华中科技大学

2015

计算机组成原理　　课程设计报告

题　　目：	5 段流水 CPU 设计
专　　业：	信息安全
班　　级：	IS1303
学　　号：	U201315179
姓　　名：	龙登余
电　　话：	18062154863
邮　　件：	2811469022@qq.com
完成日期：	2015-12-13
指导教师：	刘景宁 吴非 谭志虎 孙百勇

计算机科学与技术学院

课程设计任务书

一、设计题目

基于 Logisim 软件仿真平台的 5 段流水 CPU 设计实现

二、设计内容

设计模型机系统的总体结构、指令系统和时序信号。在对该模型机系统中的部件功能利用 EDA 软件的仿真功能进行仿真分析和功能验证的基础上，将部分电路下载到 FPGA，并与适当的外围器件相配合，实现模型机的整机系统。要求所设计的整机系统能支持自动和单步运行方式，能正确地执行存放在主存中的程序的功能，对主要的数据流和控制流通过 LED 适时显示信息。

三、 设计要求

1) 支持 20 条基本指令，具体见表 1 和表 2；
2) 支持 5 段流水机制，可处理数据冒险，结构冒险，分支冒险；
3) 能运行由自己所设计的指令系统构成的一段测试程序，测试程序应能涵盖所有指令，程序执行功能正确。
4) 能运行教师提供的标准测试程序，并统计执行时间。

四、设计流程

1) 根据课程设计指导书的要求，制定出设计方案；
2) 画出自己所设计计算机系统的原理框图，分析所需要的控制信号以及这些控制信号的有效形式；
3) 画出各指令的指令周期流程图和所需要的控制信号；
4) 设计出实现指令功能的控制器；
5) 调试、数据分析、验收检查；
6) 课程设计报告和总结。

五、成绩评定

成绩评定根据考勤、课程设计的过程、课程设计的效果、课程设计报告质量等进行综合评定；其中设计过程和结果占 70%，课程设计报告占 30%；课程设计的成绩评定等级为不及格、及格、中、良好、优秀五级；对基本功能进行扩展或设计具有非常鲜明的特征和一定程度的创新，可根据实际情况加分。

六、设计报告要求

课程设计报告主要内容包括：设计题目、设计目的、设备器材、设计原理及内容、设计步骤、遇到的问题及解决方法、设计总结、参考文献等。要求在适当位置配合相应的实验原理图、数据通路图、实验接线图等图表进行说明。总结部分主要写设计工作简介以及设计体会。应做到文理通顺，内容正确完整，书写工整，装订整齐。课程设计报告采用《计算机组织与结构》专用设计报告模板，A4 纸双面打印。

七、时间安排

课程设计的总体时间为 2 周，具体安排如下:

1) 第 1 天：到实验室布置任务和集中讲解。
2) 第 1~3 天：学生查阅资料，开始方案设计。
3) 第 4 天：中期进度检查，单周期 CPU 验收检查。
4) 第 6 天：中期进度检查，理想流水线多周期 CPU 验收检查。
5) 第 10 天：最终结果验收。

八、主要参考文献

[1] DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第 4 版).北京：机械工业出版社.

[2] David Money Harris(美).数字设计和计算机体系结构（第二版）. 机械工业出版社

[3] 秦磊华，吴非，莫正坤.计算机组成原理. 北京：清华大学出版社，2011 年.

[4] 袁春风编著. 计算机组成与系统结构. 北京：清华大学出版社，2011 年.

[5] 张晨曦，王志英. 计算机系统结构. 高等教育出版社，2008 年.

