实验 9-单周期处理器实验

一、 实验准备

本次实验开始整合之前的实验内容，搭建完整的单周期处理器 datapath。在理论课上，我们讲解了 MIPS 架构 CPU 的传统流程可分为取指、译码、执行、访存、回写 (Instruction Fetch (IF), Register Access and Decode, Execution, Memory Request, Write Back)，五阶段。在前面几次实验中，分别完成了 ALU 设计和寄存器器堆的设计；掌握了存储器 IP 的使用；实现了单周期 CPU 的取指、译码阶段，完成了 PC、控制器的设计。通过前面的几次实验，单周期 CPU 的设计的各模块已经具备，再引入数字逻辑课程中所实现的多路选择器、加法器等门级组件，通过对原理图的理解，分析单条（单类型）指令在数据通路中的执行路径，依次连接对应端口，将各个模块通过搭积木一样组合起来即可完成单周期 CPU。在进行本次实验前，你需要具备以下基础能力：
1.熟悉 Vivado 的仿真功能 (行为仿真)
2.理解数据通路、控制器的信号

二、 实验目的

1. 掌握单周期 CPU 数据通路图的构成、原理及其设计方法；
2. 掌握单周期 CPU 的实现方法，代码实现方法；
3. 认识和掌握指令与 CPU 的关系；
4. 掌握测试单周期 CPU 的方法。

三、 实验设备

PC 机一台，Basys3 开发板，Xilinx Vivado 开发套件。

四、 实验任务

4.1 实验要求

先阅读实验原理部分，实现下列模块，并按照下列要求完成实验：

- Datapath，其中主要包含 alu 和寄存器堆(ALU 和寄存器队实验已完成)，存储器（存储器实验已经完成），PC 和 controller（控制器实验已经完成，其中 Controller 包含两部分，分别为 main_decoder，alu_decoder），adder、mux2、signext、sl2(其中 adder、mux2 数字逻辑课程已实现，相关的源代码模板在前面的实验已经提供，signext、sl2 参见实验原理)。
- 指令存储器 inst_mem(Single Port Rom)，数据存储器 data_mem(Single Port Ram)；使用 BlockMemory Generator IP 构造指令，注意考虑 PC 地址位数统一。 (参考实验存储器实验专题部分)
- 参照实验原理，将上述模块依指令执行顺序连接。实验给出 top 文件，需兼容 top 文件端口设定。
- 实验给出仿真程序，最终以仿真输出结果判断是否成功实现要求指令。仿真具体参照，参照实验 1 基础操作，以及《Basys3 入门指导手册》。
- 上板测试，改写 regfile.v 增加一路读口，然后把新增加的读口地址连接到拨码开关，把新增加的读口接到七段数码管上显示（类似于前面的寄存器堆实验里面的做法，同学们可以参考一下）；验证程序运行后，通过拨码开关选择不同的寄存

器，通过七段数码管上显示查看每个寄存器的值是否符合预期（具体演示，见课程上的讲解）。

4.2 实验步骤

1. 从前面的实验中，导入 alu 模块和寄存器堆模块；
2. 从控制器实验中导入 PC、Controller 模块；
3. 从提供的基础模块中导入多路选择器、加法器模块；
4. 根据存储器实验使用 Block Memory，其中 inst_mem 导入 coe 文件；
5. 参考实验原理，连接各模块；
6. 导入顶层文件及仿真文件，运行仿真；仿真 TestBench 已经提供，请阅读 TestBench 文件，搞清楚仿真测试的含义；
7. 仿真测试通过后，进行上板实验，查看在线的结果；

