7.2.2 单周期处理器的数据通路和指令的执行过程

本小节采用数据通路设计方法中的第一种方法，设计一个不同于前述的单时钟周期MIPS CPU的数据通路。设计的步骤是：

1　分析MIPS的指令格式，分析每种格式下指令的功能；

2　根据指令功能，罗列所需器件和器件之间连接的方式；

3　确定每个器件所需的控制信号。

根据这些控制信号可以得到指令与信号的对应关系表及控制信号的逻辑表达式，据此进一步设计译码和控制单元的电路。

32位MIPS的每条指令固定长度是32位，其指令有R型、I型和J型三种指令格式，错误：引用源未找到给出了三种指令的格式。

表7-2 MIPS指令格式

指令类型	指令格式（共32位）						备注
	31-26位	25-21位	20-16位	15-11位	10-6位	5-0位	
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)	算术指令
I	opcode (6)	rs (5)	rt (5)	offset/immediate (16)			数据传输、分支、访存、立即数指令
J	opcode (6)	address (26)					跳转指令

根据7.1节可知，控制器为了实现取指令的功能应该配备的器件有程序计数器PC，根据程序计数器PC提供的地址，从指令存储器中读出指令，因此设计取指令的器件和数据通路如图7-所示。

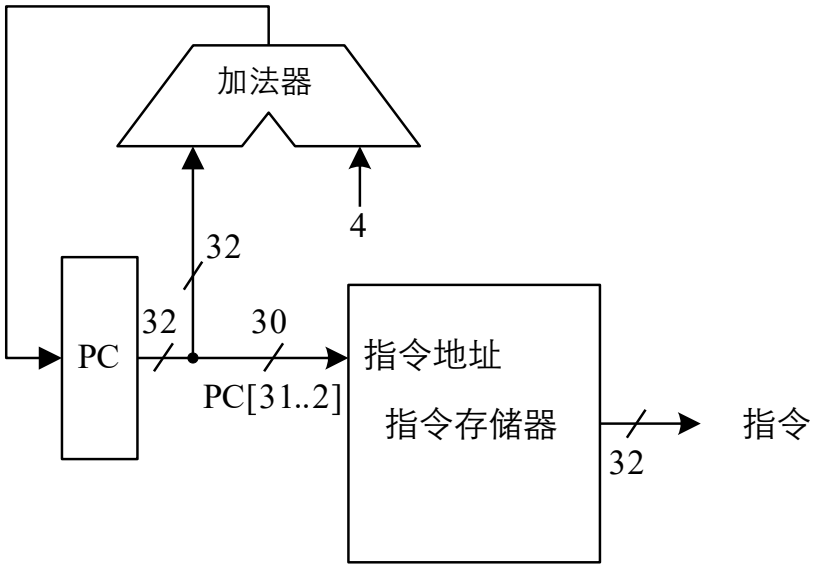


图 7-18 取指令及数据通路

图 7-中指令存储器是一个只读存储器，根据 PC 提供的地址来读取对应单元存放的指令代码，因此不需要读控制信号。MIPS32 指令的长度是 32 位，因此指令存储器宽度也设计为 32 位，PC 的 32 位地址对应的是字节地址，因此实际访存是取 PC 的高 30 位为地址来访存，PC 的最后 2 位用于选择读出的 32 位指令字中的哪一个字节。一次访存取出一条指令（4 个字节），如图 7-19 指令存储器地址与内容示意图所示。

存储器内部地址 (PC高30位地址)	存储单元内容			
0	字节3	字节2	字节1	字节0
1	字节7	字节6	字节5	字节4
⋮		⋮		
n	字节 4n+3	字节 4n+2	字节 4n+1	字节 4n

