版本

更新记录	文档名	实验指导书 _lab3
	版本号	0.3
	创建人	计算机组成原理教学组
	创建日期	2017/11/3

更新历史

序号	更新日期	更新人	版本号	更新内容
1	2017/11/5	吕昱峰	0.1	初版，单周期 CPU 取指译码实验
2	2018/11/5	吕昱峰	0.2	重新组织指导书内容。
3	2019/11/21	吕昱峰	0.3	完善设计连线的描述内容，阅读更加易于理解。
4	2024/4/19	陈昭睿	0.4	将目标指令集架构更改为RISCV

文档错误反馈:20194238@cqu.edu.cn

1 实验三 简易单周期 CPU 实验

RISCV 架构 CPU 的传统流程可分为取指、译码、执行、访存、回写 (Instruction Fetch, Decode, Execution, Memory Request, Write Back)，五阶段。

实验一完成了ALU 设计并掌握了存储器IP 的使用；实验二实现了单周期 CPU 的取指、译码阶段，完成了 PC、控制器的设计。在实验一与实验二的基础上，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，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

1.1 实验目的

1. 掌握不同类型指令在数据通路中的执行路径。
2. 掌握 Vivado 仿真方式。

1.2 实验设备

1. 计算机 1 台 (尽可能达到 8G 及以上内存)；
2. Xilinx Vivado 开发套件 (2019.1 版本)。

注：本次实验为 CPU 软核实验，不涉及开发板外围设备，故不需要开发板

1.3 实验任务

1.3.1 实验要求

阅读实验原理实现以下模块：

(1) Datapath，其中主要包含 alu(实验一已完成)，PC(实验二已完成)，adder、mux2、signext、sl2(其中 adder、mux2 数字逻辑课程已实现，signext、sl2 参见实验原理)，
(2) Controller(实验二已完成)，其中包含两部分，分别为 main_controller，alu_controller。
(3) 指令存储器 inst_mem(Single Port Ram)，数据存储器 data_mem(Single Port Ram)；使用 Block Memory Generator IP 构造指令，注意考虑 PC 地址位数统一。(参考实验二)
(4) 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。
(5) 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。

1.3.2 实验步骤

1. 从实验一中，导入 alu 模块；
2. 从实验二中导入 PC、Controller 模块；
3. 从数字逻辑实验中导入多路选择器、加法器模块；
4. 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；

1.4 实验环境

|–top.v	设计顶层文件，已提供。
|—-RISCV.v	RISCV 软核顶层文件，将Controller 与 Datapath 连接
|——controller.v	控制器模块，本次实验重点。
|—maincontrol.v	Main decoder 模块，负责译码得到各个组件的控制信号。
|——alucontrol.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，使用实验二代码
|———alu.v	ALU 模块，使用实验一代码
|———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，已提供
|———adder.v	加法器，已提供
|—-*inst_ram.ip*	*RAM IP*，通过 *Block memory generator* 进行实例化
|—-*data_ram.ip*	*RAM IP*，通过 *Block memory generator* 进行实例化