4.2 实验模块结构

|–top.v	设计顶层文件，已提供。
|—-mips.v	MIPS 软核顶层文件，将 Controller 与 Datapath 连接，已提供
|———controller.v	控制器模块，参照（改写）控制器实验代码。
|———maindec.v	Main decoder 模块，负责译码得到各个组件的控制信号。
|———aludec.v	ALU Decoder 模块，负责译码得到 ALU 控制信号。
|———datapath.v	数据通路模块，自行实现
|———pc.v	PC 模块，参照（改写）控制器实验代码
|———alu.v	ALU 模块，参照（改写）ALU 实验代码
|———sl2.v	移位模块，参考《其他组件实现.pdf》
|———signext.v	有符号扩展模块，参考《其他组件实现.pdf》
|———mux2.v	二选一选择器，自行实现
|———regfile.v	寄存器堆，参照基础模块代码，前面寄存器堆实验部分已经提供（上板实验需要自己添加一路读口）
|———adder.v	加法器，参照基础模块代码，前面寄存器堆实验部分已经提供
|—-inst_ram.ip	RAM IP，通过 Block memory generator 进行实例化，见存储器实验部分
|—-data_ram.ip	RAM IP，通过 Block memory generator 进行实例化，见存储器实验部分

五、 实验原理

5.1 实验总体框架及单周期处理器连线图

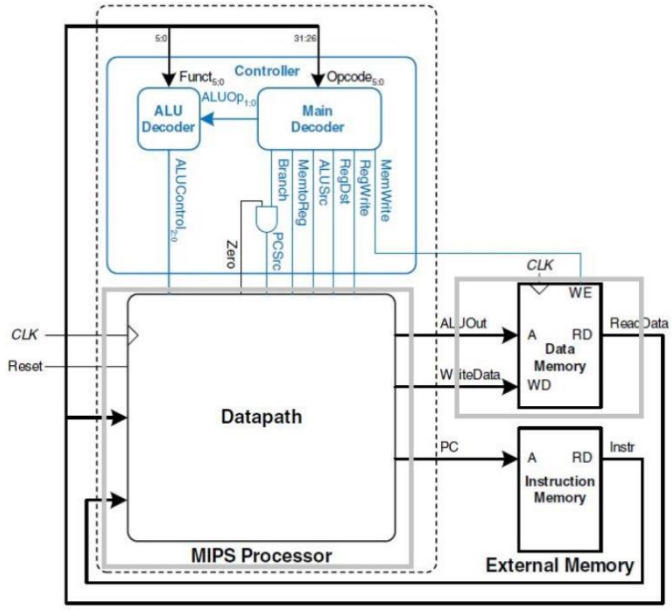

图 1: 单周期 CPU 框架图

如图 1，完整的单周期 CPU 实现框架图。图中有些部分我们之前已经在实验 5,6,7 中已经完成（参见 4.2 中的描述），继续完成 datapath 模块，即可将单周期 MIPS 软核完整实现。