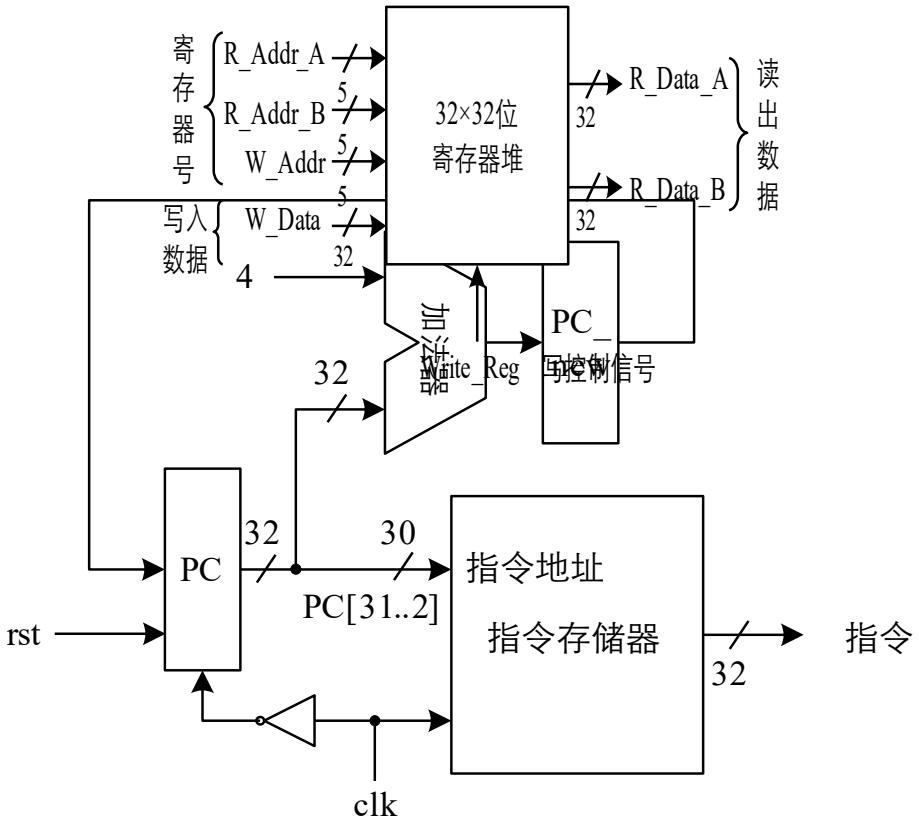
图 7-19 指令存储器地址与内容示意图

按照常规，读出的指令代码应该存入指令寄存器 IR 中，以保证当前指令代码在执行指令期间保持不变。但是，由于此指令存储器是只读存储器，只要指令地址在执行指令期间保持不变，那么从指令存储器取出的指令代码就会维持不变，因此，取消了指令寄存器的设置，简化了数据通路。

在图 7-中，除了程序计数器 PC 和指令存储器而外，还有一个器件是 PC 自增加法器，用来完成 PC 的自增操作。根据控制器的功能可知，CPU 每次根据 PC 从指令存储器取完一条指令后，应该使 PC 值自增加，使 PC 指向下一条指令。由于 MIPS32 指令字长 32 位，指令存储器按字节编址，因此，每次取指令后，PC 应该自增 4，图 7-18 中的加法器没有其他

运算功能，只能完成加法运算。

单周期 MIPS CPU 要求在每个时钟周期执行一条指令。为保持在执行指令期间，指令存



储器取出的指令不变，PC 值在执行指令期间必须保持不变，因此 PC+4 的值应该在下个指令周期开始时赋值给 PC。为此，在图 7-基础上，添加一个暂存 PC 自增值（PC+4）的寄存器 PC_new。在指令周期（也就是时钟周期）clk 的上跳沿，执行取指令操作，在 clk 下跳沿更新 PC 值。新的数据通路如图 7-20 取指令所示。系统启动时 PC 复位，因此错误：引用源未找到中还添加了复位信号 rst，当 rst=1 时，PC 清零。

图 7-20 取指令数据通路

基于以上的结果，分别选取 R、I、J 三种格式的典型指令来讲述根据指令功能搭建系统结构、设计数据通路的方法。

1、R 型指令数据通路的设计

从 MIPS32 核心指令集和表 7-2 可以看到，所有 R 型指令的共同特征是：操作码字段 OP=000000b，指令的功能则由功能码字段 func 指出。每条 R 型指令的操作数有 3 个，两个源操作数分别在 rs 和 rt 字段所指定的寄存器中，而目的操作数则位于 rd 字段所指定的寄存器中。为实现 R 型指令，应添加多端口寄存器堆和多功能运算器这 2 个器件，如错误：引用源未找到和图 7-22 所示。

图 7-21 多端口寄存器堆

错误：引用源未找到是一个 32×32 位的寄存器堆，共含有 32 个寄存器，每个寄存器 32 位，寄存器地址 5 位。该寄存器堆有 2 个读端口和 1 个写端口，能够同时读出 2 个寄存器的值，写入一个寄存器。R_Addr_A 和 R_Addr_B 分别是两个读端口的输入地址，读出的数据分别由 R_Data_A 和 R_Data_B 输出。写寄存器地址是 W_Addr，数据由 W_Data 写入，写操作需要写控制信号 Write_Reg，且写操作是边沿触发的，所有写入操作的输入信号必须在时钟边沿来临时已经有效。

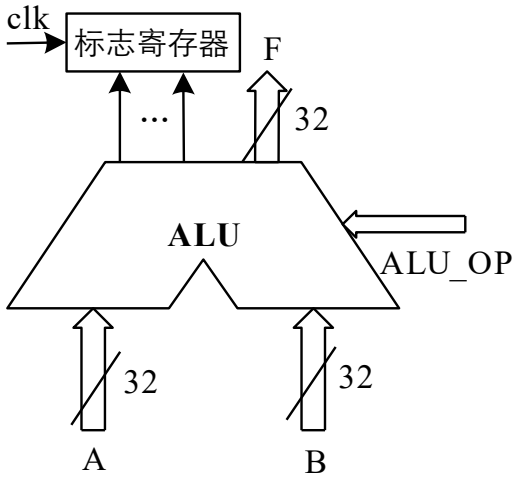
图 7-22 多功能运算器 ALU

图 7-22 所示的多功能运算器 ALU 是一个具有算术运算、逻辑运算、比较置数、移位运算等多种功能的 32 位 ALU，并能够产生运算结果的标志，置入标志寄存器。该部件运算不需要时钟，但是需要时钟脉冲才能置标志。