表 1: 实验文件树（仅作为参考，不是必须使用提供的文件）

2 实验原理

2.1 总体框架及通路图

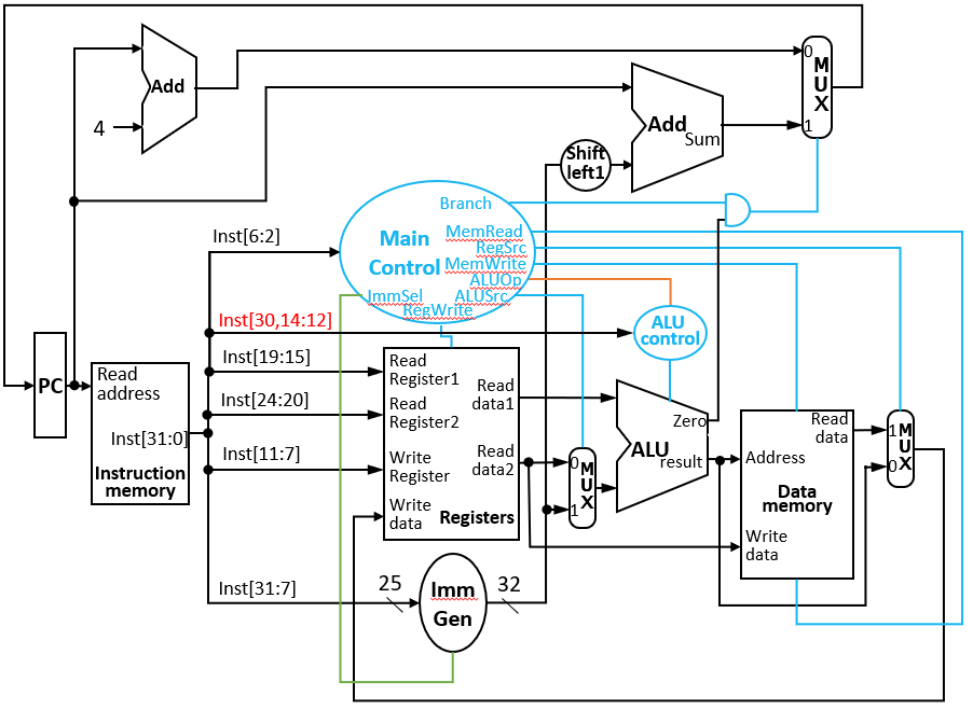

图 1: 单周期 CPU 框架图

图 1为单周期的完整通路图。实验三仅实现先前实验二所要求支持的指令，完整的 RISCV 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。

2.2 控制器译码信号规范

2.2.1 ALU 控制码译码

由于控制器的设计在实验二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，只需按照表 2中其他信号即可

opcode	ALUOp	Operation	Function field	ALU function	ALU control
ld	00	load register	xxxxxxxxxx	ADD	0001
sd	00	store register	xxxxxxxxxx	ADD	0001
branch	01	BEQ	xxxxxxx000	SUB	0010
arith.	10	ADD[I]	0000000000	ADD	0001
		SUB	0100000000	SUB	0010
		AND[I]	0000000111	AND	0011
		OR[I]	0000000110	OR	0100
		XOR[I]	0000000100	XOR	0101
		SLL[I]	0000000001	SLL	0110
		SRL[I]	0000000101	SRL	0111
		SRA[I]	0100000101	SRA	1000
		SLT[I]	0000000010	SLT	1001

表 2: ALU 控制码译码信号表

这里需要注意，不按照表中的信号也可，只要保证不同运算的 alucontrol 信号不同即可（建议按照表中的控制信号）。

2.2.2 信号控制码译码信号

信号控制码译码信号与实验二相同，在通路连接时分别接入到需要控制的端口。

op	Inst[6:0]	Branch	MemRead	RegSrc	MemWrite	ALUOp	ALUSrc	RegWr.	ImmSel
add	7'h33	1'b0	1'b0	1'b0	1'b0	2'b10	1'b0	1'b1	*
addi	7'h13	1'b0	1'b0	1'b0	1'b0	2'b10	1'b1	1'b1	I
sub	7'h33	1'b0	1'b0	1'b0	1'b0	2'b10	1'b0	1'b1	*
and	7'h33	1'b0	1'b0	1'b0	1'b0	2'b10	1'b0	1'b1	*
andi	7'h13	1'b0	1'b0	1'b0	1'b0	2'b10	1'b1	1'b1	I
or	7'h33	1'b0	1'b0	1'b0	1'b0	2'b10	1'b0	1'b1	*
ori	7'h13	1'b0	1'b0	1'b0	1'b0	2'b10	1'b1	1'b1	I
xor	7'h33	1'b0	1'b0	1'b0	1'b0	2'b10	1'b0	1'b1	*
xori	7'h13	1'b0	1'b0	1'b0	1'b0	2'b10	1'b1	1'b1	I
sll	7'h33	1'b0	1'b0	1'b0	1'b0	2'b10	1'b0	1'b1	*
slli	7'h13	1'b0	1'b0	1'b0	1'b0	2'b10	1'b1	1'b1	I
srl	7'h33	1'b0	1'b0	1'b0	1'b0	2'b10	1'b0	1'b1	*
srli	7'h13	1'b0	1'b0	1'b0	1'b0	2'b10	1'b1	1'b1	I
sra	7'h33	1'b0	1'b0	1'b0	1'b0	2'b10	1'b0	1'b1	*
srai	7'h13	1'b0	1'b0	1'b0	1'b0	2'b10	1'b1	1'b1	I
slt	7'h33	1'b0	1'b0	1'b0	1'b0	2'b10	1'b0	1'b1	*
slti	7'h13	1'b0	1'b0	1'b0	1'b0	2'b10	1'b1	1'b1	I
LW	7'h03	1'b0	1'b1	'b1	1'b0	2'b00	1'b1	1'b1	I
SW	7'h23	1'b0	1'b0	*	1'b1	2'b00	1'b1	1'b0	S
BEQ	7'h63	1'b1	1'b0	*	1'b0	2'b01	1'b0	1'b0	B