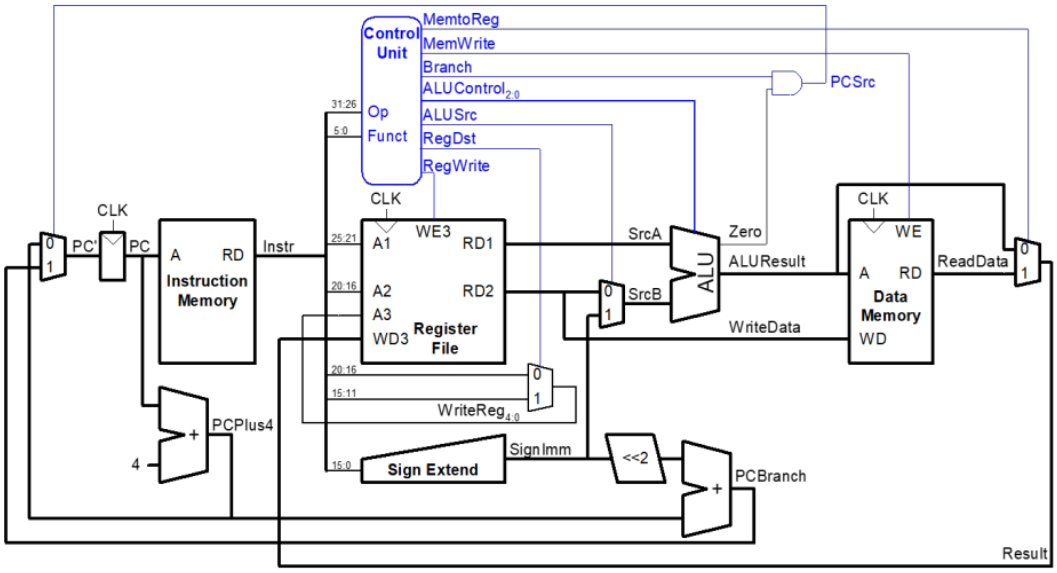

图 2: 单周期 CPU 系统连线图

图 2 为单周期的完整系统连线图，涵盖 *add*、*and*、*sub*、*or*、*slt*、*beq*、*j*、*lw*、*sw*、*addi* 等指令。本次实验仅实现上述几类指令，完整的 MIPS 指令集将与硬件综合设计中完善，此处以掌握数据通路分析为主要目的。

5.2 控制器译码信号规范

1. ALU 控制码译码

由于控制器的设计实验中并不进行强制限制，因此 ALUOp 信号在部分设计中可能不会存在，只需按照下表中其他信号即可。

ALUOp[1:0]	Funct	ALUControl[2:0]
00	X(for lw)	010 (Add)
01	X(for beq)	110 (Subtract)
10	100000 (add)	010 (Add)
10	100010 (sub)	110 (Subtract)
10	100100 (and)	000 (And)
10	100101 (or)	001 (Or)
10	101010 (slt)	111 (SLT)

表 1. ALU 控制码译码信号表

注意：**这里只需要你能保证你 ALUControl 的信号能与你自己设计的 ALU 的控制信号能匹配上就可以了。也可以不按照这个表来设计。另外，需要检查 ALU 功能的完备性，根据功能进行相应的 ALU 模块改写。**

2.信号控制码译码信号

信号控制码译码信号与实验相同，在通路连接时分别接入到需要控制的端口。

5.3 数据通路连接

在进行数据通路连接前，除了三大基本器件，还有些许小器件需要实现，包括加法器、触发器、多路选择器、移位器、符号扩展器件等。这些器件多为简单的组合逻辑，在基本模块实现那里（本实验也会提供一些部分模块实现），已经提供基础的相关实现，同学们可以参照。

1. LW 和 SW 指令

以 LW 指令为例构建基本的单周期 CPU 数据通路，需要的基本器件有 PC、Regfile、存储器，见图 3，其他小的组合逻辑部件根据需求进行添加。

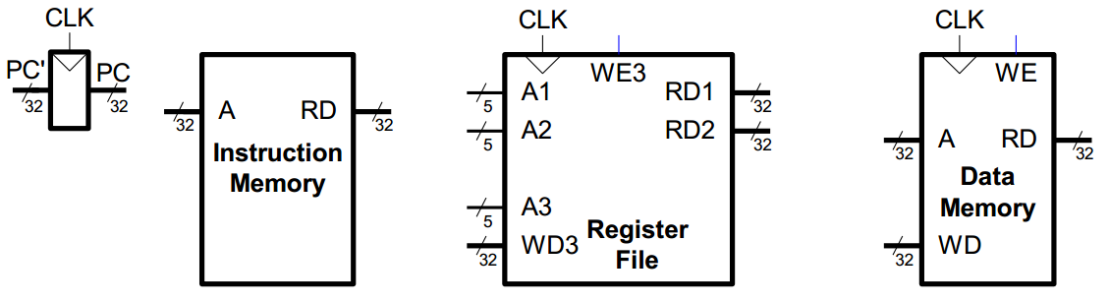

图 3. LW 所需要的模块列表

第一步为取值，将 PC(即指令地址) 输出到 Instruction Memory(指令存储器) 的 address 端口。LW 指令的汇编格式为：LW rt, offset(base)，其中 base(基地址) 为指令[25:21] 所指向的寄存器值，offset(地址偏移)为指令[15:0]。将 base 寄存器的值加上符号扩展后的立即数 offset 得到访存的地址，根据地址从存储器中读取 1 个字 (连续 4 个字节) 的值写入到 rt 寄存器中。根据 LW 指令的定义，将指令存储器读出的指令([31:0]) 中 [25:21] 连接至 regfile 寄存器堆的第一个输入地址，即 Address1(A1)。过程如图 4 所示。