把图 7-21 所示的多端口寄存器堆和图 7-22 所示的多功能运算器 ALU，结合到图 7-20 中，建立数据通路，如图 7-23 所示。在图 7-23 中，控制信号都是由译码及控制单元发出，而译码及控制单元可以采用硬布线方式或微程序方式实现。

从指令存储器取出的指令，经过初级译码，将分解出的两个源寄存器 rs、rt 直接与寄存器堆的两个读端口 A 和 B 的寄存器地址连接，将目的寄存器 rd 字段与寄存器堆的写端口地址相连，寄存器读出的 A 口数据和 B 口数据直接连接到 ALU 的输入端 A 和 B，ALU 计算后的结果则送入寄存器堆的写数据端口。

当前指令的操作码 OP 和 func 字段交由指令译码部件处理：它首先会根据 OP 字段是否



全 0 来判定是不是 R 型指令，是，则再将 func 字段翻译成 ALU 的控制信号 ALU_OP，以指定 ALU 的运算功能。

在 clk 的上升沿，启动指令存储器依据 PC 读出指令。在 clk 高电平持续期间，完成 PC 值的自增、指令译码、寄存器读操作，随后完成 ALU 运算。在 clk 的下降沿完成目的寄存器的写入、PC 值的更新和标志寄存器的更新。在错误：引用源未找到中，将 clk 反相后作为寄存器堆、PC 和标志寄存器的打入脉冲。

图 7-23 多端口寄存器堆和运算器

不是所有指令的执行结果都影响标志位，也就是说，不是所有指令执行后都对标志寄存器置位。指令对标志寄存器的影响一般遵循以下规律：

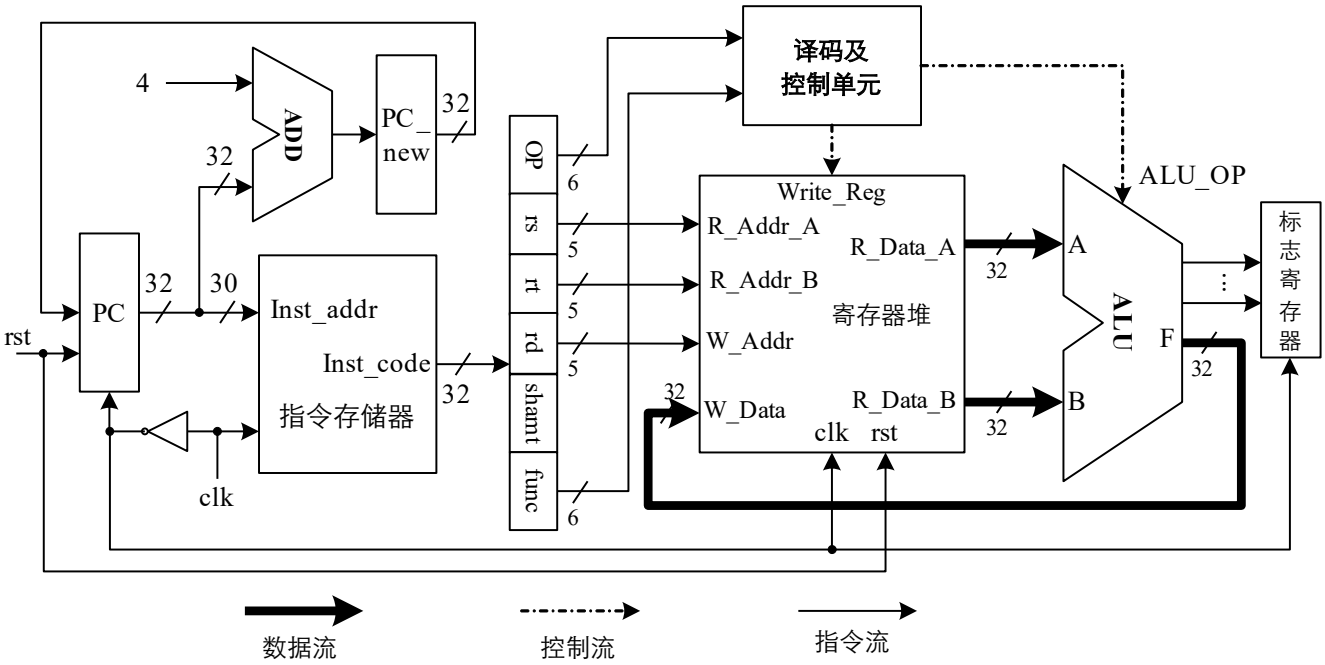
1　传送类指令和跳转类指令不影响标志位；

2　有符号算术运算类指令（包括 slt 和算术移位指令）影响 ZF 和 OF；

3　无符号算术运算类指令和逻辑运算类指令影响 ZF，不影响 OF；

4　条件转移类指令一般会使用标志位 ZF。

以 R 型加法指令 add rd,rs,rt（功能是算术加：rs + rt→rd）为例，分析数据通路。在 clk 的上升沿，启动指令存储器依据 PC 读出 add 指令。在 clk 高电平持续期间，PC 值加 4 并置入 PC_new 寄存器，指令译码，寄存器堆的读端口 A、B 分别以指令的第 25-21 位 rs 和指令的第 20-16 位 rt 作为寄存器号，同时读寄存器，由 R_Data_A 和 R_Data_B 分别输出 rs 和 rt