表 3: 控制信号译码

2.3 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，在数字逻辑课程已有涉及，此处不再赘述，具体实现参见附录。

2.3.1 LW 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。

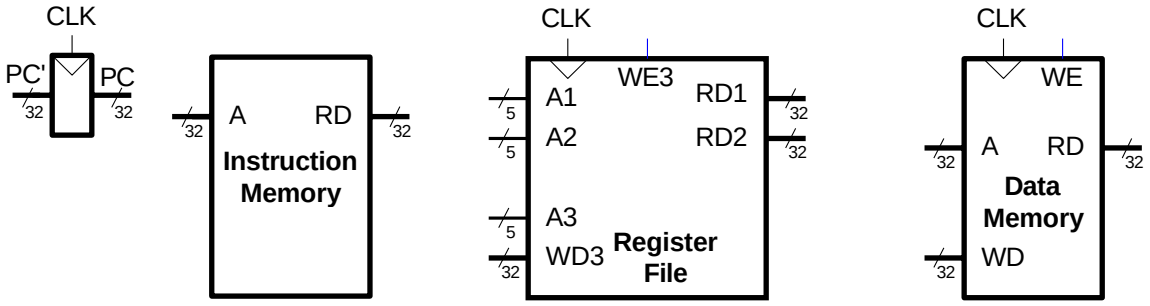

图 3

第一步为取值，将 PC(即指令地址) 输出至 Instruction Memory(指令存储器) 的 address 端口，如图 4。

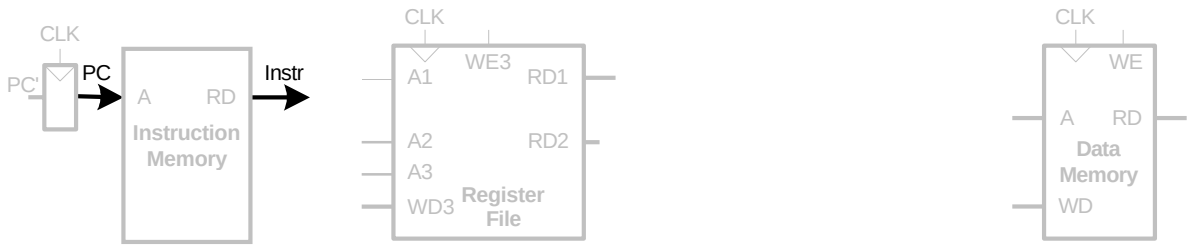

图 4

LW 指令的汇编格式为：lw rd, rs1, imm，其中 rs1为指令 [19:15] 所指向的寄存器值，imm为指令 [31:20]。将rs1寄存器的值加上符号扩展后的立即数imm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 rd 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 [19:15] 连接至 regfile 寄存器堆的第一个输入地址，即Address1(R1)。