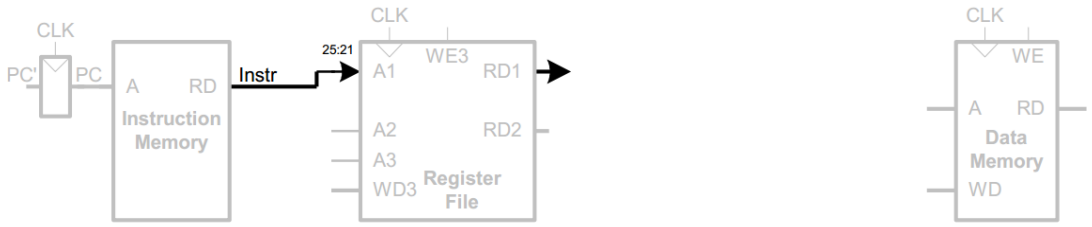

图 4. 取指令和译码过程

除此之外需要将 offset 进行扩展，因而将指令[15:0]传至有符号扩展模块，输出 32 位的符号扩展信号 (SignImm)。得到基地址 base 和 offset 后，需进行加法计算，其操作可以采用如下描述：base+ sign_extend(offset)。加法计算使用 ALU 中设计实现的加法运算，因而图 5 中，RD1 读出 base 和经过有符号扩展后的 sign_extend(offset) 分别作为 ALU 的两个输入 (SrcA、SrcB)，经过 ALU 进行加法运算后，得到 ALUResult 作为地址，输入到数据存储器 Data Memory 的 Address 端口。这里需要注意的是，由于计算地址与进行加法运算相同，所以用于控制 ALU 运算类型的信号 ALUControl 与加法运算应该相同，如图 5 中蓝色部分所示。AlUControl 信号由 Controller 译码得到，在此不再赘述。

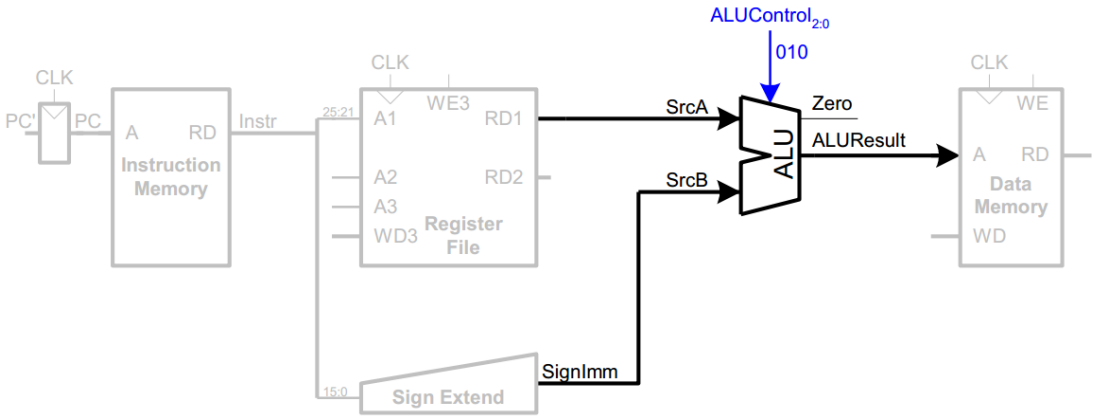

图 5. ALU 计算过程

将地址输入后，将数据存储器读出的数据写回到 regfile 中，其地址为 rt，即指令的[20:16]。如图 6，连接时需要将指令[20:16]连接至寄存器堆的 Address3(A3)端口，对应的数据信号 ReadDat 连接至 WriteData3(WD3)端口。

