 bits, outputting the result to the MUX.
- MUX (Multiplexer):** Selects between different data sources (PC, ALU result, Nadd result, Shift left 2 result) based on control signals (PCSrc, ALUSrc, MemtoReg) and outputs the selected data to the PC or the Data memory.
- SignExtend:** A unit that extends the sign of a 16-bit immediate value to 32 bits, outputting the result to the ALU.
- Data memory:** Stores data and receives data from the ALU and the MUX. It outputs data to the MUX.

Control signals (indicated by red lines) include:

- RegWrite:** Controls the Register File's write enable (WE).
- ALUctrl:** Controls the ALU's operation.
- MemRead:** Controls the Data memory's read enable (RE).
- MemWrite:** Controls the Data memory's write enable (WE).
- MemtoReg:** Controls the MUX's selection between the ALU result and the Data memory output.

Data paths are shown in blue and green, indicating the flow of information between components.

第六讲 MIPS处理器设计

- 一 . 处理器设计概述
- 二 . MIPS模型机
- 三 . MIPS单周期处理器设计**
 - 1. 单周期数据通路设计
 - 2. **单周期控制器设计**
 - 3. 单周期性能分析
- 四 . MIPS多周期处理器设计简介
- 五 . MIPS流水线处理器设计

3.2 单周期控制器设计

❖ 单周期通路所需控制信号

- **ALU控制 (ALUCtrl)** : 4位
- 其他控制信号: 7个

ALU 控制

| 输入 | | ALUCtrl | ALU运算 |
|----|---|---------|-------|
| A | B | 0000 | A & B |
| A | B | 0001 | A B |
| A | B | 0010 | A + B |
| A | B | 0110 | A - B |

7个控制信号

| 控制信号 | 失效时作用 | 有效时作用 |
|----------|----------------------|--------------------|
| RegDst | 寄存器堆写入端地址来选择Rt字段 | 寄存器堆写入端地址选择 Rd字段 |
| RegWrite | 无 | 把数据写入寄存器堆中对应寄存器 |
| ALUSrc | ALU输入端B选择寄存器堆输出R[rt] | ALU输入端B选择Signext输出 |
| PCSrc | PC输入源选择 PC+4 | PC输入选择beq指令的目的地址 |
| MemRead | 无 | 数据存储器DM读数据（输出） |
| MemWrite | 无 | 数据存储器DM写数据（输入） |
| MemtoReg | 寄存器堆写入端数据来自ALU输出 | 寄存器堆写入端数据来自DM输出 |

3.2 单周期控制器设计

❖ 控制器分成两部分：主控单元和ALU控制单元

➤ 主控单元

- 输入：指令操作码字段 **Op**（指令**31:26**位）
- 输出：
 - 7个控制信号
 - ALU控制单元所需的2位输入**ALUOp**

➤ ALU控制单元

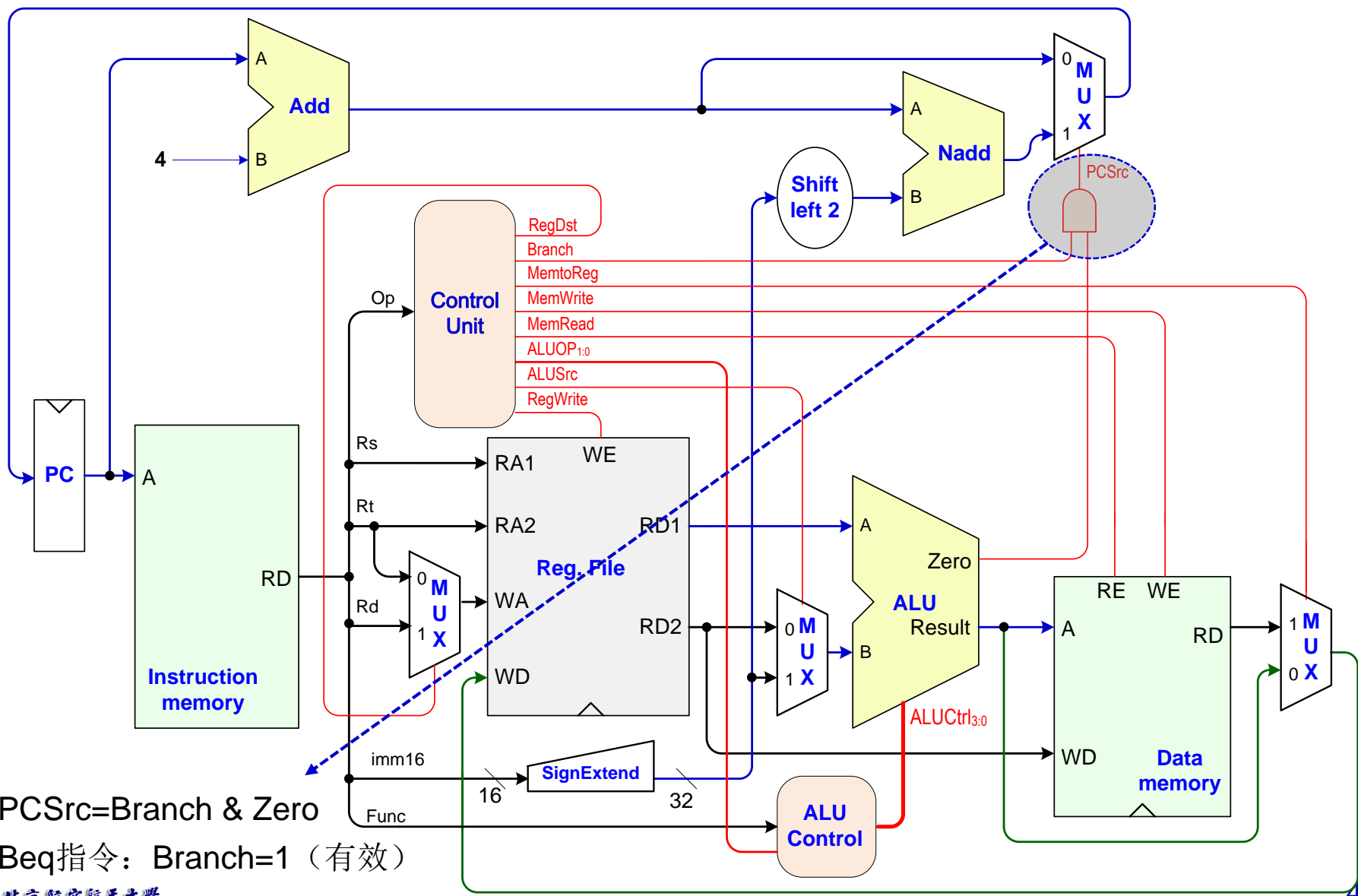
- 输入：
 - 主控单元生成的**ALUOp**
 - 功能码字段**Func**（指令**5:0**位）
- 输出：**ALU**运算控制信号 **ALUCtrl**（**4**位）

ALUOp指明ALU的运算类型

- 00: 访存指令所需加法
- 01: beq指令所需减法
- 10: R型指令功能码决定

| | | | | | |
|---------------|---------------|---------------|---------------|-----------------|---------------|
| Op (31-26) | Rs (25-21) | Rt (20-16) | Rd (15-11) | Shamt (10-6) | Func (5-0) |
|---------------|---------------|---------------|---------------|-----------------|---------------|

3.2 单周期控制器设计——模型机数据通路（带控制单元）



3.2 单周期控制器设计

❖ 主控单元控制信号分析

➤ RegDst

- R型指令: **RegDst=1**, 选择Rd
- Lw指令: **RegDst=0**, 选择Rt
- 其他指令: 不关心

➤ ALUSrc

- R型指令: **ALUSrc=0**, 选择寄存器堆的 Read data2 输出
- Lw指令: **ALUSrc=1**, 选择Signext的输出
- Sw指令: **ALUSrc=1**, 选择Signext的输出
- Beq指令 (减法运算): **ALUSrc=0**, 选择 Read data2 输出

➤ MemtoReg

- R型指令: **MemtoReg=0**, 选择 ALU 输出
- Lw指令: **MemtoReg=1**, 选择数据存储器DM输出
- 其他指令: 不关心

➤ Branch

- Beq指令: **Branch=1**, 此时若Zero=1, PC输入选择加法器Nadd输出 (分支指令目的地址), 否则选择加法器Add输出 ($PC+4$)
- 其他指令: **Branch=0**, PC输入选择加法器Add输出 ($PC+4$)

3.2 单周期控制器设计

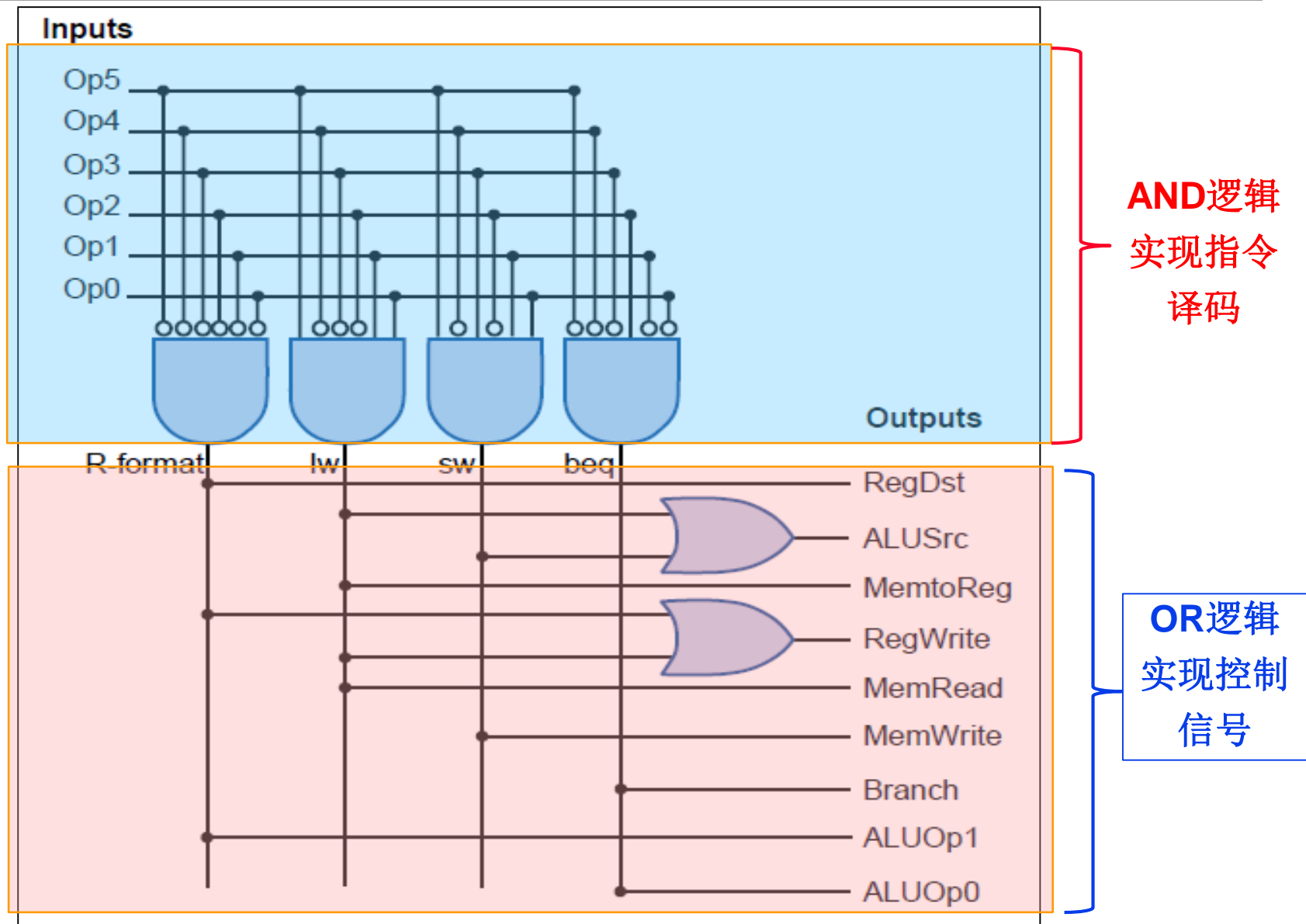
主控单元真值表

| Input or output | Signal name | R-format | lw | sw | beq |
|-----------------|-------------|----------|----|----|-----|
| Inputs | Op5 | 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 | 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 |



提供给ALU控制单元

3.2 单周期控制器设计



3.2 单周期控制器设计

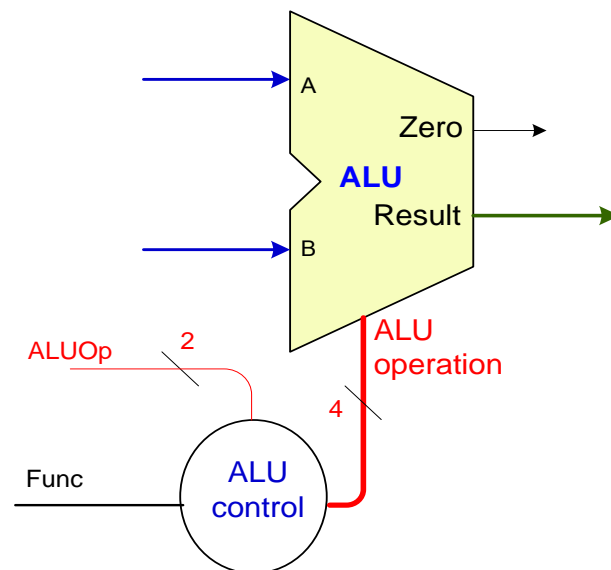
❖ ALU控制单元

➤ 输入:

- 指令的**Func**字段（指令**5:0**位）
- 由主控单元生成的 **ALUOp**

➤ **ALUOp**指明**ALU**的运算类型

- **00**: 访存指令所需的加法
- **01**: **beq**指令所需的减法
- **10**: **R**型指令功能码字段决定

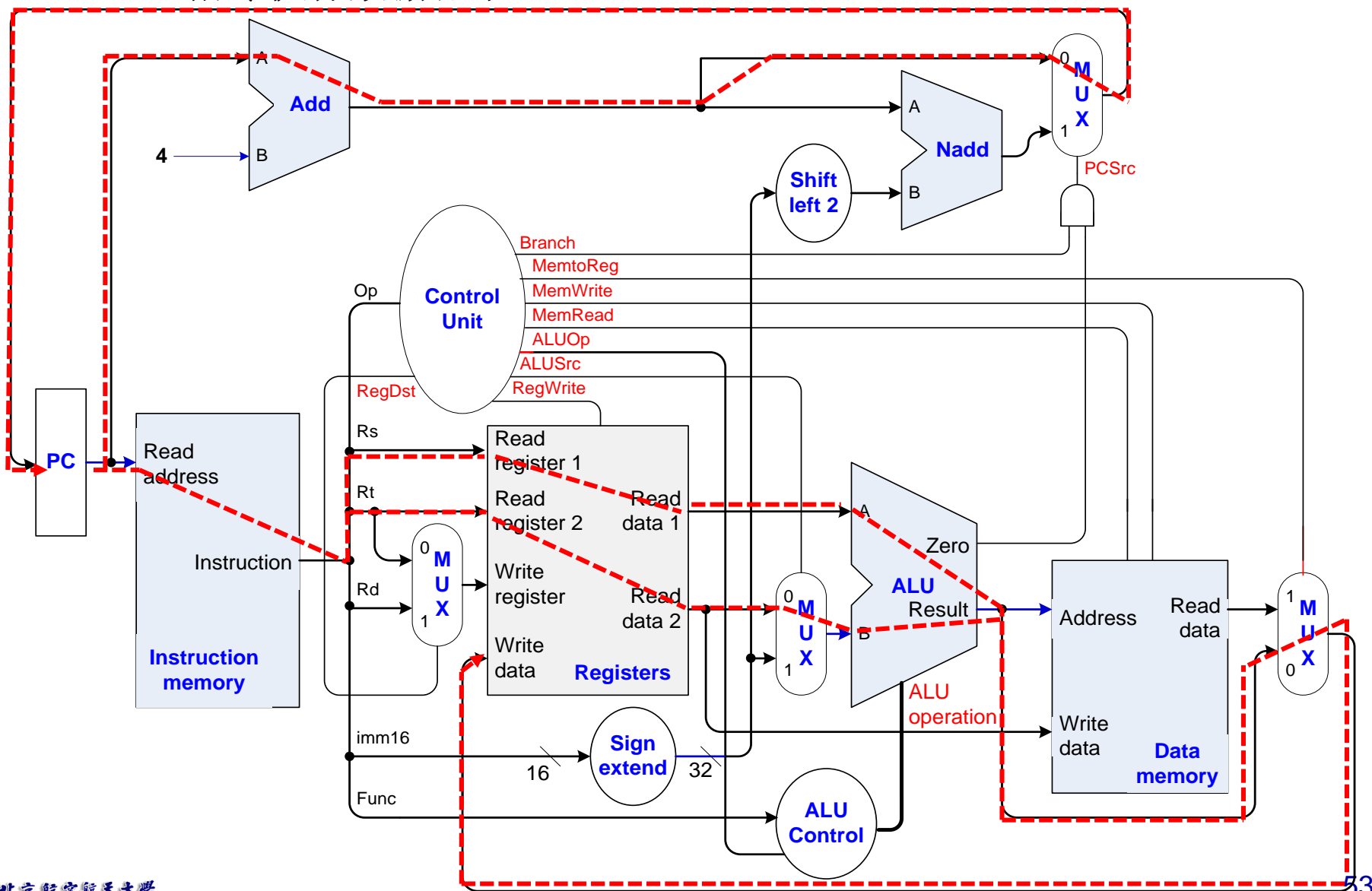


| 指令 | Func字段 | ALUOp | ALU运算类型 | ALU operation |
|-----|---------|-------|---------|---------------|
| Lw | XXXXXX | 00 | 加 | 0010 |
| Sw | XXXXXX | 00 | 加 | 0010 |
| Beq | XXXXXX | 01 | 减 | 0110 |
| Add | 100 000 | 10 | 加 | 0010 |
| Sub | 100 010 | 10 | 减 | 0110 |
| And | 100 100 | 10 | 与 | 0000 |
| Or | 100 101 | 10 | 或 | 0001 |

ALU控制单元真值表

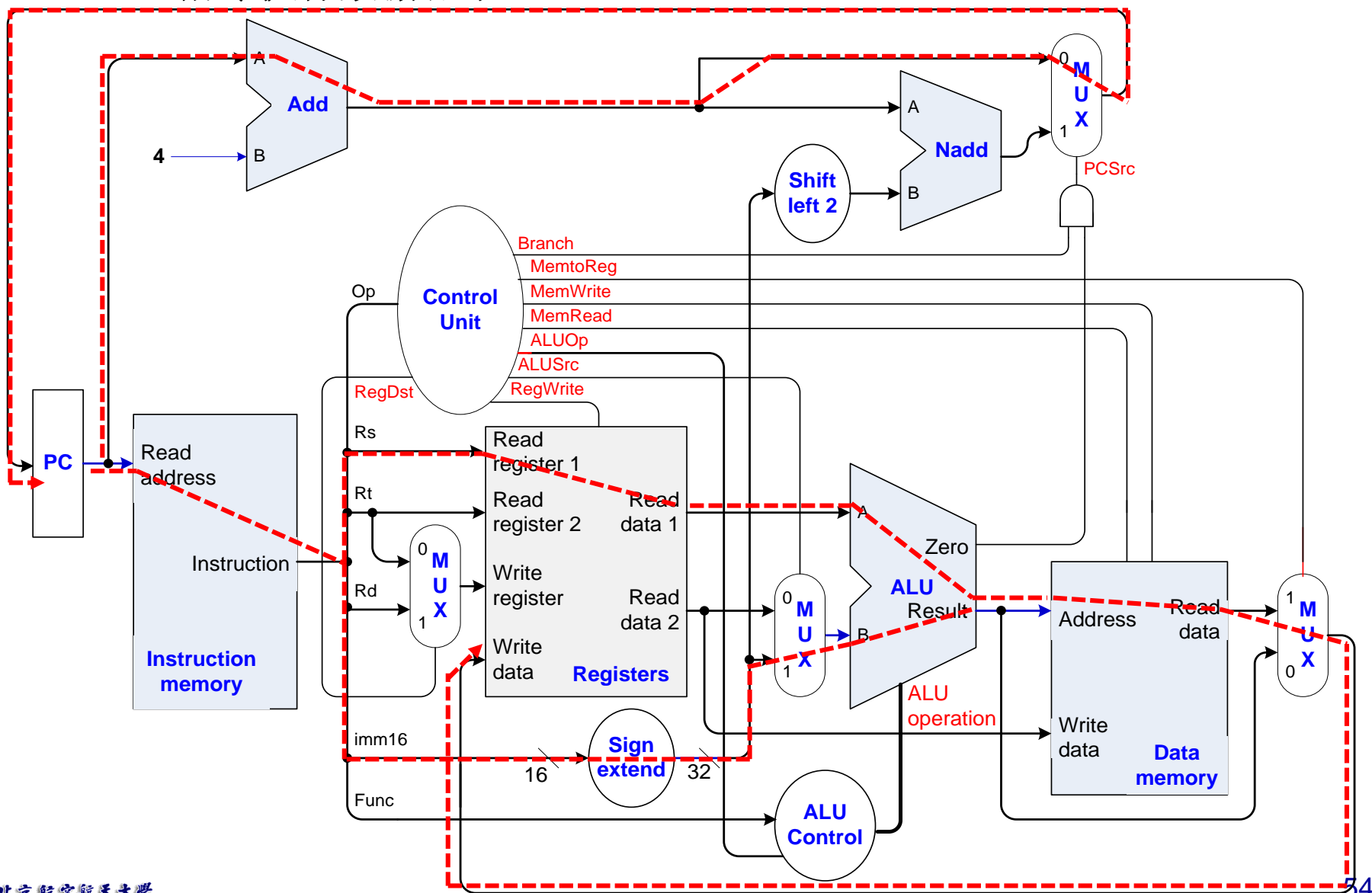
3.2 单周期控制器设计

❖ R型指令执行数据流示意



3.2 单周期控制器设计

❖ LW指令执行数据流示意



3.2 单周期控制器设计

❖ MIPS数据通路（扩展实现跳转指令 j）

➤ j add26

➤ 功能描述

■ $PC \leftarrow PC+4 [31:28] \parallel \text{add26} \ll 2$

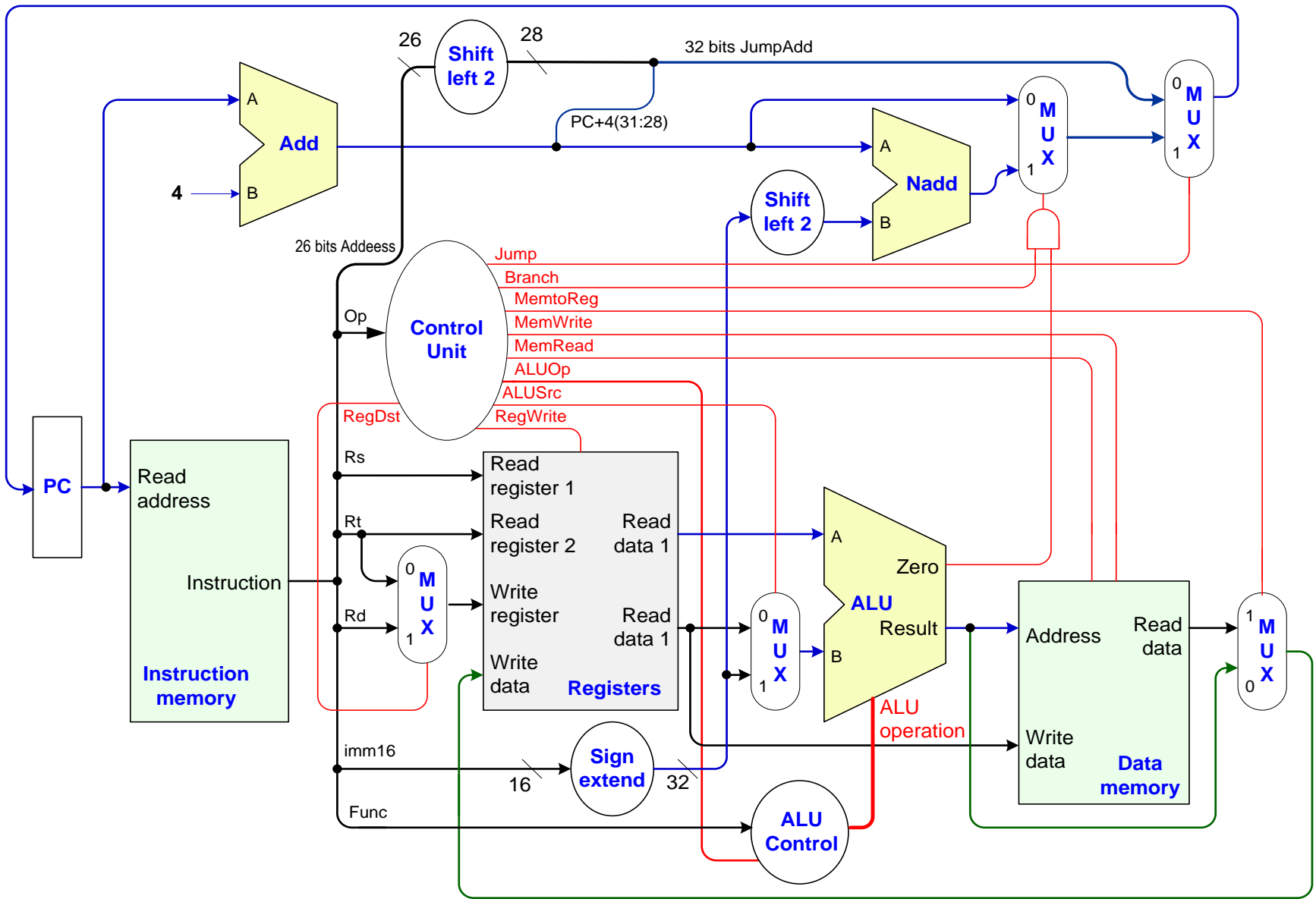
➤ 功能部件：Adder，移位器

| | |
|---------------|--|
| Op (31-26) | 26 bit Address (for Jump Instruction) (25-0) |
|---------------|--|

| 指令 | Adder | | PC | IM Ad d. | Registers | | | | ALU | | DM | | Sign- ext | Nadd | |
|-----|-------|---|-----------------|----------------|-----------|------|---------|-------------|--------|--------------------------|------|--------|--------------|-------|-------|
| | A | B | | | Reg1 | Reg2 | Wreg | Wdata | A | B | Add. | Wdata | | | |
| 合并 | PC | 4 | Adder Nadd | PC | Rs | Rt | Rd Rt | ALU DM | Rdata1 | Rdata2 Sign- ext | ALU | Rdata2 | imm16 | Adder | Shift |
| J指令 | PC | 4 | jumpadd | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |

PC输入数据源又多了一个选择，
增加一个二选一MUX

3.2 单周期控制器设计（包含跳转指令的数据通路）



小结：数据通路设计的一般性方法

单指令 数据通路 构造

- for each 指令
 - for each 新增需求
 - case 可以合并至已有部件：
修改部件设计描述、HDL建模：{F', I', O'}
 - case 需要新增部件：
建立新部件设计描述、HDL建模：{F, I, O}
增加新部件
 - for each 部件
 - 设置输入来源

多数据通 路综合

- 按垂直方向合并数据通路，并去除相同项
- for each 输入来源多余1个的输入端
 - 部署1个MUX (MUX的输入规模为输入来源数)
 - MUX设计定义、HDL建模

系统实现

- HDL建模：连接所有的部件及所有的MUX

部件描述示例——PC

4.1.1. 基本描述

PC 模块的主要功能是将 NPC[31:0]的值保存并输出。PC 的各种取值将根据所执行的指令、外部状态(中断)及处理器控制器的当前状态的不同, 由数据通路其他部件生成。

4.1.2. 模块接口

表 4-1 PC 接口信号定义

| 信号名 | 方向 | 描述 |
|----------------------|----|--------------|
| <u>Clk</u> | I | MIPS-C 处理器时钟 |
| <u>Reset</u> | I | 复位信号 |
| <u>NextPC</u> [31:0] | I | 下一个 PC 值 |
| <u>PCWr</u> | I | PC 写使能 |
| <u>PC</u> [31:0] | O | PC 输出 |

4.1.3. 功能定义

PC 模块的核心是一个寄存器。该寄存器在 PCWr 有效时将 NextPC[31:0]锁存并输出。

表 4-2 PC 功能需求定义

| 编号 | 功能名称 | 功能描述 |
|----|-------|--|
| 1 | 初始化 | 当 Reset 信号有效后, PC 输出 0xBFC00000。 |
| 2 | PC 更新 | 当时钟上升沿到来时, <u>PCWr</u> 有效则将 NPC 写入 PC 内部, 并且从 PC 端口输出。 |

部件HDL建模示例——PC

```
16 `timescale 1ns/1ns
17
18 module PC( CLK_I, Reset_I, Addr_I, PCWrite_I, PC_O ) ;
19     input          CLK_I;          // system clock
20     input          Reset_I;        // reset signal
21     input [31:0]   Addr_I;         // next PC
22     input          PCWrite_I;      // write enable
23     output [31:0]  PC_O;           // PC output
24
25
26     /* internal reg and wire */
27     reg [31:0]  addr ;              // latch the address
28
29     /* read register */
30     assign PC_O = addr;
31
32     always@ ( posedge CLK_I or posedge Reset_I )
33     begin
34         if (Reset_I)
35             addr <= 'hBFC00000;
36         else if ( PCWrite_I )
37             addr <= Addr_I;
38     end
39
40 endmodule
```

第六讲 MIPS处理器设计

一 . 处理器设计概述

二 . MIPS模型机

三 . MIPS单周期处理器设计

1. 单周期数据通路设计

2. 单周期控制器设计

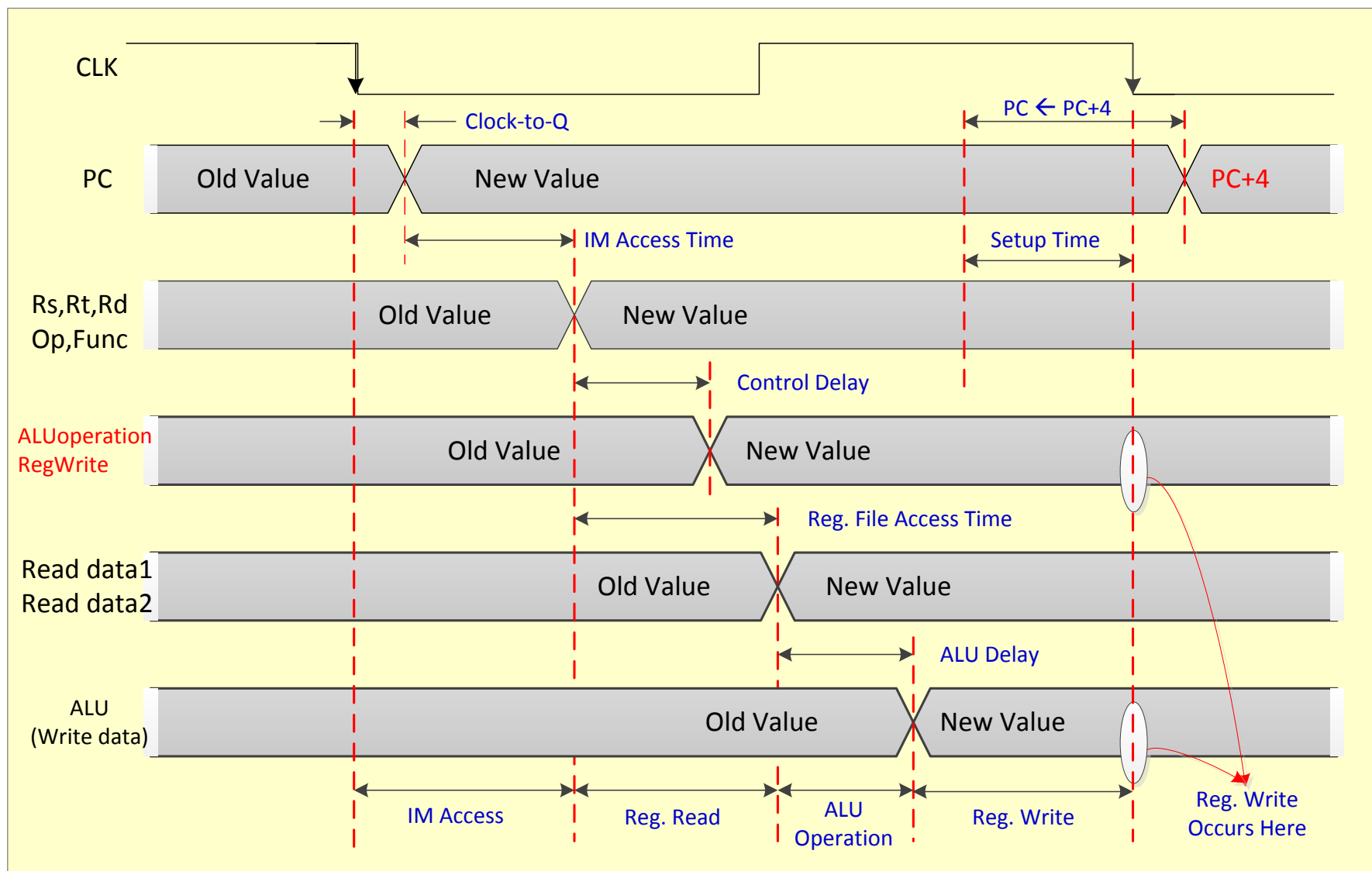
3. 单周期性能分析

四 . MIPS多周期处理器设计简介

五 . MIPS流水线处理器设计

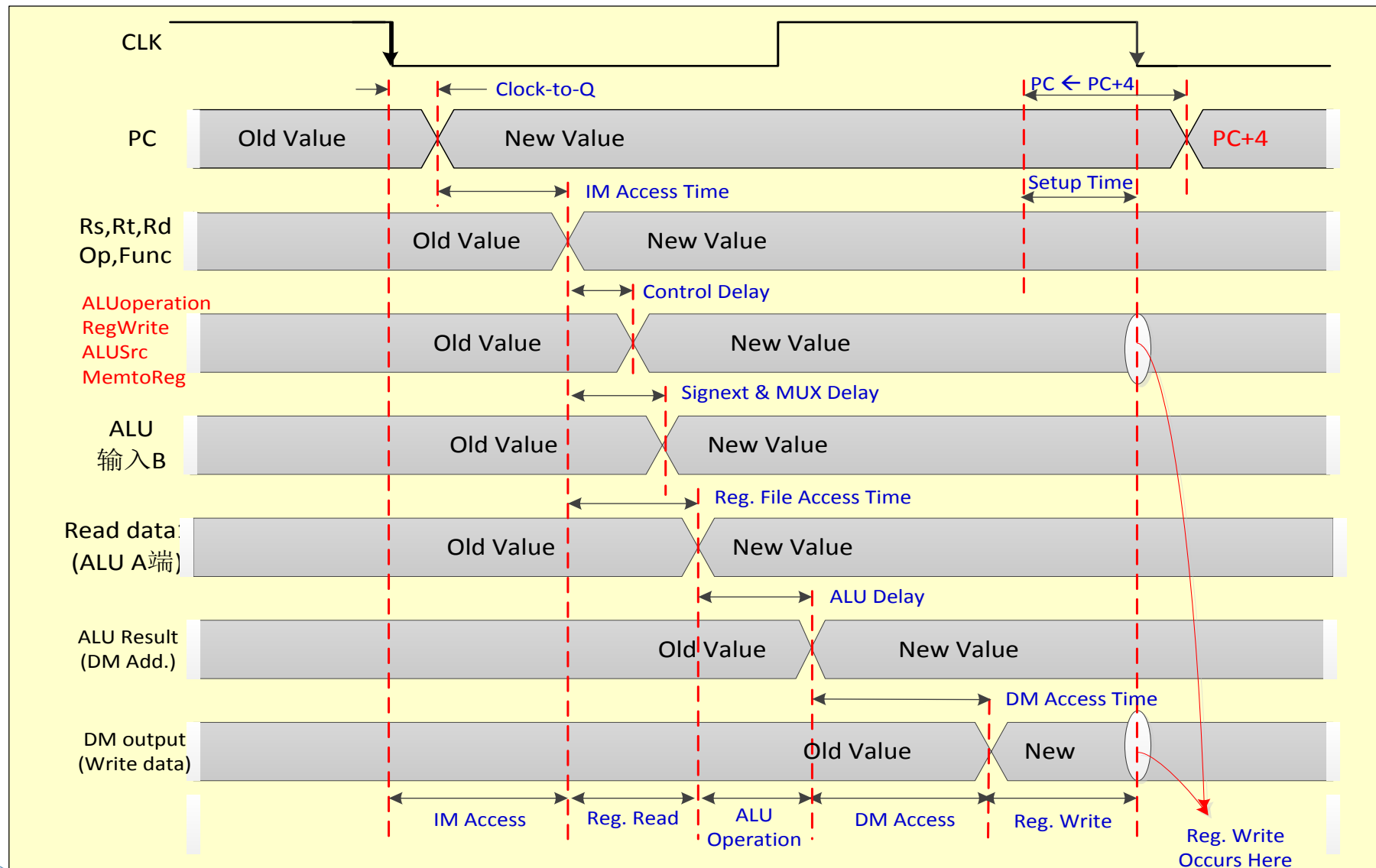
3.3 单周期数据通路性能分析

❖ R型指令的指令周期



3.3 单周期数据通路性能分析

❖ LW指令的指令周期



3.3 单周期数据通路性能分析

❖ 指令周期（指令执行时间）

➤ R指令周期

- 取指令（IM Access Time）
- 读寄存器（Register File Access Time）
- ALU运算（ALU Operation）
- 写寄存器（Register File Access Time）

➤ LW指令周期

- 取指令（IM Access Time）
- 读寄存器（Register Access Time）
- ALU运算（ALU Operation）
- 读数据（DM Access Time）
- 写寄存器（Register File Access Time）

实际上，不同类型的指令可能具有不同的指令周期

3.3 单周期数据通路性能分析

MIPS不同类型指令的指令周期

| Instruction class | Functional units used by the instruction class | | | | |
|-------------------|--|-----------------|-----|-----------------|-----------------|
| R-type | Instruction fetch | Register access | ALU | Register access | |
| Load word | Instruction fetch | Register access | ALU | Memory access | Register access |
| Store word | Instruction fetch | Register access | ALU | Memory access | |
| Branch | Instruction fetch | Register access | ALU | | |
| Jump | Instruction fetch | | | | |

数据通路各部分以及各类指令的执行时间

| Instruction class | Instruction memory | Register read | ALU operation | Data memory | Register write | Total |
|-------------------|--------------------|---------------|---------------|-------------|----------------|--------|
| R-type | 200 | 50 | 100 | 0 | 50 | 400 ps |
| Load word | 200 | 50 | 100 | 200 | 50 | 600 ps |
| Store word | 200 | 50 | 100 | 200 | | 550 ps |
| Branch | 200 | 50 | 100 | 0 | | 350 ps |
| Jump | 200 | | | | | 200 ps |