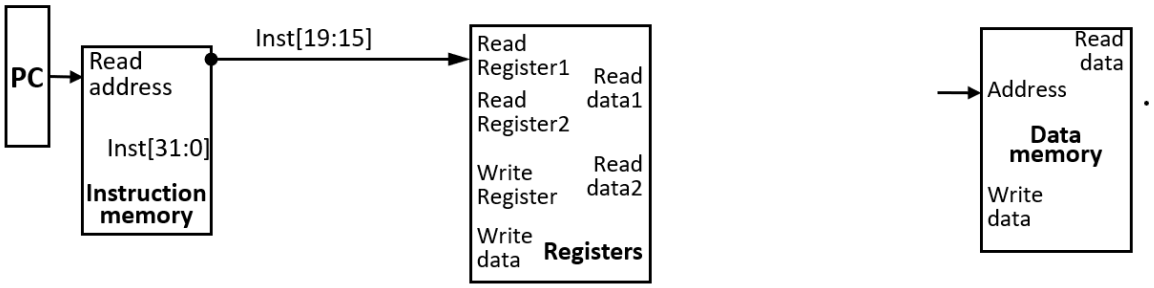

图 5

除此之外需要将 offset 进行扩展，因而将指令 [31:20] 传至有符号扩展模块，输出 32 位的符号扩展信号 (SignImm)。

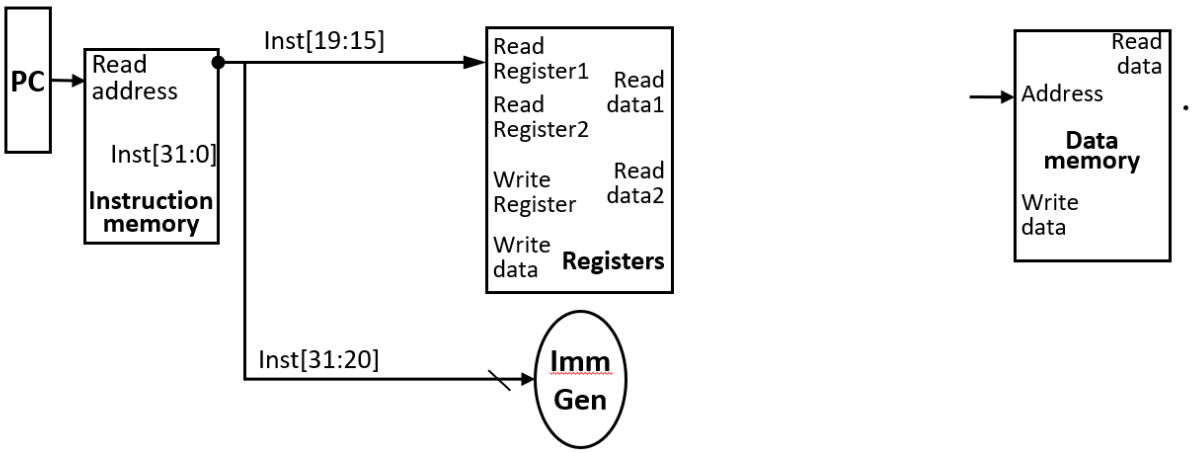

图 6

得到基地址和偏移量后，需进行加法计算。加法计算使用 ALU 中设计实现的加法运算，因而图 7中，RD1 读出的 GPR[rs1] 和经过有符号扩展后的 sign_extend(imm) 分别作为 ALU 的两个输入 (SrcA、SrcB)，经过 ALU 进行加法运算后，得到 result 作为地址，输入到数据存储器 Data Memory 的 Address端口。