注意：LW 指令回写目标寄存器是使用的 rt，而这里 R type 指令使用的 rd，所以这里后面会添加一个 mux。

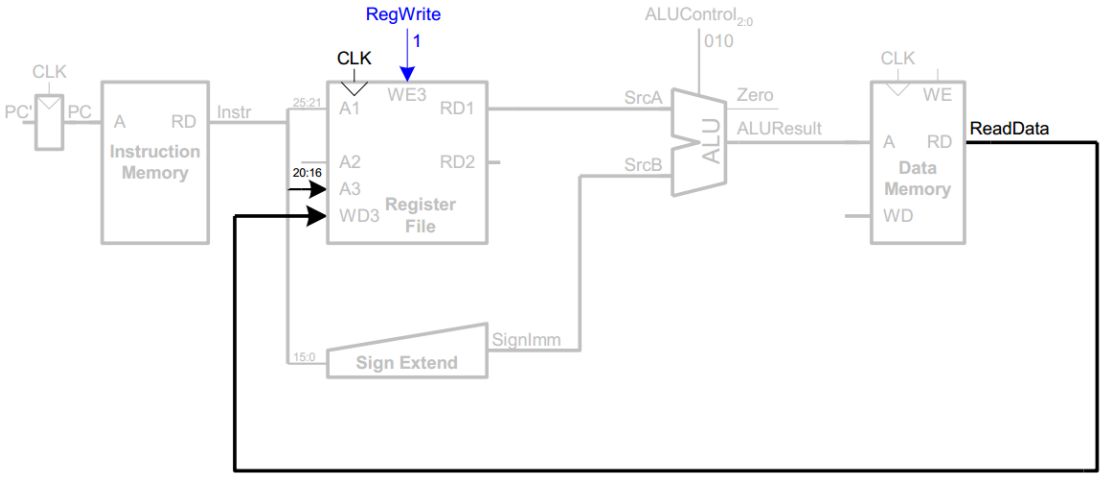

图 6. 寄存器堆写回操作

完成上述连接后，一条能够满足 LW 指令执行的数据通路即完成。 LW 指令执行结束后，需开始下一条指令的执行，重复同样的执行过程。唯一的不同在于 PC 地址需要变为下一条指令所在地址。由于实现的是 32 位 MIPS 指令集，并且采用字节读写的方式，因为需要读取后续 4 个字节的数据，即 PC+4。将得到的 PC+4 信号写入 PC(D 触发器)的输入端，即可实现每周期地址+4 的操作。

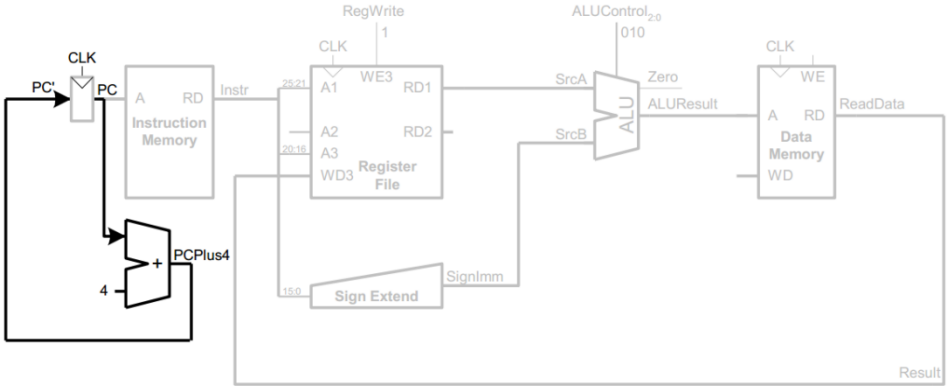

图 7. PC 更新

注意：电路设计简单起见，这里我们使用一个单独的加法器模块来作 PC 更新，如图 7 所示。

sw 指令前三个过程和 LW 指令一样，到了第四个过程变为写存储器，然后结束。计算地址的 base+sign_extend(offset) 不变，但 rt 变为读取寄存器，因而需要将其连接至 Address2(A2)，读出的数据 RD2 信号连接至数据存储器的 WD 端口。