3.3 单周期数据通路性能分析

❖ 指令执行时间计算

1. 方式一：采用单周期，即所有指令周期固定为单一时钟周期

- 时钟周期有最长的指令决定（LW指令），为 **600ps**

- 指令平均周期 = **600ps**

2. 方式二：不同类型指令采用不同指令周期（可变时钟周期）

- 假设指令在程序中出现的频率

- lw指令 : 25%

- sw指令 : 10%

- R类型指令 : 45%

- beq指令 : 15%

- j指令 : 5%

- 平均指令执行时间

$$600 \times 25\% + 550 \times 10\% + 400 \times 45\% + 350 \times 15\% + 200 \times 5\% = \mathbf{447.5ps}$$

— 若采用可变时钟周期，时间性能比单周期更高；

— 但控制比单周期要复杂、困难，得不偿失。

— 改进方法：改变每种指令类型所用的时钟数，即采用多周期实现

第六讲 MIPS处理器设计

- 一 . 处理器设计概述
- 二 . MIPS模型机
- 三 . MIPS单周期处理器设计
- 四 . MIPS多周期处理器设计简介**
 - 1. 多周期数据通路设计（简介）**
 - 2. 多周期控制器设计（略）**
- 五 . MIPS流水线处理器设计

4.1 MIPS 多周期数据通路设计

❖ 为什么不使用单周期实现方式

- 单周期设计中，时钟周期对所有指令等长。而时钟周期由计算机中可能的最长路径决定，一般为取数指令。但某些指令类型本来可以在更短时间内完成。

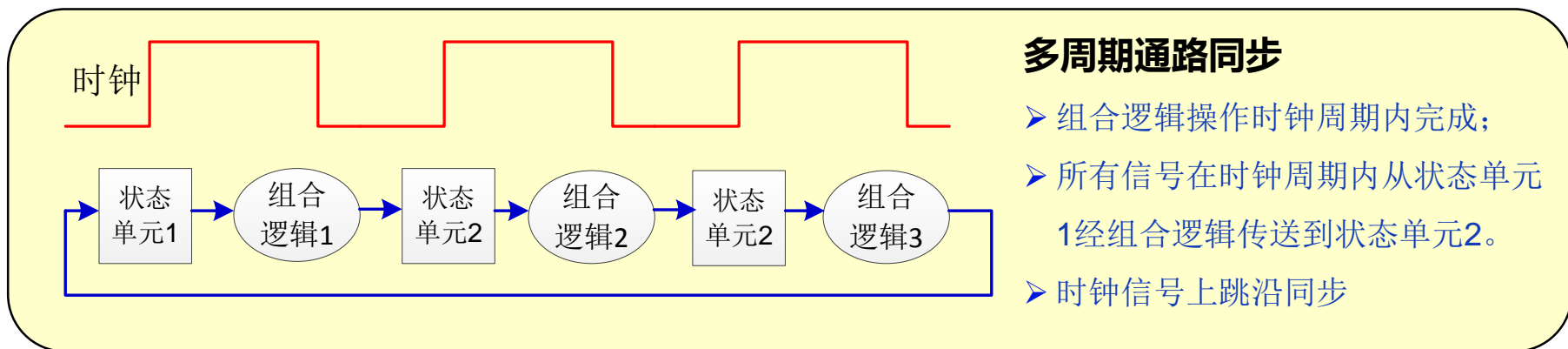
❖ 多周期方案

- 将指令执行分解为多个步骤，每一步骤一个时钟周期，则指令执行周期为多个时钟周期，不同指令的指令周期包含时钟周期数不一样。
- 优点：
 - **提高性能**：不同指令的执行占用不同的时钟周期数；
 - **降低成本**：一个功能单元可以在一条指令执行过程中使用多次，只要是在不同周期中（这种共享可减少所需的硬件数量）。

4.1 MIPS多周期数据通路设计

❖ 多周期数据通路设计总体考虑

- **普林斯顿结构**：指令和数据使用同一个存储器器；
- 一个**ALU**： R型指令算术逻辑运算、取指令后形成**PC+4**新值、及**Beq**指令转向地址计算（**PC+Signext(imm16)<<2**），都在**ALU**中完成；
- 时钟同步方法：一个时钟周期内信号总是从一个状态单元经过组合逻辑处理后传送到另一个状态单元。所以**指令每一步的执行总是从前一个状态单元接收输入，经过功能单元处理，在下一个时钟周期触发沿将结果写入下一个状态单元**，因此数据通路中需要增加了一个或多个寄存器以保存指令各执行步骤形成的结果（输出值），以便在指令的后续时钟周期内继续使用。



时钟同步方法

4.1 MIPS多周期数据通路设计

1. R型指令多周期分析

| | | | | | |
|---------------|---------------|---------------|---------------|-----------------|---------------|
| Op (31-26) | Rs (25-21) | Rt (20-16) | Rd (15-11) | Shamt (10-6) | Func (5-0) |
|---------------|---------------|---------------|---------------|-----------------|---------------|

◆ $R[rd] \leftarrow R[rs] \text{ op } R[rt]$

| 步骤 | RTL描述 | 执行部件 | 备注 |
|------|--|------------|---|
| 取指令 | $IR \leftarrow M[PC]$ $PC \leftarrow PC + 4$ | 存储器 ALU | 增加一个寄存器保存指令: IR |
| 读寄存器 | $A \leftarrow R[IR[25:21]]$ $B \leftarrow R[IR[20:16]]$ | 寄存器堆 | 增加两个寄存器保存读取的数据 R[rs] 和 R[rt] : A、B |
| 计算 | $ALUOut \leftarrow A \text{ op } B$ | ALU | 增加一个寄存器保存ALU结果: ALUOut |
| 写寄存器 | $R[IR[15:11]] \leftarrow ALUOut$ | 寄存器堆 | 完成寄存器数据写入 |

4.1 MIPS多周期数据通路设计

2. Lw指令多周期分析

| | | | |
|---------------|---------------|---------------|---------------------------------------|
| Op (31-26) | Rs (25-21) | Rt (20-16) | 16 bit Address or Immediate (15-0) |
|---------------|---------------|---------------|---------------------------------------|

◆ $R[rt] \leftarrow M[R[rs] + \text{signext}(\text{imm}16)]$

| 步骤 | RTL描述 | 执行部件 | 备注 |
|-------|---|------------|-------------------|
| 取指令 | $IR \leftarrow M[PC]$ $PC \leftarrow PC + 4$ | 存储器 ALU | 指令寄存器IR用于保存读取的指令 |
| 读寄存器 | $A \leftarrow R[IR[25:21]]$ | 寄存器堆 | 寄存器A保存从读取的数据R[rs] |
| 计算地址 | $ALUOut \leftarrow A +$ $\text{signext}(IR[15:0])$ | ALU | ALUOut保存计算得到的内存地址 |
| 访问存储器 | $DR \leftarrow M[ALUOut]$ | 存储器 | 增加一个寄存器保存数据: DR |
| 写寄存器 | $R[IR[20:16]] \leftarrow DR$ | 寄存器堆 | 完成寄存器数据写入 |

4.1 MIPS多周期数据通路设计

3. Sw指令多周期分析

| | | | |
|---------------|---------------|---------------|---------------------------------------|
| Op (31-26) | Rs (25-21) | Rt (20-16) | 16 bit Address or Immediate (15-0) |
|---------------|---------------|---------------|---------------------------------------|

◆ $M[R[rs] + \text{signext}(\text{imm16})] \leftarrow R[rt]$

| 步骤 | RTL描述 | 执行部件 | 备注 |
|-------|---|------------|-------------------|
| 取指令 | $IR \leftarrow M[PC]$ $PC \leftarrow PC + 4$ | 存储器 ALU | 指令寄存器IR用于保存读取的指令 |
| 读寄存器 | $A \leftarrow R[IR[25:21]]$ | 寄存器堆 | 寄存器A保存从读取的数据R[rs] |
| 计算地址 | $ALUOut \leftarrow A +$ $\text{signext}(IR[15:0])$ | ALU | ALUOut保存计算得到的内存地址 |
| 访问存储器 | $M[ALUOut] \leftarrow R[IR[20:16]]$ | 存储器 | 把数据R[rt]写入存储器 |

4.1 MIPS多周期数据通路设计

4. Beq指令多周期分析

| | | | |
|---------------|---------------|---------------|---------------------------------------|
| Op (31-26) | Rs (25-21) | Rt (20-16) | 16 bit Address or Immediate (15-0) |
|---------------|---------------|---------------|---------------------------------------|

◆ If ($(R[rs] - R[rt]) == 0$) then $PC \leftarrow PC + 4 + \text{signext}(\text{imm16}) \ll 2$
 else $PC \leftarrow PC + 4$

◆ 问题：ALU 要完成 $PC + 4$ ， $PC + \text{signext}(\text{imm16}) \ll 2$ 和 $R[rs] - R[rt]$ 三次运算操作，需要在三个不同时钟周期内实现，**如何合理安排？**

| 步骤 | RTL描述 | 执行部件 | 备注 |
|---------------|---|-------------|--------------------------------------|
| 取指令 | $IR \leftarrow M[PC]$ $PC \leftarrow PC + 4$ | 存储器 ALU | 第一周期后 $PC+4$ 完成， PC 为新值 |
| 读寄存器 / 计算转向地址 | $A \leftarrow R[IR[25:21]]$ $B \leftarrow R[IR[20:16]]$ $ALUOut \leftarrow PC + \text{Signext}[IR[15:0]] \ll 2$ | 寄存器堆 ALU | ALU 计算转向目标地址保存于 $ALUOut$ |
| 完成转移 | if ($A - B == 0$) then $PC \leftarrow ALUOut$ | ALU | ALU 减法结果不写入 ALU ， PC 修改是一条条件操作 |

4.1 MIPS多周期数据通路设计

❖ 指令数据通路综合（R型指令、Lw、Sw、Beq）

| 步骤 | R型指令 | Lw指令 | Sw指令 | Beq指令 |
|---------------|--|---|--------------------------|--|
| 取指令 | $IR \leftarrow M[PC]$ $PC \leftarrow PC + 4$ <div>ALU完成加法 结果送PC</div> | | | |
| 读寄存器 /译码 | $A \leftarrow R[IR[25:21]]$ $B \leftarrow R[IR[20:16]]$ $ALUOut \leftarrow PC + Signext[IR[15:0]] < < 2$ | | | |
| 计算 | $ALUOut \leftarrow A \text{ op } B$ | $ALUOut \leftarrow A + Signext(IR[15:0])$ | | If (A-B==0) then $PC \leftarrow ALUOut$ |
| R型完成/ 访问内存 | $R[IR[15:11]] \leftarrow ALUOut$ | $DR \leftarrow M[ALUOut]$ | $M[ALUOut] \leftarrow B$ | |
| 写寄 存器 | | $R[IR[20:16]] \leftarrow DR$ | | |

4.1 MIPS多周期数据通路设计

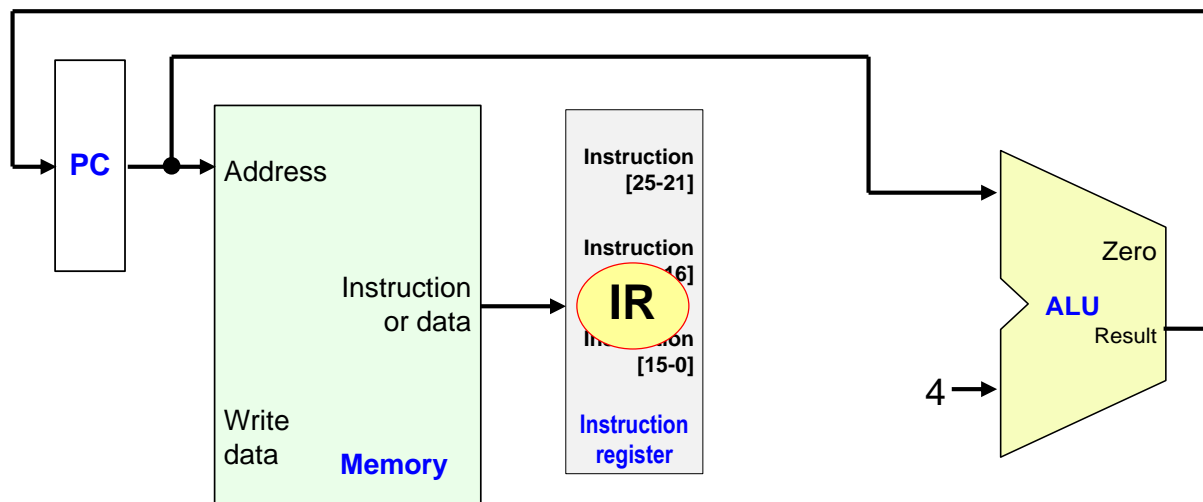
❖ 指令数据通路综合（R型指令、Lw、Sw、Beq）

| 步骤 | R型指令 | Lw指令 | Sw指令 | Beq指令 |
|---------------|---|--|--------------------------|--|
| 取指令 | $IR \leftarrow M[PC]$ $PC \leftarrow PC + 4$ | | | P0 |
| 读寄存器 /译码 | $A \leftarrow R[IR[25:21]]$ $B \leftarrow R[IR[20:16]]$ $ALUOut \leftarrow PC + \text{Signext}[IR[15:0]] \ll 2$ | | | P1 |
| 计算 | $ALUOut \leftarrow A \text{ op } B$ | $ALUOut \leftarrow A + \text{Signext}(IR[15:0])$ | | If $(A-B==0)$ then $PC \leftarrow ALUOut$ |
| R型完成/ 访问内存 | $R[IR[15:11]] \leftarrow ALUOut$ | $DR \leftarrow M[ALUOut]$ | $M[ALUOut] \leftarrow B$ | |
| 写寄存器 | | $R[IR[20:16]] \leftarrow DR$ | | |