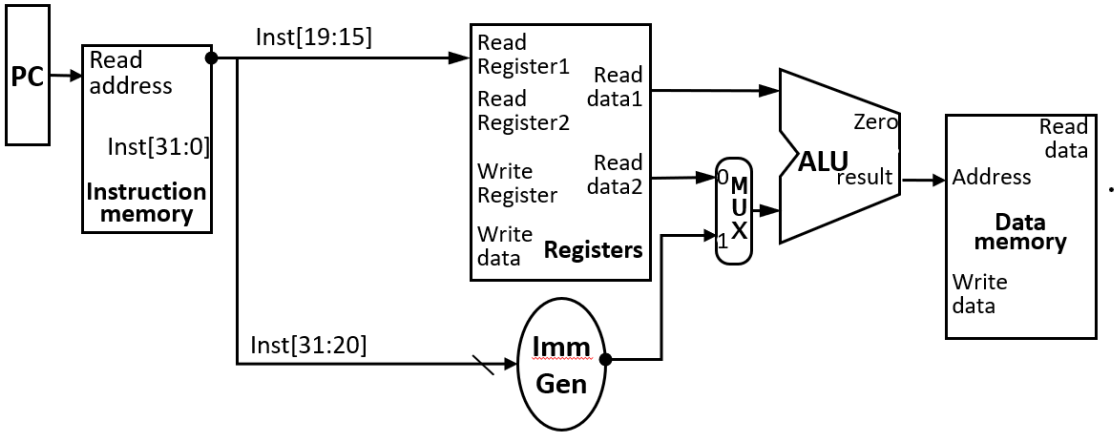

图 7

将地址输入后，将数据存储器读出的数据写回到 Register file 中，其地址为 rd，即指令的 [11:7]。如图 8，连接时需要将指令 [11:7] 连接至寄存器堆的 Write data 端口，对应的数据信号 Read data连接至 Write data 端口。

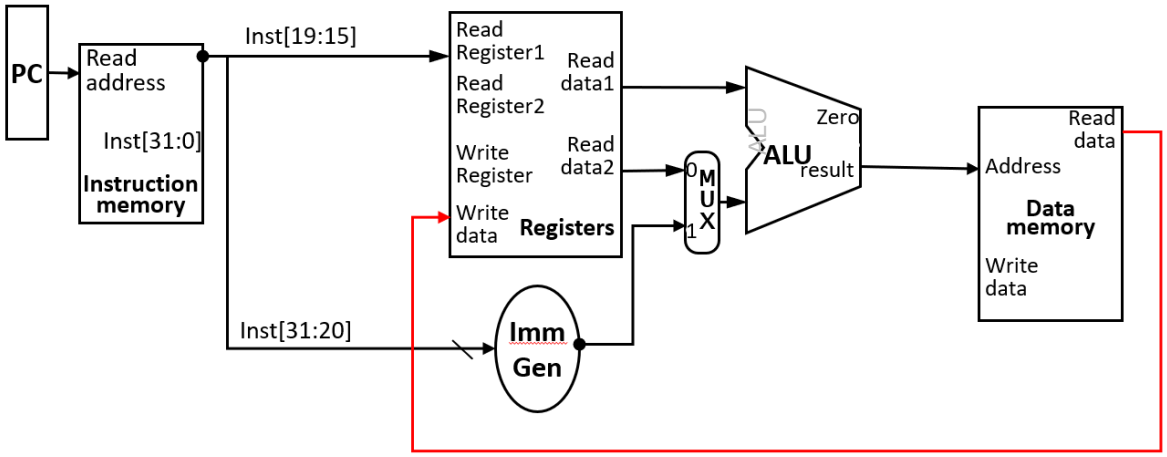

图 8

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 RISCV 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。

注意：这里我们不采用很多参考书籍中复用 ALU 的方式计算加法，而是使用一个单独的加法器模块，如图 9，原因在于，ALU 在实现多种运算逻辑后，其组合逻辑更加复杂，复用 ALU 进行 PC+4 的运算需要控制信号、输入的多路选择，增加了设计的复杂性的同时，也不符合硬件电路设计的思维。在电路设计中的复用更多是出于简化设计、优化延迟、降低功耗等目的，而非单一功能完成后进行复用的软件设计思维。