寄存器的内容，译码和控制单元把 func 字段翻译成 ALU 的控制信号 ALU_OP，ALU 按 ALU_OP 控制信号所指定的操作类型，对这两个操作数进行加法操作。在 clk 的下降沿，将加法运算的结果写入目的寄存器 rd，其地址由 W_Addr 指定，根据运算结果置标志寄存器，用 PC_new 的内容更新 PC 值。

类似的，译码和控制单元根据不同指令的 func 字段，在指令执行期间发送不同的操作类型控制信号 ALU_OP，以使在 ALU 中进行不同指令所对应的运算。而这些 R 型指令的数据通路和指令执行的时序与 add rd,rs,rt 基本相同。

2、I 型指令数据通路的设计

表 7-3 MIPS I 型立即数寻址指令格式及编码

字段	OP	rs	rt	imm	功能描述
位数	6	5	5	16	
汇编助记符	编码				
addi rt, rs, imm	001000	rs	rt	imm	算术加：rs + imm→rt
lw rt, offset(rs)	100011	rs	rt	offset	取数：（rs +offset）→rt
sw rt, offset(rs)	101011	rs	rt	offset	存数：rt →（rs +offset）

如表 7-2 所示，数据传输类指令、访存指令、分支转移指令和含立即数的指令一般采用 I 型指令格式。把其中 I 型格式的分支转移指令的数据通路与 J 型格式的转移指令的数据通路放在一起，在后续篇幅中介绍。下面，以表 7-3 所示的 addi 指令和两条访存指令 lw/sw 为例，分别分析立即数寻址指令的数据通路和访存指令的数据通路，再综合图 7-23，得出实现 R+I 型指令的 CPU 结构及数据通路。

首先分析 I 型立即数寻址指令 addi rt, rs, imm，发现与 R 型指令有明显不同的是 I 型指令没有 rd 寄存器，而使用 rt 作为目的寄存器。I 型立即数寻址指令的源操作数中有一个为立即数，位于指令的低 16 位。显然错误：引用源未找到的数据通路已经不能够实现 I 型立即数寻址指令。为实现 I 型立即数寻址指令，需要在图 7-23 基础上修改数据通路，修改如下：

1　增加二选一数据选择器，支持目的寄存器的两种来源

对于 R 型指令，目的寄存器是 rd；而对于 I 型指令，目的寄存器是 rt。为此，应该在指令机器码输出与寄存器堆模块的写地址输入端口前，设置一个二选一选择器，假定控制信号是 rd_rt_s，那么当 rd_rt_s=0 时，将指令的 rd 字段送入写地址 W_Addr；当 rd_rt_s=1 时，将指令的 rt 字段送入写地址 W_Addr。

2　增加器件，使指令低 16 位的立即数 imm 经过扩展，与 rs 执行运算操作

对于有符号数的操作来说，执行的是符号扩展，符号扩展是用 16 位立即数的最高位（补码的符号）填充高 16 位，从而构成 32 位的数；对于无符号数的运算，执行的是 0 扩展，即用 0 来填充高 16 位。设置一位控制信号 imm_s 来控制这两种扩展：当 imm_s=1 时进行符号扩展；当 imm_s=0 时进行 0 扩展。

扩展选择是由指令来决定的。对于有符号数的运算指令（譬如 addi）和存数/取数指令中的地址偏移量 offset，均需要符号扩展；而对于无符号数的运算指令（譬如 addiu）和逻辑运算类指令则应该执行 0 扩展。

3　增加二选一数据选择器，支持 ALU 运算数据的两种来源

对 ALU 运算来说，R 型指令是执行 rs 和 rt 运算，结果送 rd。而 I 型立即数寻址指令则是执行 rs 和扩展后的立即数 imm 运算，结果送 rt。因此，基于 R 型指令 CPU 的数据通路，ALU 的输入数据 B 端经修改，应该有两个选择：rt 或者 imm，通过设置二选一数据选择器来选择。假定控制信号为 rt_imm_s，那么当 rt_imm_s=0 时，将寄存器堆 B 端口读出的数据 R_Data_B 送 ALU 的 B 端；当 rt_imm_s=1 时，将扩展好的立即数 imm_data 送 ALU 的 B 输入端。

按照上述思路，基于图 7-23 改造的数据通路如??所示。

图 7-24 I 型立即数寻址指令的数据通路

MIPS 指令系统中只有取数/存数指令可以直接访问存储器，其他指令则对寄存器型操作数和立即数型操作数进行处理。MIPS 的两条存储器访问指令的格式和编码如错误：引用源未找到所示。