目 录

1 课程设计概述

1.1 课设目的

计算机组成原理是计算机专业的核心专业基础课。课程设计属于设计型实验，不仅锻炼学生简单计算机系统的设计能力，而且通过进行设计及实现，进一步提高分析和解决问题的能力。同时，设计过程允许同学之间的交流与讨论，增强学生的表达能力以及理解能力。(

1.2 设计任务

计算机系统设计的总体目标是设计模型机系统的总体结构、指令系统和时序信号。所设计的系统能够支持指令系统定义的全部 20 条指令。

具体设计任务如下：完成单周期 CPU 的设计，完成理想流水 CPU 的设计，完成流水线的插入气泡的功能，完成流水线数据重定向的功能，完成分支预测的功能。

1.3 设计要求

根据理论课程所学的知识，设计出简单计算机系统的总体方案，结合各单元实验积累和课堂上所学知识，选择适当芯片，设计简单的计算机系统，具体要求如下：

1) 根据课设指导书的要求，制定设计方案。
2) 画出自己所涉及计算机系统的原理图和器件连接头。

2 实验原理与环境

2.1 实验原理

计算机组成原理，数字逻辑， logisim

2.2 实验环境

描述实验台配置，计算机系统，设计开发工具等，可以用表格与文字相结合的形式，具体参照下面的表格，注意表格的交叉引用以及题注方式。具体测试环境见表 2.1。

表 2.1 光纤通道启动器与阵列控制器配置

配 置	启动器	阵列控制器
CPU	Xeon 2.0	Intel 80321
Memory	512MB	1GB
Fiber Channel HBA	Qlogic 23102G	Agilent DX2
OS	Windows 2003	ARM Linux

3 总体方案设计

3.1构建单周期 CPU

3.1.1 总体设计

本次我采用的方案是专用通路，硬布线控制，且程序和数据分开。同时在实施方案的过程中，除 logisim 提供的器件外，电路中需要的功能部件全部由自己连接并封装成子电路。

需要实现的 CPU 的指令系统的指令表如下表所示。

指令	15~12	11~10	9~8	7~6	5~3	2~0	指令	指令功能
or	0	rs	rt	rd	0	0	or	$rd = $rs | $rt（5~3 位无用）
and	0	rs	rt	rd	0	1	and	$rd = $rs & $rt
add	0	rs	rt	rd	0	2	add	$rd = $rs + $rt
sub	0	rs	rt	rd	0	3	sub	$rd = $rs - $rt
sllv	0	rs	rt	rd	0	4	sllv	$rd = $rs << $rt
srlv	0	rs	rt	rd	0	5	srlv	$rd = $rs >> $rt
srav	0	rs	rt	rd	0	6	srav	$rd = $rs >> $rt 算术右移
slt	0	rs	rt	rd	0	7	slt	$rd = ($rs < $rt) ? 1 :0
beq	1	rs	rt	offset-s			beq	beq =?
bne	2	rs	rt	offset-s			bne	bne != ?
bgt	3	rs	rt	offset-s			bgt	bgt >?（有符号比较）
DISP	4	rs	rt	immediate-u			DISP	DISP[imm] = $rs
lui	5	0	rt	immediate-u			lui	$rt = imm << 8
ori	6	rs	rt	immediate-u			ori	$rt = $rs | imm
andi	7	rs	rt	immediate-u			andi	$rt = $rs & imm
addi	8	rs	rt	immediate-s			addi	$rt = $rs + imm
lw	9	rs	rt	immediate-s			lw	$rt = MEM[$rs + imm]
sw	10	rs	rt	immediate-s			sw	MEM[$rs+imm] = $rt
jump	11	jump address					jump	jump
halt	12	0					halt	halt (时钟暂停)