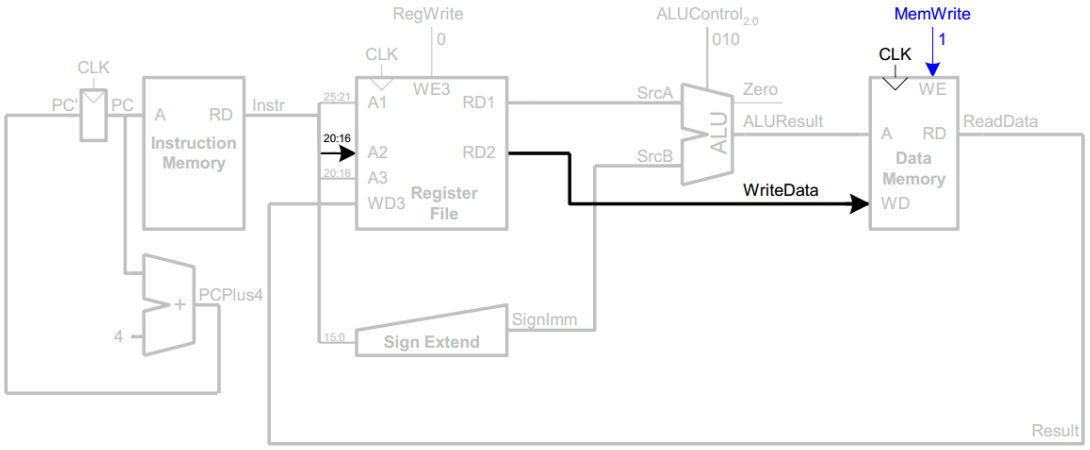

图 7. PC 更新

注意：LW/SW 指令回写目标寄存器是使用的 rt，而这里 R type 指令使用的 rd，所以这里后面会添加一个 mux。

2. R-type 指令

R-type 指令和 LW 指令不同的地方在于，R type 指令使用目标寄存器是 rd，而不是 LW 使用的 rt，写回寄存器堆的数据是从 ALU 过来，而不是 LW 操作中是从存储器过来。因此，通路改造方法为添加多路选择器。lw 指令写入 regfile 的地址为 rt，而 R-Type 为 rd，在此处加入一个多路选择器，输入分别为指令的[20:16]和[15:11]，使用 RegDst(register destination) 信号控制，多路选择其输出信号命名为 WriteReg(write register)。ALU 输入为 rs,rt，分别对应 srcA,srcB，因此需要在 srcB 处加入多路选择器，选择来源为 RD2 或立即数 SignImm，控制信号为 ALUSrc(ALU source)。最后写回到 regfile 的值应该为 ALU 计算得到的值，为 ALUResult，加入多路选择器控制 Result 来源为 ALU 或数据存储器，控制信号为 MemtoReg(Memory to regfile)。