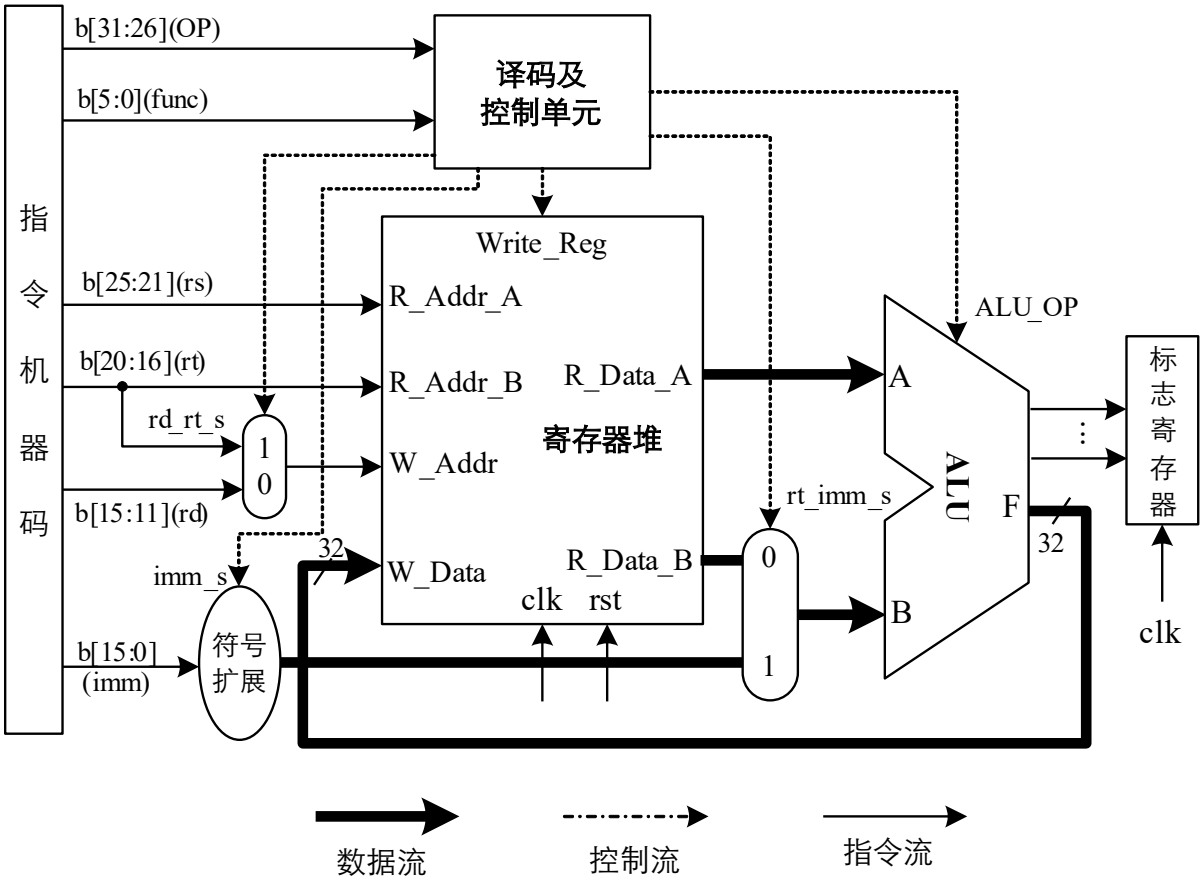
错误：引用源未找到的 lw/sw 指令中，offset 是有符号的 16 位相对偏移量。lw/sw 指令的存储器操作数有效地址 EA=（rs）+ offset，是相对寄存器寻址。取数指令 lw 将 EA 指定存储器地址中数据读出，写入 rt 寄存器；存数指令 sw 则将 rt 寄存器中数据写入 EA 所指定的存储器单元地址中。偏移量 offset 需要经过符号扩展后，与 rs 寄存器中的基址相加，得到存储器的访问地址 EA。

同理，在??所示数据通路的基础上，做一些变动，以实现表 7-3 所示的两条访存指令。分析如下：

1　需要添加一个数据存储器 RAM，存放指令访问的数据。该存储器读操作不需要脉冲，写操作需要脉冲，假定读写控制信号是 Mem_Write，当 Mem_Write=0 时读存储器，当 Mem_Write=1 且 clk 下降沿（反相）到来时写存储器。