4.1 MIPS多周期数据通路设计

1. 多周期数据通路构建示例（P0）

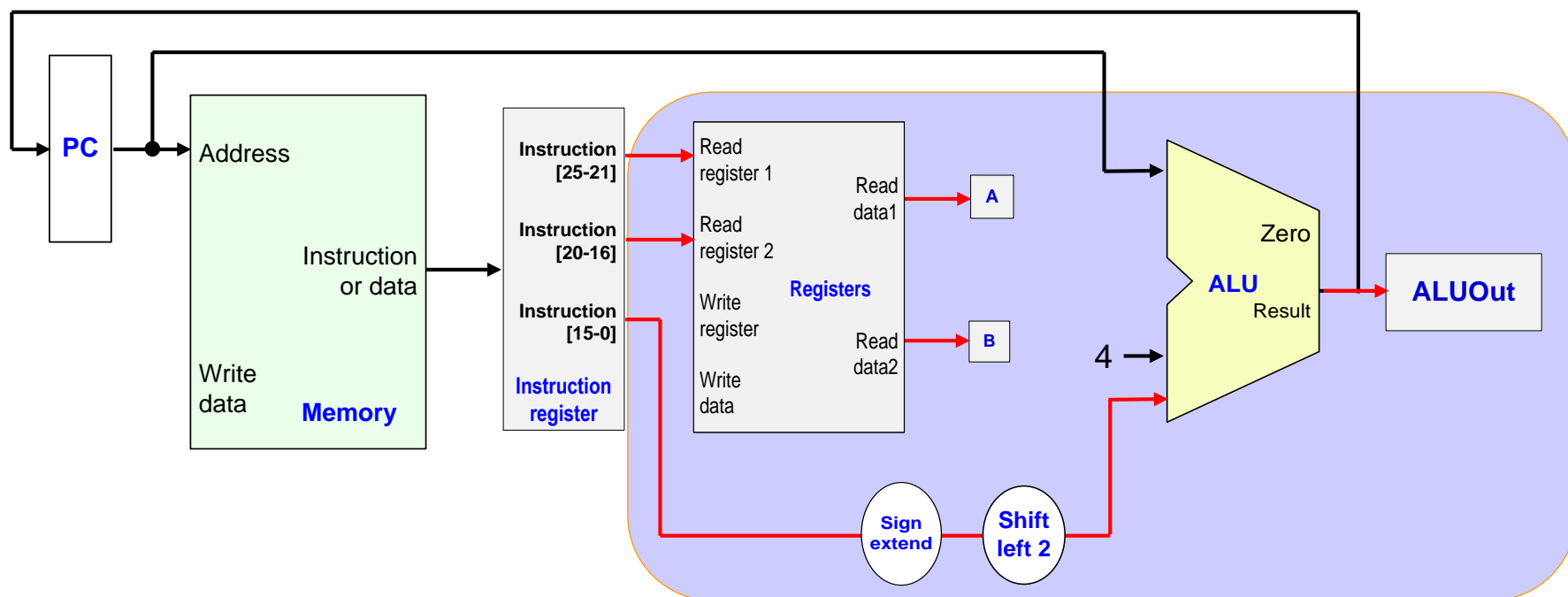
- ◆ $IR \leftarrow M[PC]$
- ◆ $PC \leftarrow PC + 4$



4.1 MIPS多周期数据通路设计

2. 多周期数据通路构建示例（P1）

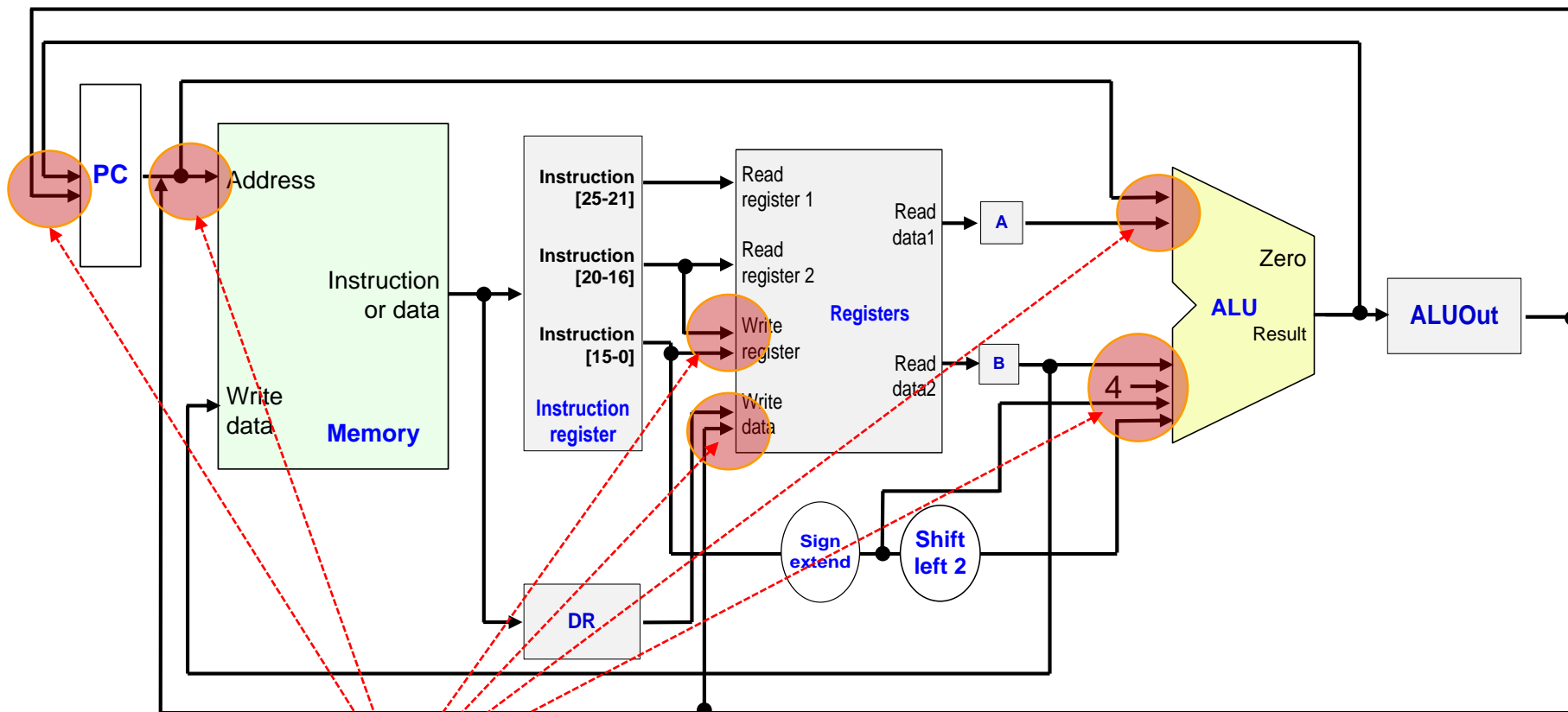
- ◆ $A \leftarrow R[IR[25:21]], B \leftarrow R[IR[20:16]]$
- ◆ $ALUOut \leftarrow PC + Signext[IR[15:0]] \ll 2$



4.1 MIPS多周期数据通路设计

3. 多周期数据通路构建

- ◆ 存在多个输入源的，需要增加多路选择器MUX

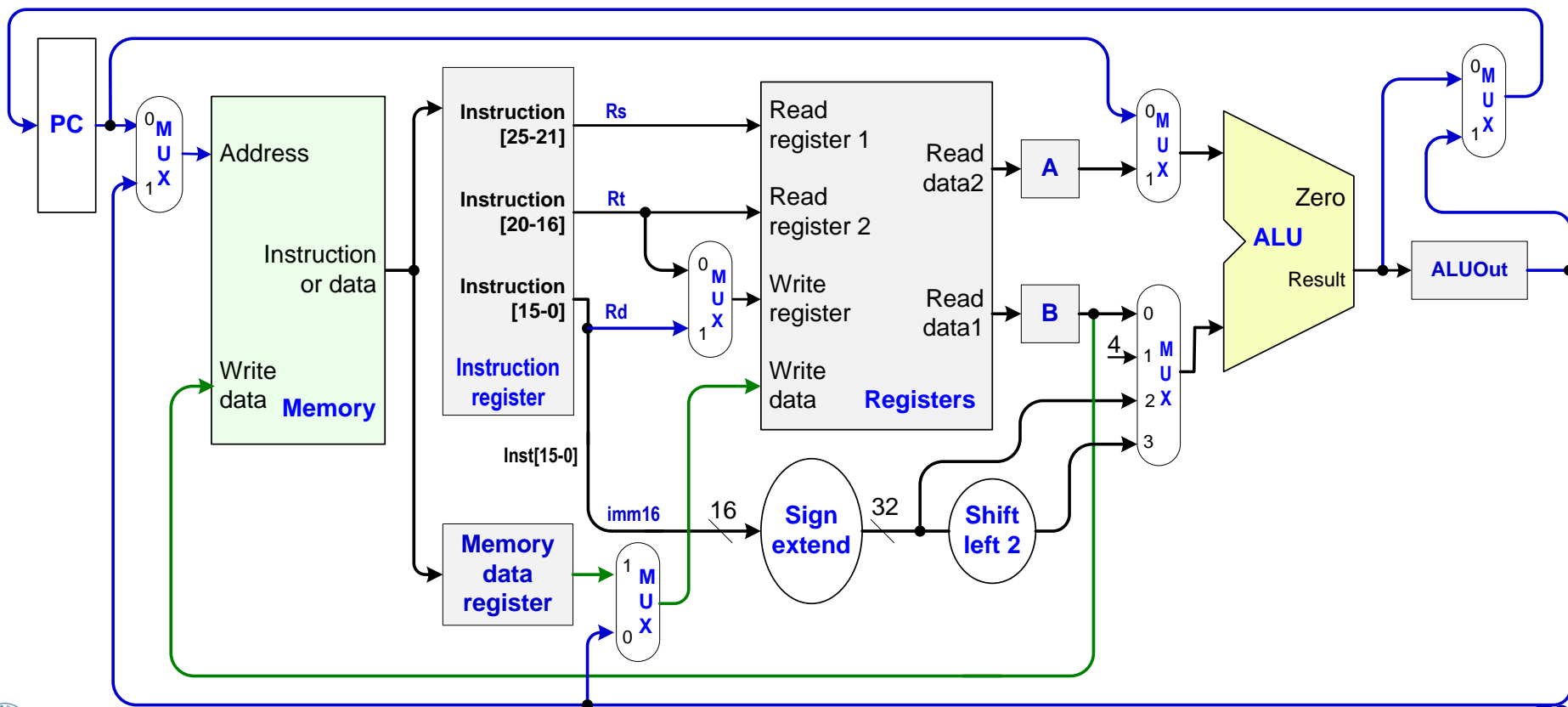


加入多路选择MUX

4.1 MIPS多周期数据通路实现

❖ MIPS的多周期数据通路

- **PC**: 输入端多路选择 (2选1)
- **存储器**: 地址输入端多路选择 (2选1)
- **ALU**: 输入A端多路选择 (2选1), 输入B端多路选择 (4选1)
- **寄存器堆**: 写地址输入端 (2选1), 写数据输入端 (2选1)



4.1 MIPS多周期数据通路设计

■ 增加J指令（转移）通路

Op
(31-26)

26 bit Address (for Jump Instruction)
(25-0)

➤ $PC \leftarrow PC + 4 [31:28] \parallel add26 < < 2$

| 步骤 | R型指令 | Lw指令 | Sw指令 | Beq指令 | J指令 |
|-----------|--|---|--------------------------|--|--|
| 取指令 | $IR \leftarrow M[PC]$ $PC \leftarrow PC + 4$ | | | | |
| 读寄存器/译码 | $A \leftarrow R[IR[25:21]]$ $B \leftarrow R[IR[20:16]]$ $ALUOut \leftarrow PC + Signext[IR[15:0]] < < 2$ | | | | |
| 计算 | $ALUOut \leftarrow A \text{ op } B$ | $ALUOut \leftarrow A + Signext(IR[15:0])$ | | If (A-B==0) then $PC \leftarrow ALUOut$ | $PC \leftarrow PC[31:28] \parallel IR[25:0] < < 2$ |
| R型完成/访问内存 | $R[IR[15:11]] \leftarrow ALUOut$ | $DR \leftarrow M[ALUOut]$ | $M[ALUOut] \leftarrow B$ | | |
| 写寄存器 | | $R[IR[20:16]] \leftarrow DR$ | | | |