表格 3. 1 指令系统表

总体结构图如图 3.1 所示。

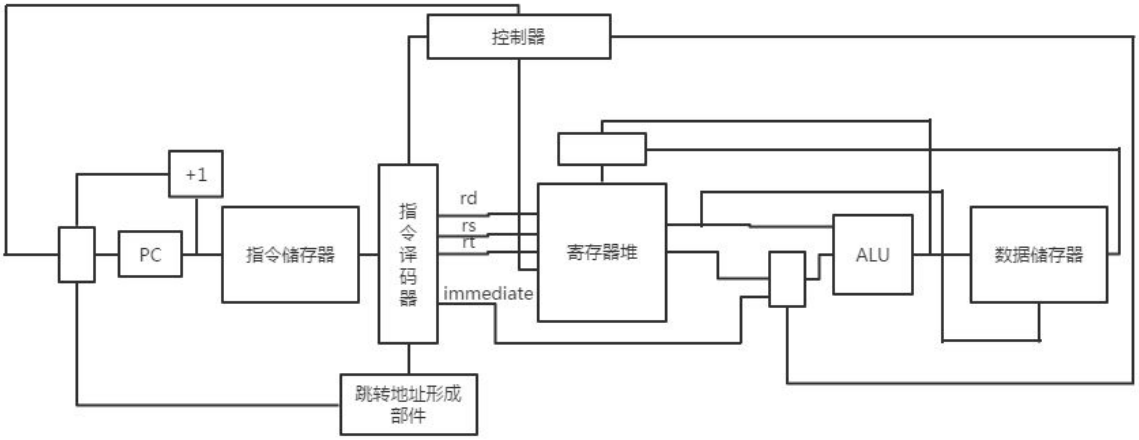


图 3. 1 总体结构框架

3.1.2 主要功能部件

运算器的输入输出模型为，

输入	说明	输出	说明
A	16 位操作数 A	Fout	16 位运算结果
B	16 位操作数 B	overflow	溢出信号，为 1 时溢出
ALUop	运算控制信号，控制，控制 A 与 B 进行的运算	zero	零标志位：当输出结果为 0 时，该标志位为 1.

表格 3. 2 运算器输入输出关系表

1．程序计数器 PC

程序计数器 PC 用一个 16 位的寄存器来实现，其触发方式为下降沿触发。需要接入时钟信号和清零信号，其中时钟信号接入时钟端，清零信号接入清零端，清零信号方便调试。

2．指令存储器 IM

指令储存器 IM 由一个地址位为 16 位，数据位为 16 位的 ROM 实现。

3．寄存器堆 RF

首先给出寄存器对 RF 的输入输出关系：

输入	说明	输出	说明
r1	读数据选择端口	Do1	数据输出端口，输出 r1 选择的寄存器的内容
r2	读数据选择端口	Do2	数据输出端口，输出 r2 选择的寄存器的内容
w	写数据选择端口	R3R2	32 位输出端口，前 16 位为 3 号寄存器内容，后 16 位为 2 号寄存器内容
Din		R1R0	32 位输出端口，前 16 位为 1 号寄存器内容，后 16 位为 0 号寄存器内容
RegWrite	寄存器写控制信号,为 1 时可以写入		
CP	时钟信号		
clear	清零信号，为 1 时对 RF 所有寄存器清零		

表格 3. 3 寄存器堆输入输出关系表

4．指令译码器

指令译码器以指令作为输入，按照指令结构将指令的内部结构信息输出。

根据 表格 3.1 指令系统表，其输入输出模型为：

输入	说明	输出	说明
instruction	16 位的指令	Funct	Funct 字段 3bit
		P	P 字段 2bit
		rd	rd 字段 2bit
		rt	rt 字段 2bit
		rs	rs 字段 2bit
		OP	OP 字段 4bit
		immediate	偏移地址和立即数字段 8bit
		jump_address	无条件跳转地址字段 12bit

表格 3．4 指令译码器输入输出表

3.1.3 其它功能部件

1．跳转地址形成部件

当有跳转指令的时候需要计算出跳转地址，这一计算过程我打算封装成一个子电路。

其输入输出关系表如下表所示：

输入	说明	输出	说明
PC+1	当前指令地址加“1”的结果	Jump-address	跳转地址 16 位
jump_address	无条件跳转地址 12bit		
offset	偏移地址 8bit		
OP	操作码 4bit		