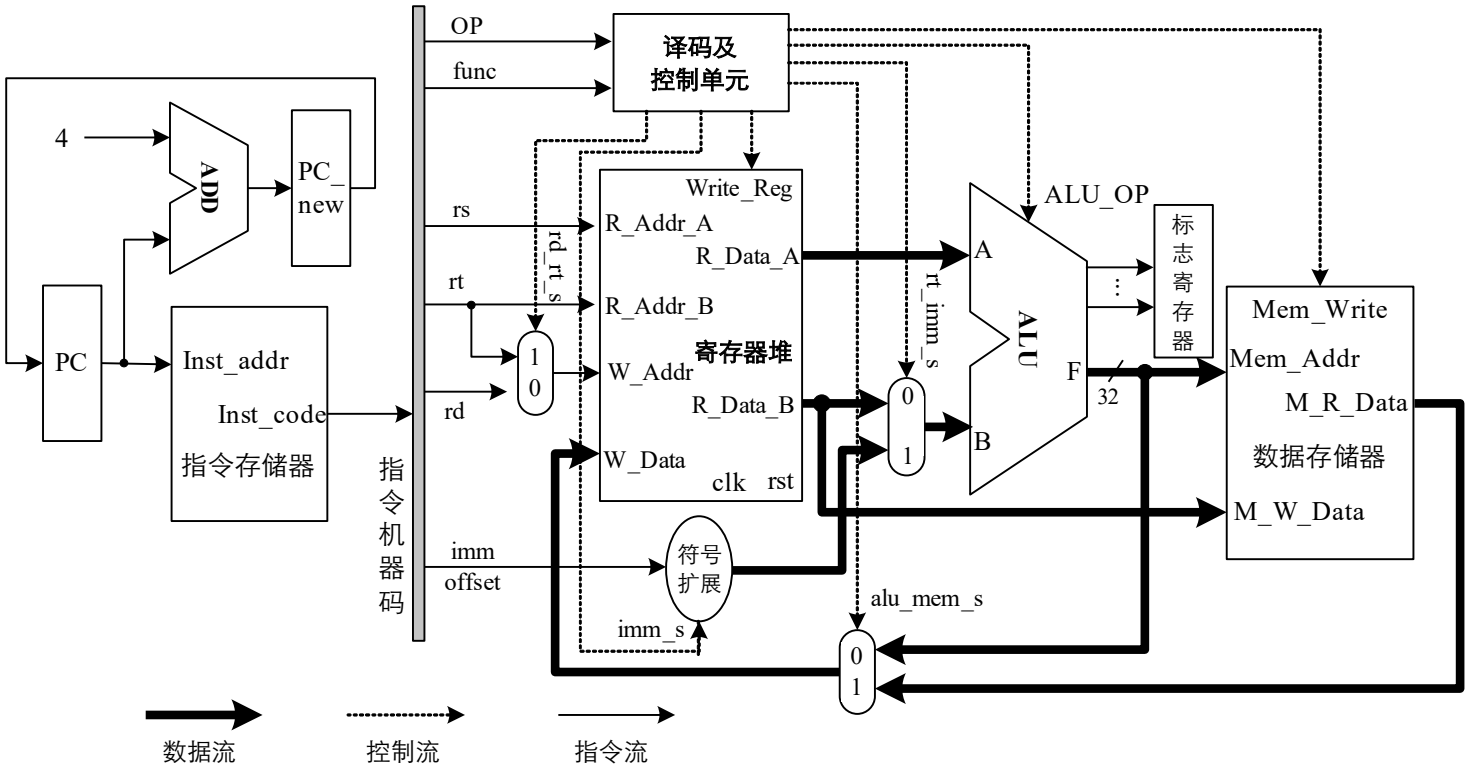
2　使用 ALU 来实现对有效地址 EA 的计算。ALU 的加数和被加数是 rs 和经过符号扩展的 offset，计算得到的 EA，可以由 ALU 的输出直接送存储器地址端口。

3　存储器读出的数据：分析 lw 指令，需写入寄存器 rt，这意味着：寄存器堆的写入



数据多了一个选择。如图 7-24，写端口的地址处，通过设置控制信号 rd_rt_s=1 来选择写入 rt 寄存器。在寄存器堆的写入数据端口 W_Data 处，需要另设置一个二选一多路选择器，由信号 alu_mem_s 控制，当 alu_mem_s=0 时，将 ALU 的输出送寄存器堆的写数据端口；当 alu_mem_s=1 时，将存储器的读出数据送寄存器堆的写数据端口。

4　存储器的写入数据：分析 sw 指令，需要将 rt 读出的数据写入存储器。由于 rt 可通过寄存器堆的 B 端口读出数据，因此可以将寄存器堆的 B 端口数据直接送至存储器的写数据



端口。

由以上分析得到的新的完整的数据通路，如错误：引用源未找到所示。为突出主要部分，图中省略不画 clk 和 rst 信号。相比??，图 7-25 中多了一个数据 RAM 存储器及其地址和读出数据的连接；在寄存器堆的写端口处，多了一个二选一通道选择器。

在图 7-25 中，指令存储器和数据存储器是分别独立的两个存储器，称为哈佛结构。在图 7-25 中分析 addi rt, rs, imm（算术加 rs + imm→rt）指令的数据通路。在 clk 的上升沿，启动指令存储器依据 PC 读出 addi 指令。在 clk 高电平持续期间，PC 值加 4 并置入 PC_new 寄存器，指令译码，寄存器堆的读端口 A 以指令的第 25-21 位 rs 读寄存器，读出 rs 读寄存器的内容输出到 ALU。指令的第 15-0 位立即数 imm 经过符号扩展（控制信号 imm_s=1），输出到 ALU（控制信号 rt_imm_s=1），与读出 rs 寄存器的内容相加，在 clk 的下降沿，将加法运算的结果写入寄存器 rt（控制信号 alu_mem_s=0），其地址由 W_Addr 根据指令的第 20-16 位 rt 寄存器号指定（控制信号 rd_rt_s=1）。根据运算结果置标志寄存器，用 PC_new 的内容更新 PC 值。