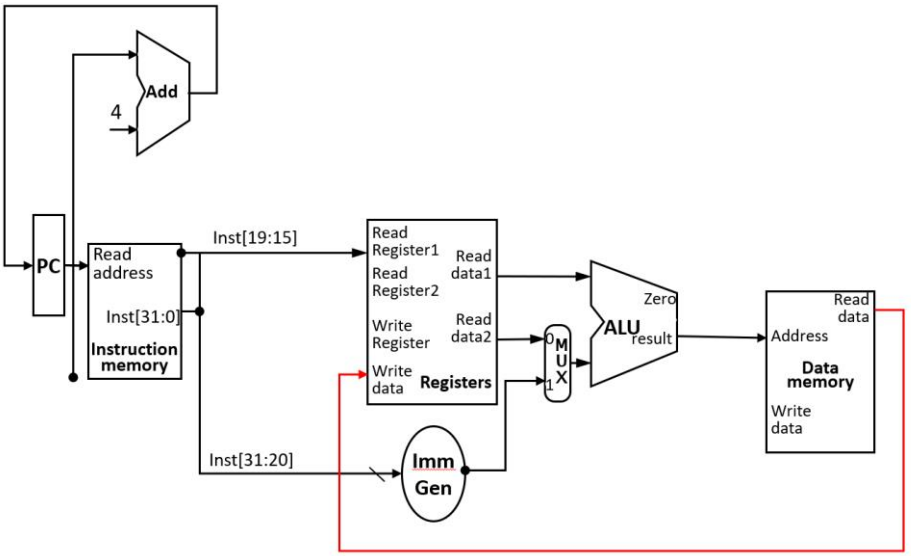

图 9

2.3.2 BEQ指令

BEQ指令定义为：BEQ rs1, rs2, imm。如果寄存器rs1 的值等于寄存器rs2 的值则转移，否则顺序执行。转移目标由立即数imm进行有符号扩展并左移 1 位(RISCV32中pc偏移量是半字对齐)的值加上该分支指令对应的PC计算得到。

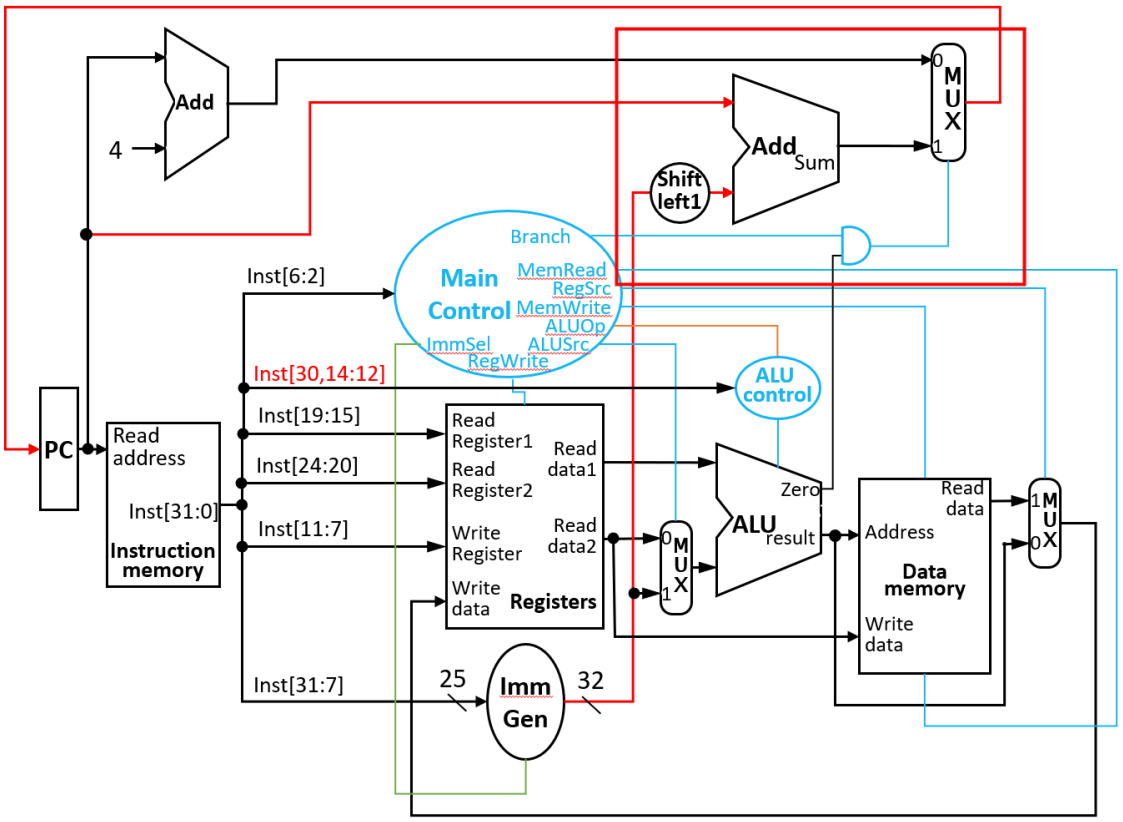

图 12

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 rs1、rs2 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。

先将imm进行有符号扩展，再左移1位，最后与PC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 12。

A 指令集手册及查找表

见《riscv-card》和《RISCVGreenCard》。