表格 3. 5 跳转地址形成部件

2．启停控制器

因为 CPU 的时钟信号应该以准确固定的方式进入电路，如固定第一个边沿信号为上升沿，或者第一个边沿信号为下降沿。该电路的第一个边沿信号是下降沿。

其输入输出关系表为：

输入	说明	输出	说明
CP	时钟信号	clock	由启停控制器控制的时钟信号
clear	清零信号，要用到 D 触发器，该信号给触发器清零		
start	启动信号，为 1 时启动		

表格 3. 6 启停控制器

3．slt 写回结果形成部件

根据 表格 3.1 指令系统表 的 slt 指令说明，我打算用减法来取得 slt 的写回结果，因此需要一个专门的部件来形成这个写回，其输入输出关系为：

输入	说明	输出	说明
flag	运算结果符号位	slt	1 比特 为 0 或 1
overflow	运算器溢出信号		
OP	指令操作码		
Funct	指令的 Funct 字段		

表格 3. 7 slt 写回结果形成部件

4．跳转控制信号形成部件

为避免控制器的输入信号太多（控制器的设计后面会有说明），我单独用一个子电路来计算跳转地址信号。

其输入输出关系表为：

输入	说明	输出	说明
OP	指令操作码	branch	1bit，为 1 时说明要跳转
zero	零标志位，1bit 。由运算器给出		
+/-	1bit，运算器理想输出结果的正负性，为 0 时依然为正。		

表格 3. 8 跳转控制信号输入输出表

5．ALU 控制信号形成部件

根据表格 3.1 指令系统表，无法直接从指令直接获得 ALU 的运算控制信号，因此需要一个专门的电路来做这样的转换操作。

因为 ALU 的控制信号由操作码 OP 和 Funct 字段唯一决定，因此其输入输出关系为：

输入	说明	输出	说明
OP	指令操作码 4bit	ALU-op	ALU 运算控制信号
Funct	指令 Funct 字段 3bit		

表格 3. 9 ALU 控制信号形成部件

6．DISP 部件

DISP 主要用于显示寄存器的值。现给出其输入输出关系表：

输入	说明	输出	说明
R3R2	寄存器堆的 3 号和 2 号寄存器内容	DISP-0	红色显示器
R1R0	寄存器堆的 1 号和 0 号寄存器内容	DISP-1	蓝色显示器
r_num	寄存器选择信号，用于选择寄存器，2bit		
CLK	时钟信号		
disp_color	显示的颜色，为 0 时红色，为 1 时蓝色		
CLR	清零信号，给 DISP 中的寄存器清零		
disp	控制信号，用于控制是否更新寄存器		

表格 3. 10 DISP 部件

3.1.4 构造寄存器写控制信号

R 型指令和部分其他指令需要将值写回寄存器，因此需要寄存器写控制信号，记作：RegWrite。

RegWrite 控制信号的输入输出关系为：

输入	说明	输出	说明
OP	指令操作码	RegWrite	寄存器写控制信号，为 1 时说明要写

表格 3. 11 RegWrite 控制信号输入输出关系

3.1.5 构造寄存器写选择端输入的选择信号

根据 **表格 3.1 指令系统表** 寄存器堆的写选择端的输入有可能是 rt，也有可能是 rd，因此需要用一个控制信号对他们进行选择，该信号记作：RegDst。

RegDst 控制信号的输入输出关系为：

输入	说明	输出	说明
OP	指令操作码	RegDst	寄存器写选择控制信号，为 1 时选择 rd

表格 3. 12 寄存器写选择控制信号

3.1.6 构造运算器输入选择控制信号（ALUsrc）

根据 **表格 3.1 指令系统表** addi 指令和 R 型指令的输入到运算器的数据有冲突，其中 R 型指令输入到运算器的信号为$rs,$rt,而 addi 输入到运算器的信号为$rs, immediate, 因为 $rt 来自寄存器堆，而 immediate（扩展后的立即数）来自指令的立即数地段的扩展结果，因此需要一个信号对他们做出选择。该信号命名为：ALUsrc。