图 7-25 R-I 型指令的数据通路

分析 lw rt, offset(rs)（取数：（rs +offset）→rt）指令的数据通路。在 clk 的上升沿，启动指令存储器依据 PC 读出 lw 指令。在 clk 高电平持续期间，PC 值加 4 并置入 PC_new 寄存器，指令译码，寄存器堆的读端口 A 以指令的第 25-21 位 rs 读寄存器，读出 rs 寄存器的内容输出到 ALU。指令的第 15-0 位地址偏移量 offset 经过符号扩展（控制信号 imm_s=1），输出到 ALU（控制信号 rt_imm_s=1），与 rs 寄存器的内容相加，以加法运算的结果作为地址访存（控制信号 Mem_Write=0）。在 clk 的下降沿，存储器读出的数据写入寄存器 rt，其地址由 W_Addr 根据指令的第 20-16 位 rt 寄存器号指定（控制信号 rd_rt_s=1）。同时用 PC_new 的内容更新 PC 值。

类似的，在图 7-25 中可以分析得到其他（除分支转移指令外）的 I 型指令的数据通路。

3、转移指令数据通路的设计

表 7-4 三种格式的转移指令举例

R 型指令	字段	OP	rs	rt	rd	shamt	func	功能描述
	位数	6	5	5	5	5	6	
jr rs		000000	rs	00000	00000	00000	001000	无条件跳转： rs → PC
I 型指令	字段	OP	rs	rt	offset			功能描述
	位数	6	5	5	16			
beq rs, rt, label		000100	rs	rt	offset			相等转移：if(rs=rt) then PC+4+offset×4→PC else PC+4→PC
J 型指令	字段	OP	address					功能描述
	位数	6	26					
jal label		000011	address					无条件跳转： (PC+4)→$31, {(PC+4)高 4 位, address,0,0}→PC

表 7-4 分别给出了 R、I、J 三种格式的转移指令。其中 R 型格式的 jr 指令和 J 型格式的 jal 指令都是无条件转移指令，而 I 型格式的 beq 指令是条件转移指令。

无条件转移指令中，jal 指令不仅转移，而且在转移前，将 jal 指令的下一条指令的地址保存到编号为 31 的$ra（$31）寄存器中。R 型格式的无条件转移指令 jr 指令，将 rs 寄存器中的 32 位数据当做指令地址，直接置入 PC。jr 指令通常和 jal 指令搭配使用，用于子程序的调用与返回。jal 指令相当于 call（子程序调用）指令，jr 指令相当于 ret（子程序返回）指令。

I 型条件转移指令 beq rs, rt, label 的功能是，相等转移 if(rs=rt) then PC+4+offset×4→PC else PC+4→PC。其中 offset 是有符号数，是跳转目标指令的地址相对于下一条指令的地址的偏移量。offset 需要符号扩展为 32 位后，再左移两位，与新的 PC 值（PC+4）相加，得到转移地址，当条件满足时，即 rs=rt，发生转移，否则顺序执行下一条指令。

分析表 7-4 给出的三条转移指令可知，转移地址的产生方法有 3 种：

1　rs；

2　PC+4+offset×4；

3　{(PC+4)高 4 位,address,0,0}；

转移地址产生后，要送入 PC 才能完成跳转。对于 PC 来说，如果加上 PC 自增，则 PC 的来源有 4 种，需要一个 4 选 1 多路选择器，命由 2 位控制信号 PC_s[1:0]来选择。

接下来需要考虑如何产生转移地址：

1　对于 PC 自增，则使用 PC_new（PC+4）；

2　对于 rs，直接使用寄存器堆的读出 A 数据端口；

3　对于相对转移，则需要添加一个地址加法器，将 PC_new 和符号扩展并左移 2 位后的 offset 相加；

4　对于页面寻址的转移地址，则需要简单的左移和拼接操作。

jal 指令需要将 PC+4 的值存入$ra（$31）寄存器中，因此寄存器写入地址的选择需要多

一种可能性（即$31），对于写入数据的选择也需要多一种可能（即 PC+4），因此需要修改寄存器堆在写入地址前的多路选择器和写入数据前的多路选择器。

beq 指令需要根据减法的结果是否为（全）零来判断 rs 和 rt 是否相等，因此要根据标志 ZF 来判断是否转移。??显示了能实现 R、I、J 型指令的 CPU 的完整结构和数据通路（同样的原因，在??中也省略不画出 clk 和 rst 信号）。控制信号 w_r_s 用于选择写入的寄存器地址来源，wr_data_s 信号用于选择写入寄存器的数据来源。

在图 7-26 中，分析 jr rs（无条件跳转：rs → PC）指令的数据通路。在 clk 的上升沿，启动指令存储器依据 PC 读出 jr 指令。在 clk 高电平持续期间，指令译码，PC 值加 4 并置入 PC_new 寄存器，寄存器堆的读端口 A 以指令的第 25-21 位 rs 读寄存器，在 clk 的下降沿，将读出 rs 寄存器的内容更新 PC 值，控制信号 PC_s[1:0]=1。

分析 beq rs, rt, label（相等转移：if(rs=rt) then PC+4+offset×4→PC else PC+4→PC）指令的数据通路。在 clk 的上升沿，启动指令存储器依据 PC 读出 beq 指令。在 clk 高电平持续期间，指令译码，PC 值加 4 并置入 PC_new 寄存器，指令的第 15-0 位 offset 左移 2 位后与 PC_new 的内容一起送到地址加法器 ADD 中相加，寄存器堆的读端口 A、B 分别以指令的第 25-21 位 rs 和指令的第 20-16 位 rt 作为寄存器号，同时读寄存器，由 R_Data_A 和 R_Data_B 分别输出 rs 和 rt 寄存器的内容到 ALU 中相减。在 clk 的下降沿，根据运算结果置标志寄存器，根据 ZF 标志的值选择用 PC_new（PC+4）的内容，或从地址加法器 ADD 得到的相加的结果（ PC+4+offset×4）来更新 PC 值，控制信号 PC_s[1:0]分别=0 或=2。