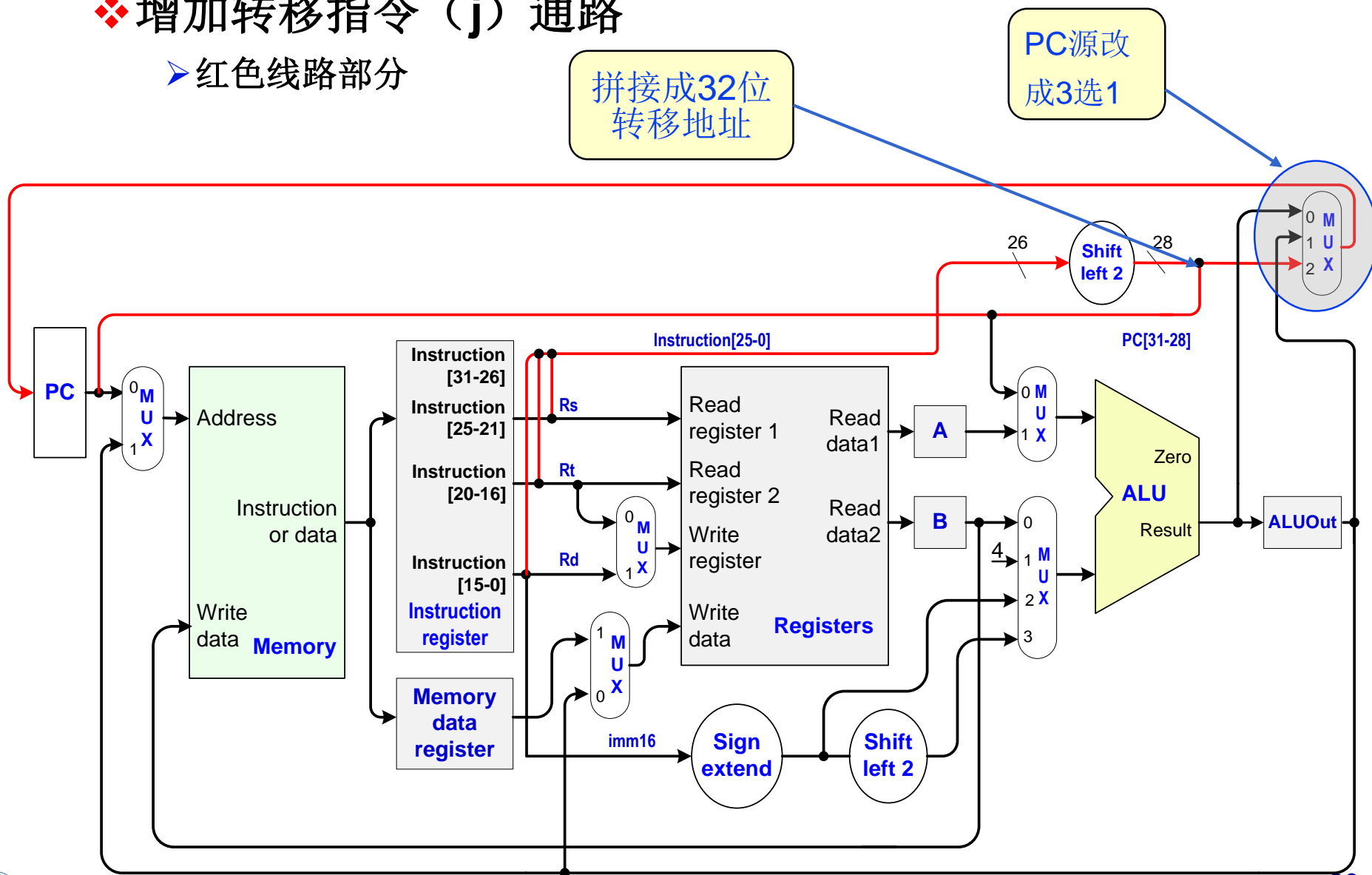
4.1 MIPS多周期数据通路实现

❖ 增加转移指令（j）通路

➤ 红色线路部分

拼接成32位
转移地址

PC源改
成3选1



多周期性能分析

❖ 假设主要功能单元的操作时间

- 存储器 : 200ps
- ALU : 100ps
- 寄存器堆: 50ps
- 多路复用器、控制单元、PC、符号扩展单元、线路没有延迟

各类指令执行时间

| 步骤 | R型指令 | Lw指令 | Sw指令 | Beq指令 | J指令 | 执行时间 |
|---------------|--|---|--------------------------|--|--|-------|
| 取指令 | $IR \leftarrow M[PC], PC \leftarrow PC + 4$ | | | | | 200ps |
| 读寄存器/ 译码 | $A \leftarrow R[IR[25:21]], B \leftarrow R[IR[20:16]]$ $ALUOut \leftarrow PC + Signext[IR[15:0]] \ll 2$ | | | | | 50ps |
| 计算 | $ALUOut \leftarrow A \text{ op } B$ | $ALUOut \leftarrow A + Signext(IR[15:0])$ | | If (A-B==0) then $PC \leftarrow ALUout$ | $PC \leftarrow PC[31:28] \parallel IR[25:0] \ll 2$ | 100ps |
| R型完成/ 访问内存 | $R[IR[15:11]] \leftarrow ALUOut$ | $DR \leftarrow M[ALUOut]$ | $M[ALUOut] \leftarrow B$ | | | 200ps |
| 写寄存器 | | $R[IR[20:16]] \leftarrow DR$ | | | | 50ps |

多周期控制器设计

❖ 时钟周期

➤ 时钟周期取各步骤中最长的时间，**200ps**

各类指令执行时间

| 时钟周期 | R型指令 | Lw指令 | Sw指令 | Beq指令 | J指令 | 周期时间 |
|------|--|---|--------------------------|--|---|-------|
| TC1 | $IR \leftarrow M[PC], PC \leftarrow PC + 4$ | | | | | 200ps |
| TC2 | $A \leftarrow R[IR[25:21]], B \leftarrow R[IR[20:16]]$ $ALUOut \leftarrow PC + Signext[IR[15:0]] \ll 2$ | | | | | 200ps |
| TC3 | $ALUOut \leftarrow A \text{ op } B$ | $ALUOut \leftarrow A + Signext(IR[15:0])$ | | If (A-B==0) then $PC \leftarrow ALUout$ | $PC \leftarrow PC[31:28]$ $\parallel IR[25:0] \ll 2$ | 200ps |
| TC4 | $R[IR[15:11]] \leftarrow ALUOut$ | $DR \leftarrow M[ALUOut]$ | $M[ALUOut] \leftarrow B$ | | | 200ps |
| TC5 | | $R[IR[20:16]] \leftarrow DR$ | | | | 200ps |

多周期性能分析

❖ 各型指令所需的时钟周期数和时间

- R型指令 : 800ps
- lw指令 : 1000ps
- sw指令 : 800ps
- beq指令 : 600ps
- j指令 : 600ps

❖ 假设指令在程序中出现的频率

- lw指令 : 25%
- sw指令 : 10%
- R型指令: 45%
- beq指令: 15%
- j指令 : 5%

❖ 则一条指令的平均CPI

- $5 \times 25\% + 4 \times 10\% + 4 \times 45\% + 3 \times 15\% + 3 \times 5\% = 4.05$

❖ 一条指令的平均执行时间:

- $1000 \times 25\% + 800 \times 10\% + 800 \times 45\% + 600 \times 15\% + 600 \times 5\% = 810\text{ps}$