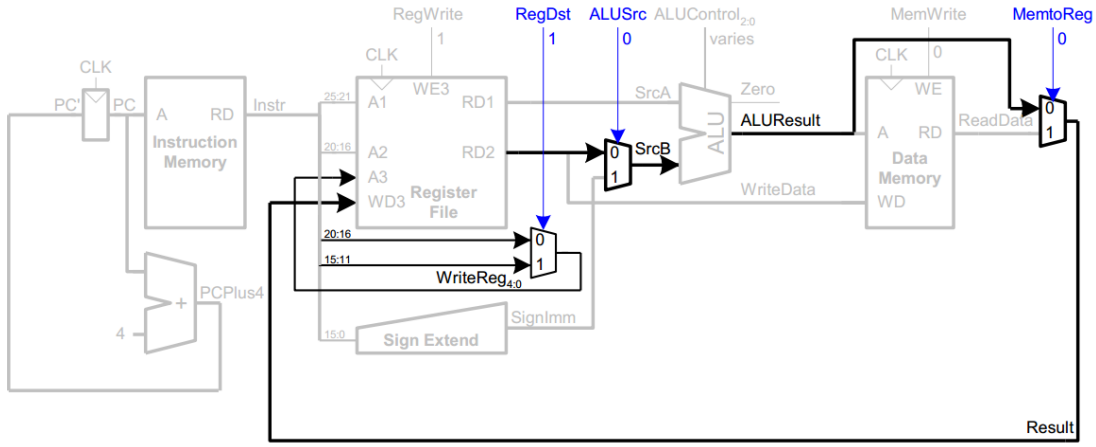

图 8. R-type 数据通路

3. Branch 指令

beq 需要三步操作，分别为条件判断，偏移计算，PC 转移，下面分别实现每步操作。条件判断需要判断 rs、rt 所在的寄存器值是否相等，可以使用 ALU 的减法进行计算，输

出 zero 信号，并与译码得到的 Branch 信号 (判断是否为分支指令) 进行 and 操作，作为 PCSrc 信号。第二步偏移计算公式为：PC+4+ sign_extend(offset)*4。先将 offset 进行有符号扩展，再左移 2 位后，再最后与 PC+4 相加。最后 PC 转移根据 PCSrc 信号进行控制，满足条件时，PCSrc 为 1，选择 PCBranch 作为下一条指令的地址。通路图修改见图 9.

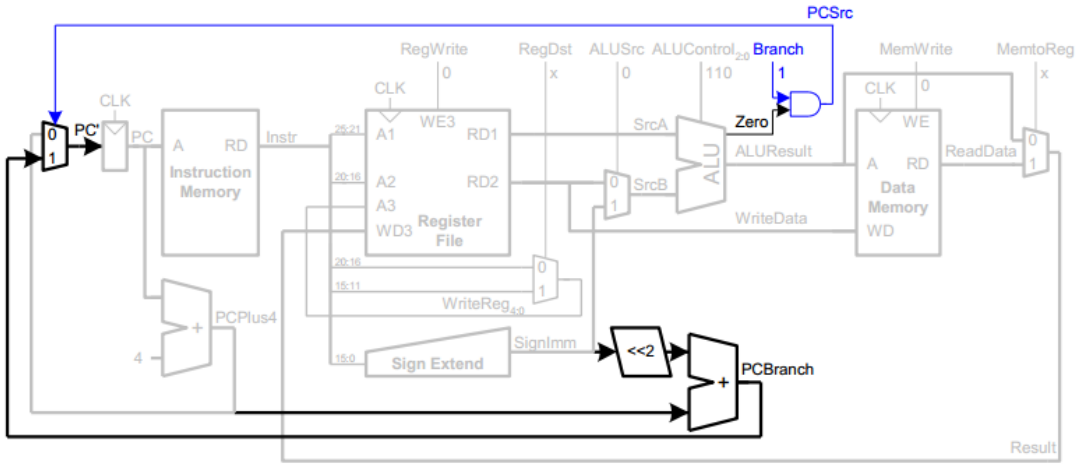

图 9. Branch 数据通路

4. J 指令

跳转指令实际为无条件跳转，不需要做任何判断，因此更加简单。只需要计算地址，更改 PC 即可。其跳转目标由该指令对应的延迟槽指令的 PC(PC+4) 的最高 4 位与立即数 instr_index(instr[25:0]) 左移 2 位后的值拼接得到。如图 10, instr[25:0] 左移 2 位，拼接 pc[31:28]，而后通过多路选择器。多路选择器直接由 jump 进行控制。