分析 jal label（无条件跳转：(PC+4)→$31, {(PC+4)高 4 位,address,0,0}→PC）指令的数据通路。在 clk 的上升沿，启动指令存储器依据 PC 读出 jal 指令。在 clk 高电平持续期间，指令译码，PC 值加 4 并置入 PC_new 寄存器，指令的第 25-0 位 address 左移 2 位并低 2 位补 00 后，与 PC_new 的高 4 位拼接在一起，构成 32 位的转移地址。在 clk 的下降沿，PC_new 寄存器的内容写入$ra（即$31）寄存器，此时控制信号 w_r_s=2，wr_data_s=2。用拼接得到的 32 位转移地址来更新 PC 值，控制信号 PC_s[1:0]=3。

类似的，在图 7-26 中可以分析得到其他 I 型分支转移指令和 J 型转移指令的数据通路。

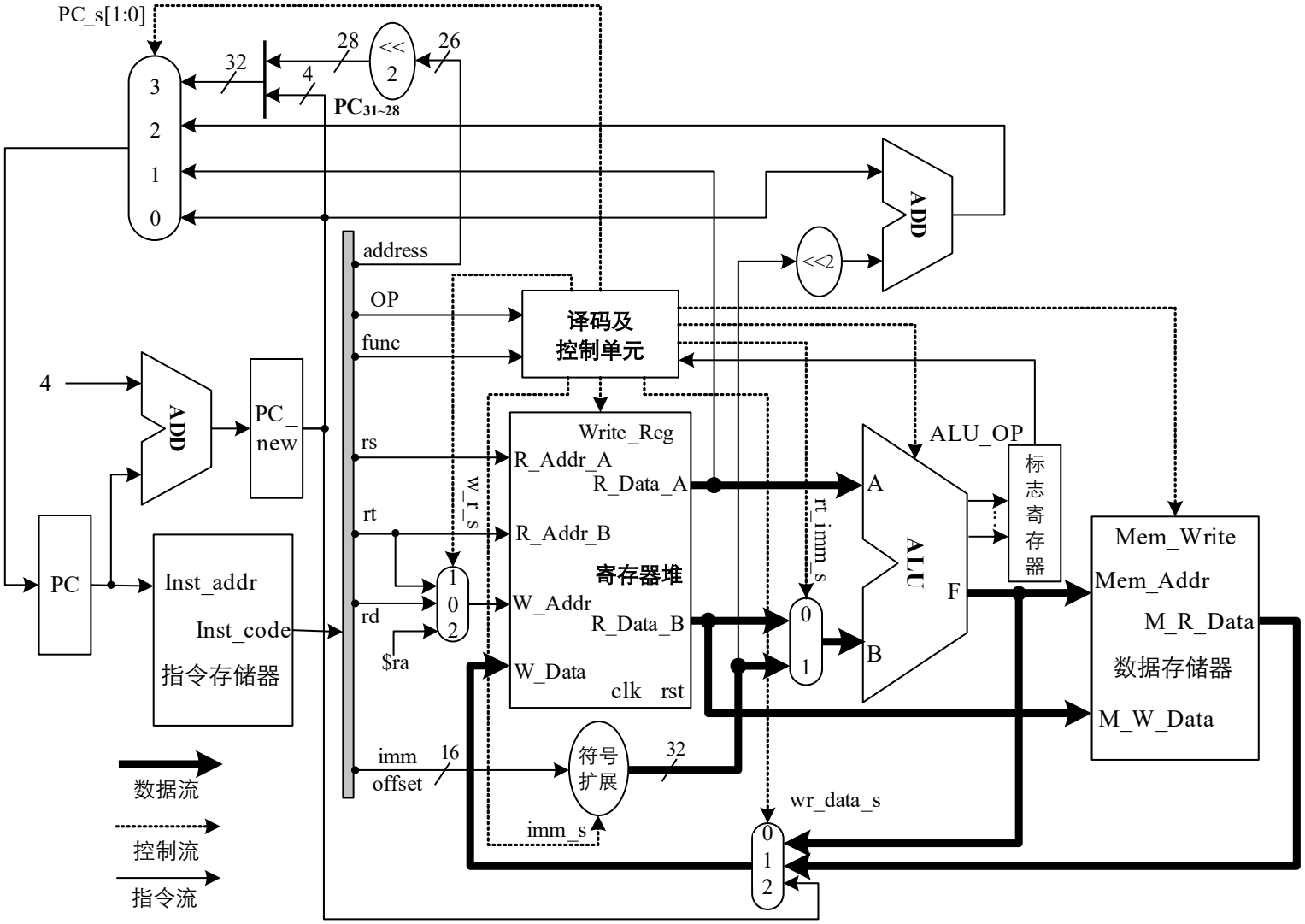


图 7-26 实现 R、I、J 型指令的 CPU 数据通路