ALUSrc 控制信号的输入输出关系为：

输入	说明	输出	说明
OP	指令操作码	ALUSrc	运算器输入选择控制信号。

3.1.7 构造 RAM 的写控制信号（Memwrite）

指令 lw 需要将运算器的运算结果写入 RAM，因此需要一个信号来控制其写的行为。该信号命名为：Memwrite。

Memwrite 控制信号的输入输出关系为：

输入	说明	输出	说明
OP	指令操作码	Memwrite	RAM 写控制信号。

表格 3. 13 Memwrite

3.1.8 构造写回寄存器堆数据的选择控制信号（MemtoReg）

指令 sw 需要从 RAM 获取的值写入寄存器堆，同时，R 型指令也需要将计算结果写回寄存器堆，这造成冲突，因此需要一个信号来控制其写的行为。该信号命名为：MemtoReg。

MemtoReg 控制信号的输入输出关系为：

输入	说明	输出	说明
OP	指令操作码	MemtoReg	寄存器堆写入数据选择控制信号。

表格 3. 14 MemtoReg

3.1.9 其他控制信号

其他控制信号的输入输出关系与 MemtoReg 一样，分别为 lui，disp，slt，halt。

其中 disp 控制信号接入 DISP 部件的 disp 端，lui，slt，halt 控制信号会在详细设计中给出

3.1.10 控制器封装

控制器提供的提供的 3.1.4 到 3.1.9 的控制控制信号，以及 ALUop 形成部件故将他们放在一起，然后封装成一个子电路即可。

根据 3.1.4 到 3.1.9 以及 ALUop 形成部件，控制器的输入输出关系表如下：

输入	说明	输出	说明
OP	指令操作码	slt	略
Funct	指令Funct字段	disp	略
		RegDst	略
		lui	略
		Memwrite	略
		halt	略
		MemtoReg	略
		RegWrite	略
		ALUsrc	略
		ALUop	略

表格 3. 15 控制器封装

。

3.2可支持理想流水线的多周期 CPU 设计

3.2.1 总体设计

指令格式的设计的寻址方式的设计等都由 表格 3. 1 指令系统表 给出。结合单周期 CPU 的电路的电路图，为实现五段理想流水，只需要将电路图进行功能周期划分，划分的五段周期分别为 ID，IF，EX， MEM，WB。其中 ID 为取指阶段，IF 为译码阶段，EX 为执行阶段，MEM 为访问 RAM 阶段，WB 为写回阶段。

由此得到 4 个接口 ，分别为 ID/IF，IF/EX, EX/MEM, MEM/WB。

因为该阶段不考虑冲突，因此没有冲突检测和冲突处理部分。

3.2.2 接口部件设计

为方便调试，PC 的值和指令传送到每一个接口。下面给出各个接口的储存内容。同样，各个接口的控制信号都是一样的，非别为使能信号，时钟信号，清零信号。这些公共部分略去。

1. ID/IF

   该接口需要寄存 PC，PC+1，还有指令。

2. IF/EX

   因为 EX 以及之后的阶段会用到控制信号，因此除公共内容为，还要将由控制器解析出的控制信号寄存到该接口。

3. EX/MEM

   MEM 阶段需要用到控制信号，故还需要添加存放控制信号的寄存器。

4. MEM/WB

   WB 写回阶段同样需要用到控制信号，故还需要添加存放控制信号的寄存器。

3.2.3 流水冲突检测器

设指令进入电路的顺序为 A,B,C 当 A,B 都进入电路且 A，B 之间没有冲突，在 C 要进入电路的时候即要判断(C,A)的冲突，又要判断(C,B)的冲突，两个只要有一个发生冲突，就说明有冲突，如果用气泡来处理冲突，则需要插入一个气泡，然后在进行判断，直到没有冲突的时候 C 才能进入电路。因此该冲突检测器的输入应该有三个，又因为(C,A)与 (C,B)的结构相同，因此可以将检测两条指令冲突的电路封装起来，然后在冲突检测器里使用两个 2 指令冲突检测器，其结果相或即可得冲突结果。