第六讲 MIPS处理器设计

一 . 处理器设计概述

二 . MIPS模型机

三 . MIPS单周期处理器设计

四 . MIPS多周期处理器设计简介

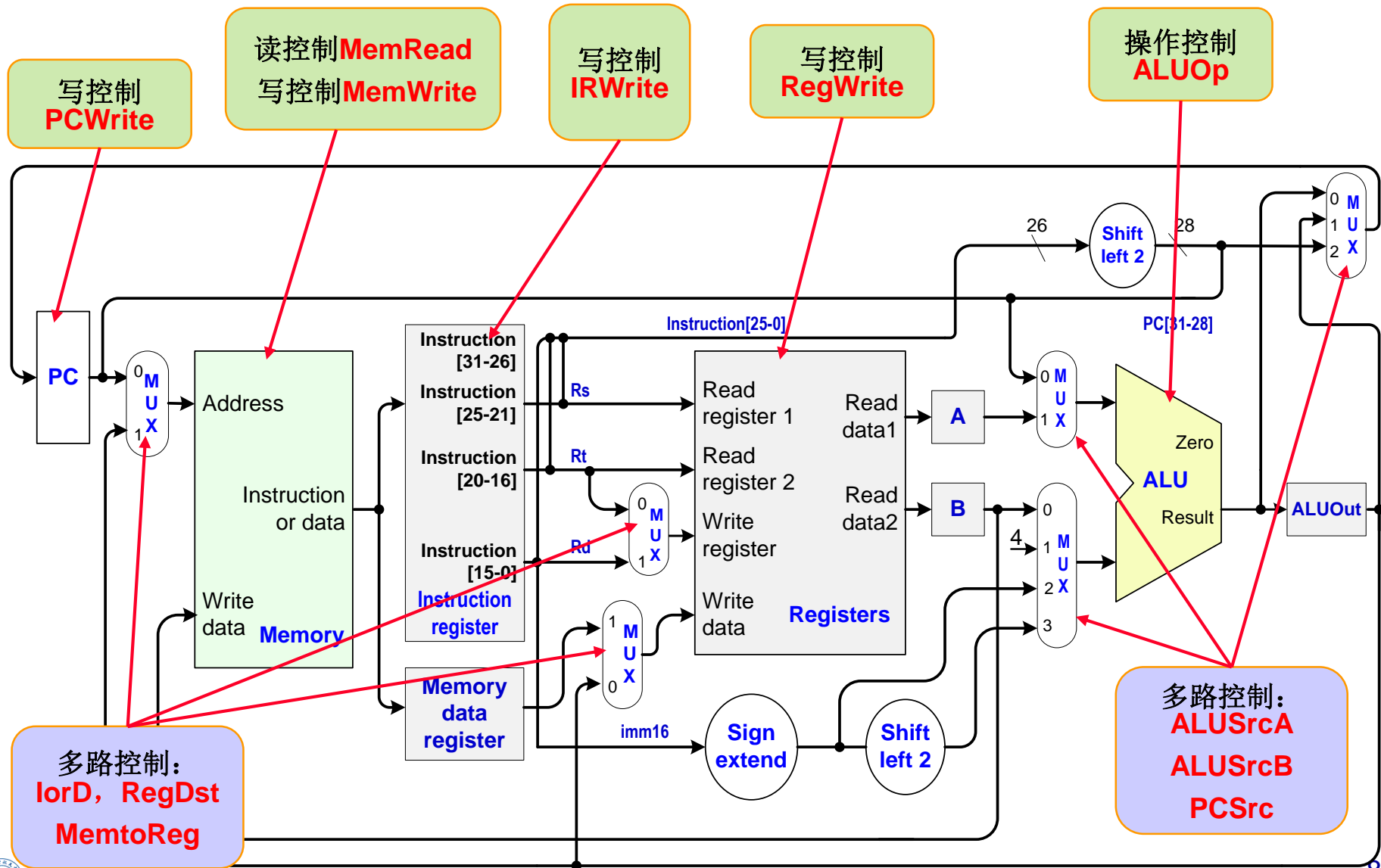
1. 多周期数据通路设计（简介）

2. 多周期控制器设计（略，可自学）

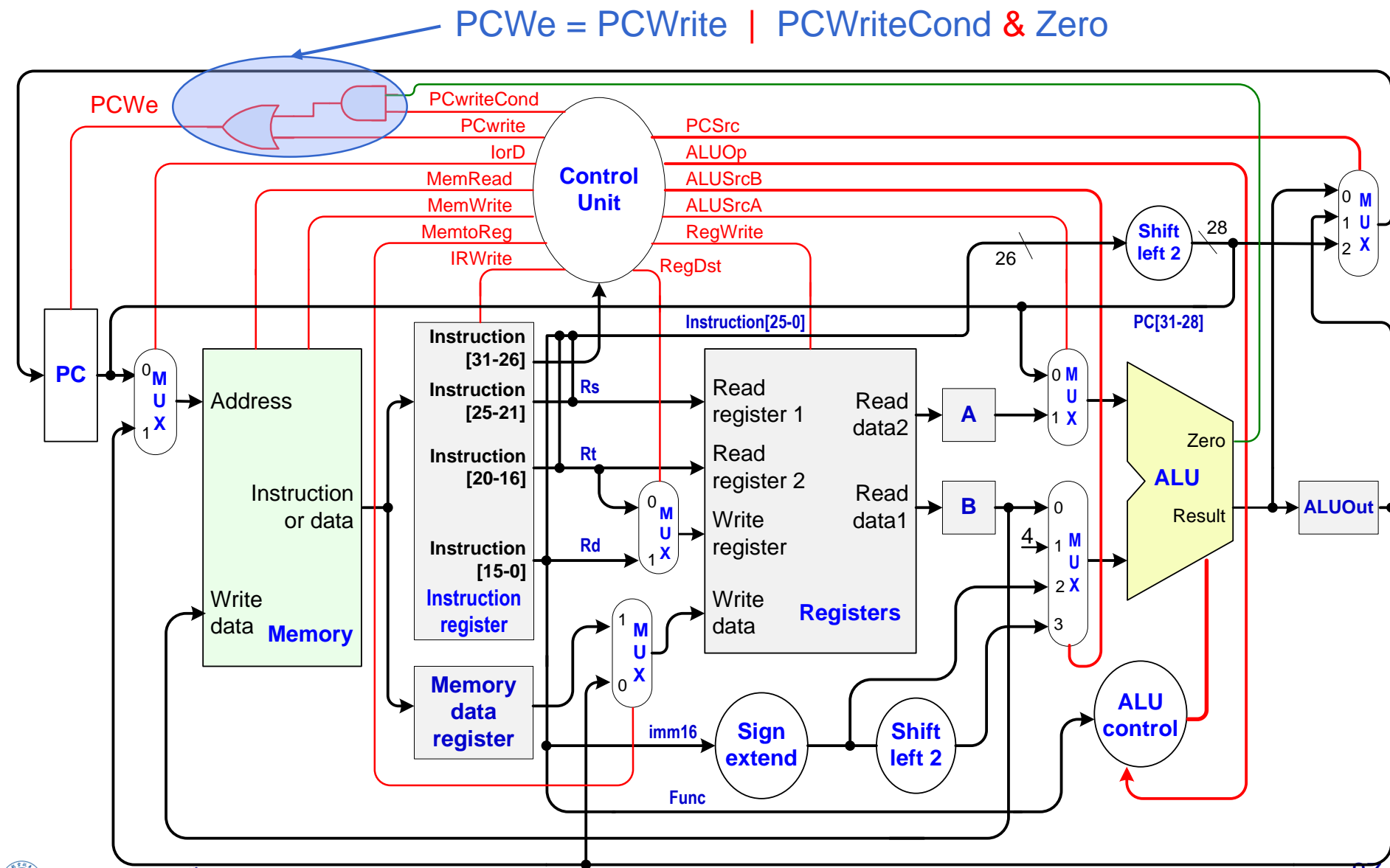
五 . MIPS流水线处理器设计

4.2 多周期控制器设计

❖ 数据通路控制信号设计



4.2 多周期控制器设计——完整数据通路与控制信号



4.2 多周期控制器设计

多周期通路控制信号

| 控制信号 | 失效时作用 | 有效时作用 |
|-------------|-------------------|---|
| RegDst | 寄存器堆写入端地址来选择Rt字段 | 寄存器堆写入端地址选择Rd字段 |
| RegWrite | 无 | 把数据写入寄存器堆中对应寄存器 |
| ALUSrcA | ALU输入A端选择PC | ALU输入A端选择寄存器A |
| MemRead | 无 | 存储器读数据（输出） |
| MemWrite | 无 | 存储器写数据（输入） |
| MemtoReg | 寄存器堆写入端数据选择ALUOut | 寄存器堆写入端数据选择DR |
| IorD | 存储器地址输入选择PC | 存储器地址输入选择ALUOut |
| IRWrite | 无 | 存储器输出（指令）写入IR |
| PCWrite | 无 | PC写入，PC输入源由PCSrc选择 |
| PCWriteCond | 无 | 如ALU的Zero端输出有效，则PC写入，输入源由PCSrc选择（Beq指令） |

4.2 多周期控制器设计

多周期通路控制信号（续）

| 控制信号 | 取值 (二进制) | 作用 |
|---------|-------------|--|
| ALUOp | 00 | ALU执行加法 |
| | 01 | ALU执行加法 |
| | 10 | ALU操作由Func字段（IR[5:0]）决定 |
| ALUSrcB | 00 | ALU输入端B数据源选择寄存器B |
| | 01 | ALU输入端B数据源选择常数4 |
| | 10 | ALU输入端B选择符号扩展输出（Signext(imm16)） |
| | 11 | ALU输入端B选择移位器输出（Signext(imm16)<<2） |
| PCSrc | 00 | PC输入源选择ALU输出（取指阶段，PC+4） |
| | 01 | PC输入源选择寄存器 ALUOut（Beq指令） |
| | 10 | PC输入源选择转移地址 $PC[31:28] \parallel IR[25:0] \ll 2$ （j指令） |

4.2 多周期控制器设计

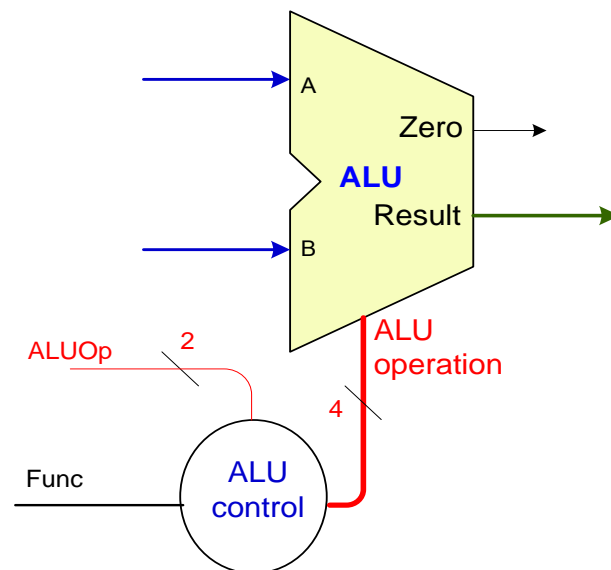
❖ ALU控制单元

➤ 输入:

- 指令的**Func**字段（指令**5:0**位）
- 由主控单元生成的 **ALUOp**

➤ **ALUOp**指明**ALU**的运算类型

- **00**: 访存指令所需的加法
- **01**: **beq**指令所需的减法
- **10**: **R**型指令功能码字段决定



| 指令 | Func字段 | ALUOp | ALU运算类型 | ALU operation |
|-----|---------|-------|---------|---------------|
| Lw | XXXXXX | 00 | 加 | 0010 |
| Sw | XXXXXX | 00 | 加 | 0010 |
| Beq | XXXXXX | 01 | 减 | 0110 |
| Add | 100 000 | 10 | 加 | 0010 |
| Sub | 100 010 | 10 | 减 | 0110 |
| And | 100 100 | 10 | 与 | 0000 |
| Or | 100 101 | 10 | 或 | 0001 |

ALU控制单元真值表

4.2 多周期控制器设计

多周期数据通路汇总

| 时钟周期 | R型指令 | Lw指令 | Sw指令 | Beq指令 | J指令 |
|------|---|--|--|--|-----|
| TC1 | $IR \leftarrow M[PC]$ $PC \leftarrow PC + 4$ | | | | |
| TC2 | $A \leftarrow R[IR[25:21]], B \leftarrow R[IR[20:16]]$ $ALUOut \leftarrow PC + \text{Signext}[IR[15:0]] \ll 2$ | | | | |
| TC3 | $ALUOut \leftarrow A \text{ op } B$ | $ALUOut \leftarrow A + \text{Signext}(IR[15:0])$ | $\text{If } (A-B==0) \text{ then } PC \leftarrow ALUOut$ | $PC \leftarrow PC[31:28] \parallel IR[25:0] \ll 2$ | |
| TC4 | $R[IR[15:11]] \leftarrow ALUOut$ | $DR \leftarrow M[ALUOut]$ | $M[ALUOut] \leftarrow B$ | | |
| TC5 | | $R[IR[20:16]] \leftarrow DR$ | | | |

4.2 多周期控制器设计

1. 数据通路控制信号状态（P0，所有指令TC1）

- ◆ $IR \leftarrow M[PC]$
- ◆ $PC \leftarrow PC + 4$

| 通路 部件 | 存储器 地址 MUX | 存储器 | IR | ALU-A 端MUX | ALU-B 端MUX | ALU | PC输入 端MUX | PC |
|----------------|------------------|---------|---------|----------------|-----------------|---------------|---------------|---------|
| 操作 | 选择PC | 读出 | 写入 | 选择PC | 常数4 | 加 | 选择ALU | 写入 |
| 控制 信号 状态 | lorD=0 | MemRead | IRWrite | ALUSrcA = 0 | ALUSrcB = 01 | ALUOp = 00 | PCSrc = 00 | PCWrite |

状态**ST0**:

- MemRead有效
- IRWrite有效
- PCWrite有效
- lorD = 0
- ALUSrcA = 0
- ALUSrcB = 01
- ALUOp = 00
- PCSrc = 00

4.2 多周期控制器设计

2. 数据通路控制信号状态（**P1**，所有指令**TC2**）

- ◆ $A \leftarrow R[IR[25:21]]$, $B \leftarrow R[IR[20:16]]$
- ◆ $ALUOut \leftarrow PC + \text{Signext}[IR[15:0]] \ll 2$

| 通路部件 | 寄存器堆 | ALU-A端MUX | ALU-B端MUX | ALU |
|--------|------|---------------|----------------|--------------|
| 操作 | 无 | 选择PC | 选择移位器 | 加 |
| 控制信号状态 | 无 | $ALUSrcA = 0$ | $ALUSrcB = 11$ | $ALUOp = 00$ |

状态**ST1**:

- $ALUScrA = 0$
- $ALUSrcB = 11$
- $ALUOp = 00$

4.2 多周期控制器设计

3. 数据通路控制信号状态（P2, Lw/Sw指令TC3）

◆ $ALUOut \leftarrow A + \text{Signext}(IR[15:0])$

| 通路 部件 | ALU-A端MUX | ALU-B端MUX | ALU |
|----------------|---------------|----------------|--------------|
| 操作 | 选择寄存器A | 选择 符号扩展 | 加 |
| 控制 信号 状态 | $ALUSrcA = 1$ | $ALUSrcB = 10$ | $ALUOp = 00$ |

状态**ST2**:

- $ALUSrcA = 1$
- $ALUSrcB = 10$
- $ALUOp = 00$

4.2 多周期控制器设计

4. 数据通路控制信号状态（P3, Lw指令TC4）

◆ $DR \leftarrow M[ALUOut]$

| 通路 部件 | 存储器地址端 MUX | 存储器 | 数据寄存器DR |
|----------------|---------------|---------|---------|
| 操作 | 选择ALUOut | 读出 | 写入 |
| 控制 信号 状态 | lorD = 1 | MemRead | 无 |

状态ST3:

- lorD = 1
- MemRead有效

4.2 多周期控制器设计

5. 数据通路控制信号状态（P4, Lw指令TC5）

◆ $R[IR[20:16]] \leftarrow DR$

| 通路 部件 | 寄存器堆写 地址MUX | 寄存器堆写 数据MUX | 寄存器堆 |
|----------------|----------------|----------------|----------|
| 操作 | 选择Rt | 选择DR | 写入 |
| 控制 信号 状态 | RegDst = 0 | MemtoReg = 1 | RegWrite |

状态ST4:

- RegDst = 0
- MemtoReg = 1
- RegWrite有效

4.2 多周期控制器设计

6. 数据通路控制信号状态（P5, Sw指令TC4）

◆ $M[ALUOut] \leftarrow B$

| 通路 部件 | 存储器 地址MUX | 存储器 | |
|----------------|--------------|----------|--|
| 操作 | 选择ALUOut | 写入 | |
| 控制 信号 状态 | lorD = 1 | MemWrite | |

状态ST5:

- lorD = 1
- MemWrite有效

4.2 多周期控制器设计

7. 数据通路控制信号状态（P6，R型指令TC3）

◆ $ALUOut \leftarrow A \text{ op } B$

| 通路 部件 | ALU-A端MUX | ALU-B端MUX | ALU |
|----------------|-------------|--------------|----------------|
| 操作 | 选择 寄存器A | 选择 寄存器B | 根据Func字段 运算 |
| 控制 信号 状态 | ALUSrcA = 1 | ALUSrcB = 00 | ALUOp = 10 |

状态ST6:

- ALUSrcA = 1
- ALUSrcB = 00
- ALUOp = 10

4.2 多周期控制器设计

8. 数据通路控制信号状态（P7，R型指令TC4）

◆ $R[IR[15:11]] \leftarrow ALUOut$

| 通路 部件 | 寄存器堆写 地址MUX | 寄存器堆写 数据MUX | 寄存器堆 |
|----------------|----------------|----------------|----------|
| 操作 | 选择Rd | 选择ALUOut | 写入 |
| 控制 信号 状态 | RegDst = 1 | MemtoReg = 0 | RegWrite |

状态ST7:

- RegDst = 1
- MemtoReg = 0
- RegWrite有效

4.2 多周期控制器设计

9. 数据通路控制信号状态（P8, Beq指令TC3）

◆ If ($A - B == 0$) then $PC \leftarrow ALUOut$

| 通路 部件 | ALU-A端 MUX | ALU-B端 MUX | ALU | PC输入端 MUX | PC |
|----------------|----------------|-----------------|---------------|---------------|-------------|
| 操作 | 选择 寄存器A | 选择 寄存器B | 减 | 选择 ALUOut | 有条件写入 |
| 控制 信号 状态 | ALUSrcA = 1 | ALUSrcB = 00 | ALUOp = 01 | PCSrc = 01 | PCWriteCond |

状态**ST8**:

- ALUSrcA = 1
- ALUSrcB = 00
- ALUOp = 01
- PCSrc = 01
- PCWriteCond有效

4.2 多周期控制器设计

10. 数据通路控制信号状态（P9，J指令TC3）

◆ $PC \leftarrow PC[31:28] \parallel IR[25:0] \ll 2$

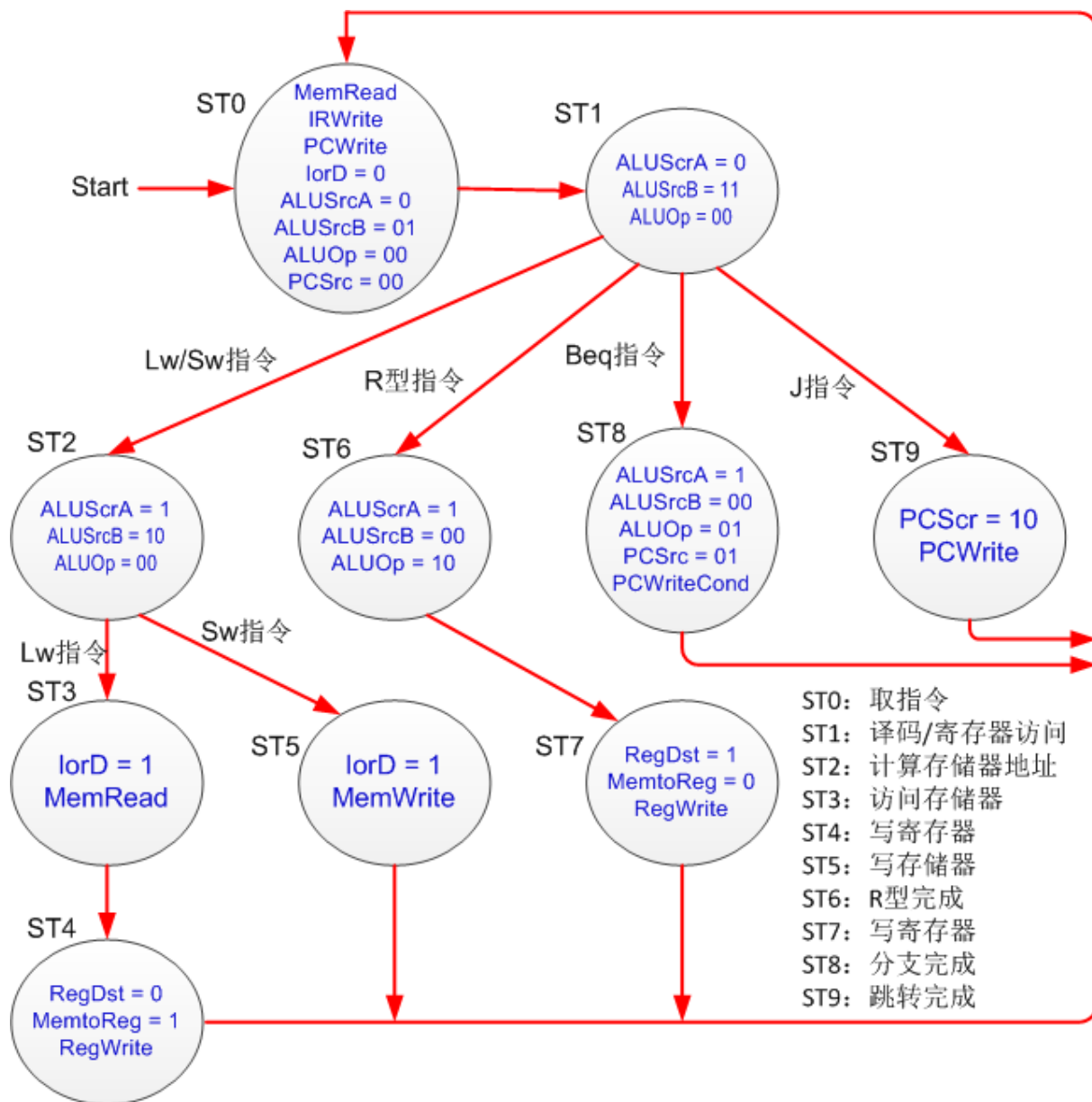
| 通路 部件 | PC输入端MUX | PC | |
|----------------|------------|---------|--|
| 操作 | 选择转移地址 | 写入 | |
| 控制 信号 状态 | PCSrc = 10 | PCWrite | |

状态ST9:

- PCSrc = 10
- PCWrite有效

4.2 多周期控制器设计——主控单元

多周期
主控单元
有限状态图



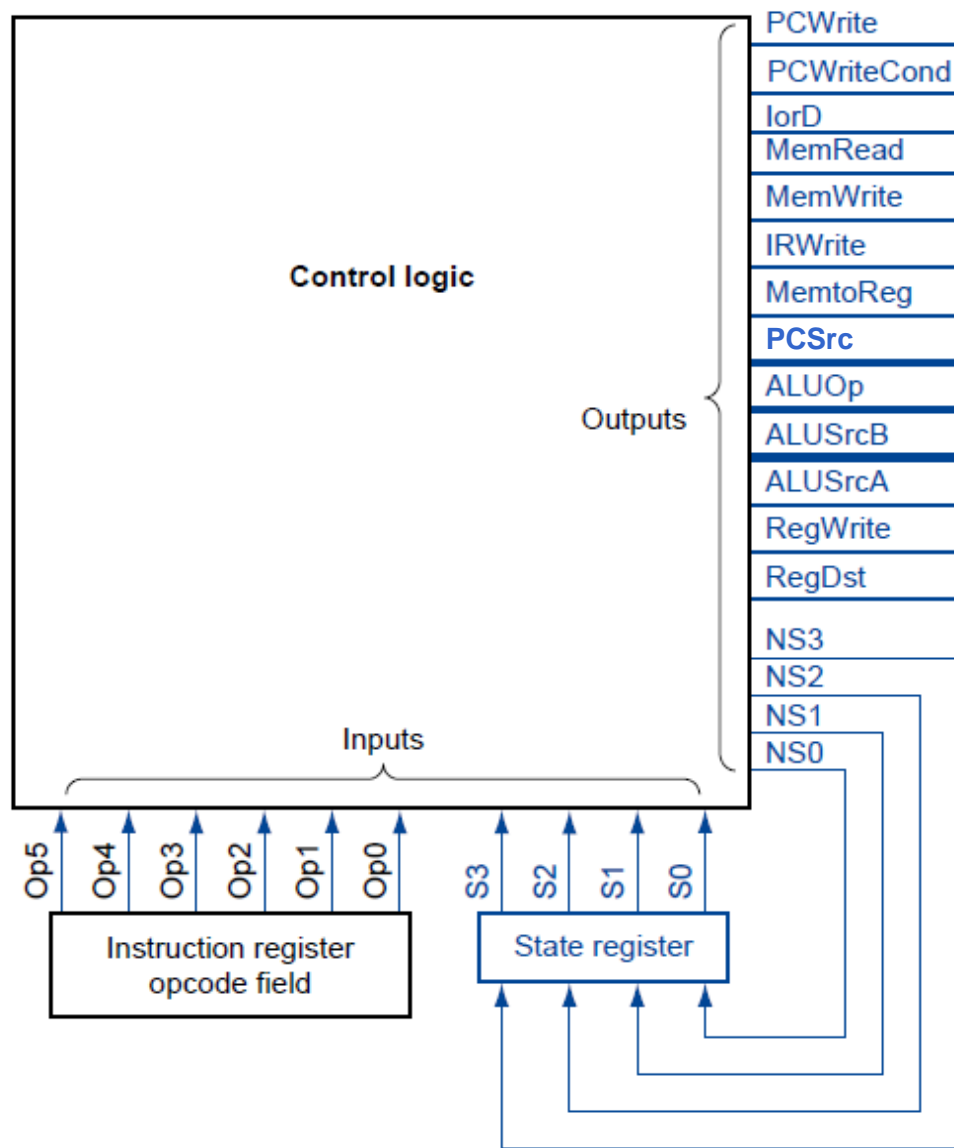
4.2 多周期控制器设计——主控制单元

❖ 有限状态机FSM

- ST0、ST1等10个状态
- 状态编码用4位二进制表示，如 $S_3S_2S_1S_0=0110$ 表示ST6
- 次态用 $NS_3NS_2NS_1NS_0$ 表示

❖ 摩尔型FSM

- 控制信号输出仅取决于当前状态
- 次态由现态和当前输入（IR寄存器中6位操作码）决定



4.2 多周期控制器设计——主控制单元

❖ 控制信号逻辑示例

控制信号仅和当前状态有关

$$PCWrite = ST0 + ST9$$

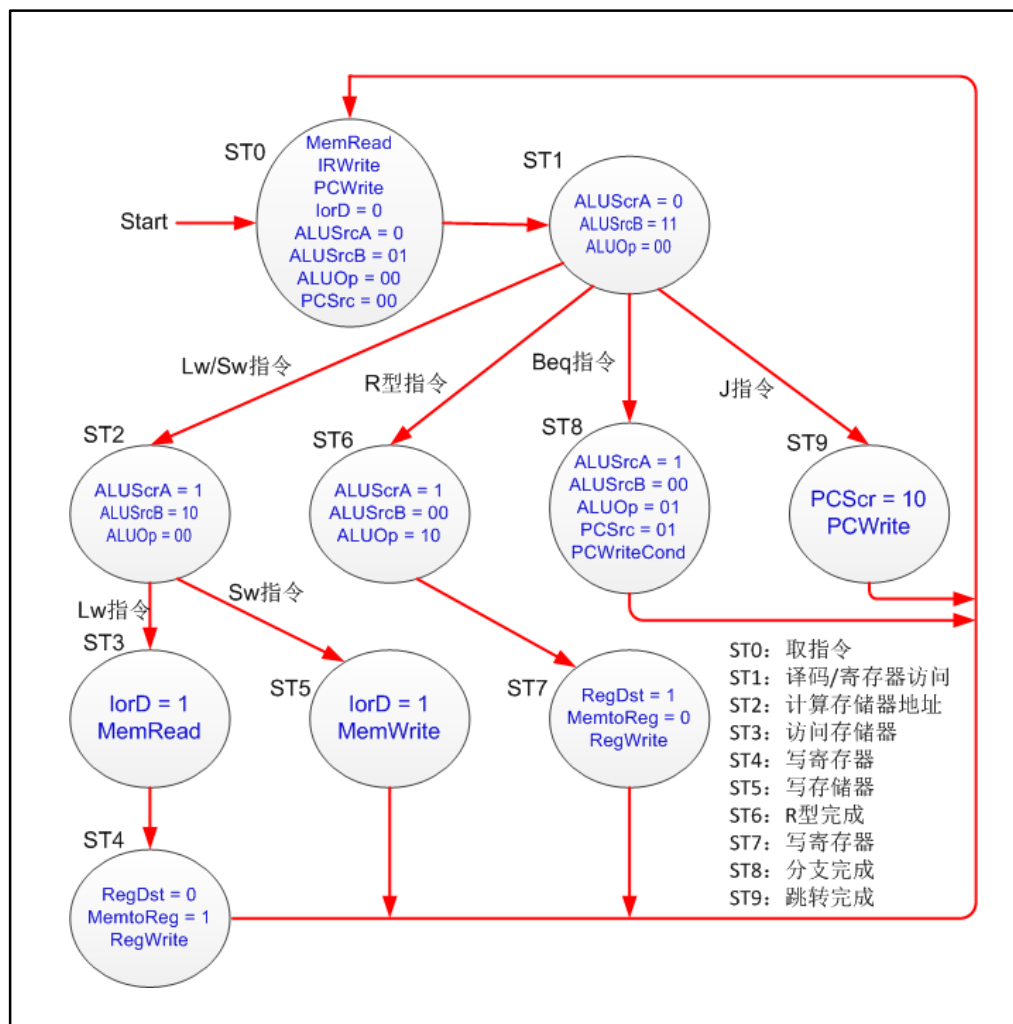
$$= \overline{S_3} \overline{S_2} \overline{S_1} \overline{S_0} + S_3 \overline{S_2} \overline{S_1} S_0$$

$$ALUSrcA = ST2 + ST6 + ST8$$

$$= \overline{S_3} \overline{S_2} \overline{S_1} \overline{S_0} + \overline{S_3} \overline{S_2} \overline{S_1} S_0 + S_3 \overline{S_2} \overline{S_1} \overline{S_0}$$

$$RegWrite = ST4 + ST7$$

$$= \overline{S_3} \overline{S_2} \overline{S_1} \overline{S_0} + \overline{S_3} \overline{S_2} S_1 S_0$$



4.2 多周期控制器设计——主控制单元

❖ 次态信号逻辑，次态输出与当前状态和当前输入（**Opcode**）相关

次态信号真值表

| 当前状态输入 $S_3S_2S_1S_0$ | 操作码输入 ($Op_5Op_4Op_3Op_2Op_1Op_0$) | 次态输出 $NS_3NS_2NS_1NS_0$ |
|----------------------------|---|----------------------------|
| 0000 (ST0) | | 0001 (ST1) |
| 0001 (ST1) | 100011 (lw指令) | 0010 (ST2) |
| 0001 (ST1) | 101011 (sw指令) | 0010 (ST2) |
| 0001 (ST1) | 000000 (R型指令) | 0110 (ST6) |
| 0001 (ST1) | 000100 (Beq指令) | 1000 (ST8) |
| 0001 (ST1) | 000010 (J指令) | 1001 (ST9) |
| 0010 (ST2) | 100011 (lw指令) | 0011 (ST3) |
| 0010 (ST2) | 101011 (sw指令) | 0101 (ST5) |
| 0011 (ST3) | | 0100 (ST4) |
| 0110 (ST6) | | 0111 (ST7) |
| ST4, ST5, ST7, ST8, ST9 | | 0000 (ST0) |

$$NS_3 = ST1(Beq + J)$$

$$= \overline{S_3} \overline{S_2} \overline{S_1} \overline{S_0} (\overline{Op_5} \overline{Op_4} \overline{Op_3} \overline{Op_2} \overline{Op_1} \overline{Op_0} + \overline{Op_5} \overline{Op_4} \overline{Op_3} \overline{Op_2} \overline{Op_1} \overline{Op_0})$$