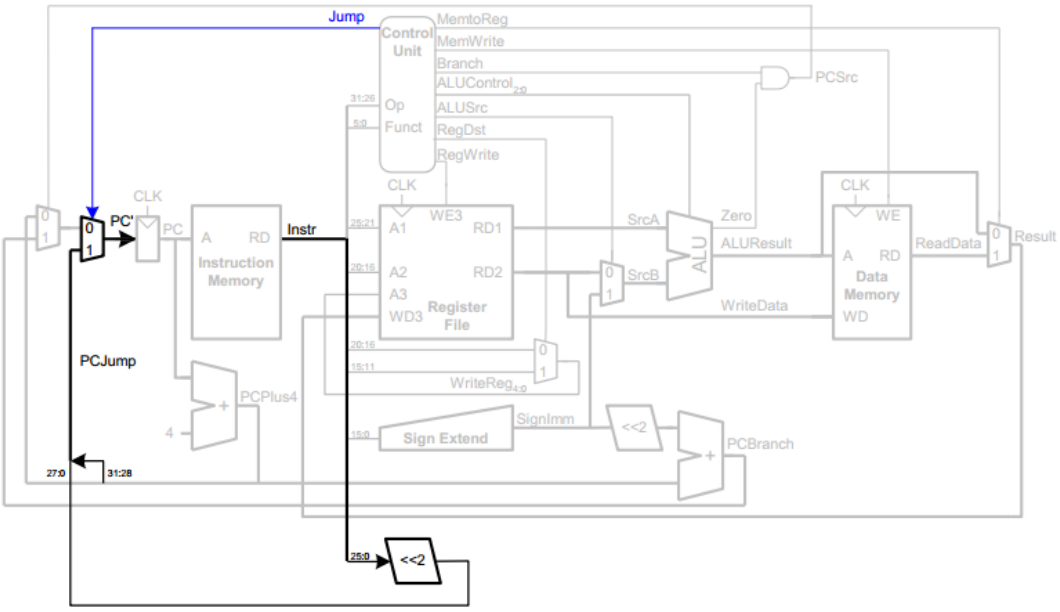

图 10. Branch 数据通路

六、 实验报告要求

1. 按照实验要求 4.1 搭建如图 1 所示实验系统，按照 4.2 要求实现所要求的模块的内容。
2. 在指令存储器中存入下列程序，并通过仿真验证每一类指令的结果，附上仿真波形图，说

明其正确性。并需要通过仿真结果，说明所设计的处理器是否能够跑完所有代码，在 end 处执行了正确的代码。

#	Assembly	Description	Address	Machine	
main:	addi $2, $0, 5	# initialize $2 = 5	0	20020005	20020005
	addi $3, $0, 12	# initialize $3 = 12	4	2003000c	2003000c
	addi $7, $3, -9	# initialize $7 = 3	8	2067fff7	2067fff7
	or $4, $7, $2	# $4 <= 3 or 5 = 7	c	00e22025	00e22025
	and $5, $3, $4	# $5 <= 12 and 7 = 4	10	00642824	00642824
	add $5, $5, $4	# $5 = 4 + 7 = 11	14	00a42820	00a42820
	beq $5, $7, end	# shouldn't be taken	18	10a7000a	10a7000a
	slt $4, $3, $4	# $4 = 12 < 7 = 0	1c	0064202a	0064202a
	beq $4, $0, around	# should be taken	20	10800001	10800001
	addi $5, $0, 0	# shouldn't happen	24	20050000	20050000
around:	slt $4, $7, $2	# $4 = 3 < 5 = 1	28	00e2202a	00e2202a
	add $7, $4, $5	# $7 = 1 + 11 = 12	2c	00853820	00853820
	sub $7, $7, $2	# $7 = 12 - 5 = 7	30	00e23822	00e23822
	sw $7, 68($3)	# [80] = 7	34	ac670044	ac670044
	lw $2, 80($0)	# $2 = [80] = 7	38	8c020050	8c020050
	j end	# should be taken	3c	08000011	08000011
	addi $2, $0, 1	# shouldn't happen	40	20020001	20020001
end:	sw $2, 84($0)	# write adr 84 = 7	44	ac020054	ac020054

3. 仿真通过后，将代码进行实际上板验证，验证并观察单周期 CPU 真实运行的结果，并分析结果的正确性；验收的时候，解释其运行结果的正确性（疫情如无法线下验收，可以排一个视频，自己演示并解释一下实验结果，发到邮箱）；
4. 在实验报告中，详细阐述实验设计过程，代码模块结构，以及观察到实验结果以及验证原理（根据仿真波形说明你设计的处理器为什么是正确的，能运行的），并分析实验结果。