冲突有三种，数据冲突，控制冲突，结构冲突，其中结构冲突在程序和数据分离的 CPU 中不存在，因此冲突只有数据冲突和控制冲突。其冲突检测的详细设计请看详细设计部分。

3.2.4 插入气泡的流水冲突处理

在检测到冲突之后就可以进行气泡插入处理了，插入气泡需要以下两个处理。

1. 向前阻塞

向前提供阻塞信号，使得 PC 暂停取下一条指令，这个可以用冲突检测器的输出信号取反后接入到 PC 的使能端。

2. 向后插入气泡

向后提插入气泡，插入的气泡必须与所有指令都没有冲突的指令。

可以考虑的指令为：0x0000

详细设计的细节请看详细设计部分。

3.2.5 数据重定向的流水冲突处理

数据重定向可以由两个单元来实现，一个是对运算器的运算输入的数据来源的重定向，另一个是对 sw 指令中写入 RAM 内存的写入数据的重定向。

在 EX 阶段

当 EX 与 MEM 有数据冲突的时候，需要将 MEM 阶段的写会的数据重定向到

ALU 的指定的输入端，其来源有两个可能，一个是 MEM 阶段的 ALU—OUT，即上一条指令在运算器中运算的到的输出结果，另一个是 MEM 阶段的 RAM 的输出结果，此时的 MEM 的指令一定要是 lw 指令。

当 EX 与 WB 有数据冲突的时候，需要将 WB 阶段的写会寄存器堆的数据重定向到 ALU 的指令输入端。

当 sw 的指令与之前的指令有冲突的时候，同样需要进行数据重定向操作。

在此处给出两个重定向单元的输入输出模型，详细部分见详细设计。

输入	说明	输出	说明
EX	EX 阶段指令	TO_A	接入运算器输入端 A 的数据
MEM	MEM 阶段指令	TO_B	接入运算器输入端 B 的数据
WB	WB 阶段指令		
MEM_ALUout	MEM 阶段的 ALU-out		
MEM_MEMout	RAM 输出		
WB_in	WB 阶段的写回寄存器堆的数据		
original_a	原运算器接入输入端 A 的数据		
original_b	原运算器接入输入端 B 的数据		

表格 3. 16 运算器输入数据重定向

输入	说明	输出	说明
original-in	原来接入RAM的D端的数据	TO_D	接入到RAM的D端的数据
$rt_1	ID阶段的RF的Do2端输出的数据		
WB _in	WB 极端的写回寄存器堆的数据		
MEM	MEM 阶段指令		
WB	WB 阶段指令		
END_1	比 WB 阶段指令早一步执行的指令		

表格 3. 17 sw 数据重定向

3.2.6 动态分支预测的流水冲突处理

4 详细设计与实现

4.1 构建单周期 CPU

4.1.1 主要功能部件实现

1．程序计数器 PC

选用的元器件： 计数器（数据位宽： 16， 最大值： 0xffff，溢出时操作：重新计数，触发方式：下降沿，标签： PC）

输入： 时钟信号、清零信号。时钟和清零信号是主电路的时钟和输入引脚输入，所以设计为隧道方式引入到 PC 部分。

输出： 计数器的输出端。

具体实现： 实现效果如下图所示：

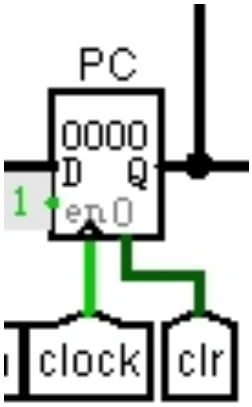


图 4. 1 PC 的电路实现

2．运算器 ALU

根据 表格 3.1 指令系统表，得到需要的运算，构造的运算器如下：

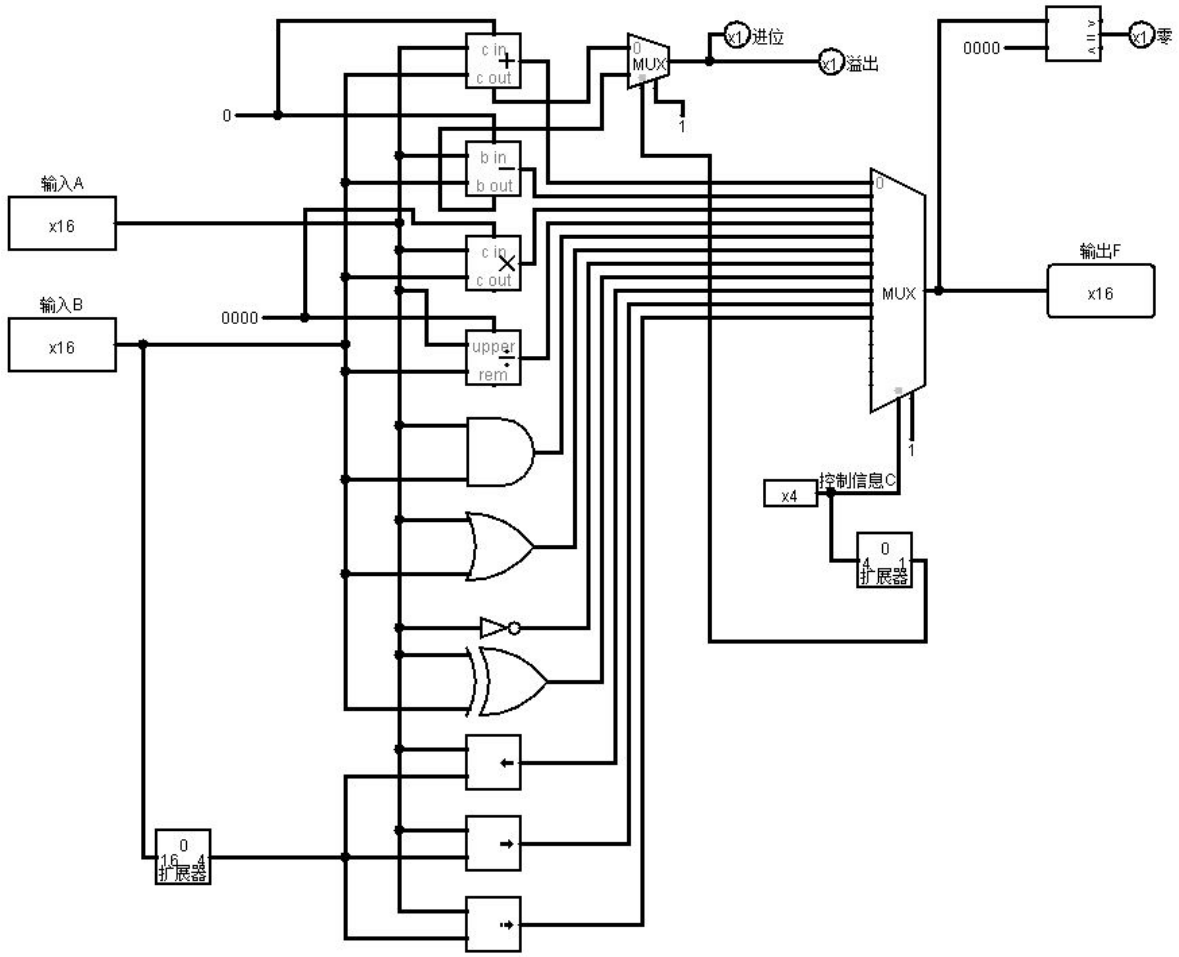


图 4. 2 运算器 ALU

3．寄存器堆

根据 表格 3.3 寄存器堆输入输出关系表 ，设计出来的基础器对 RF 的电路图如下图所示：

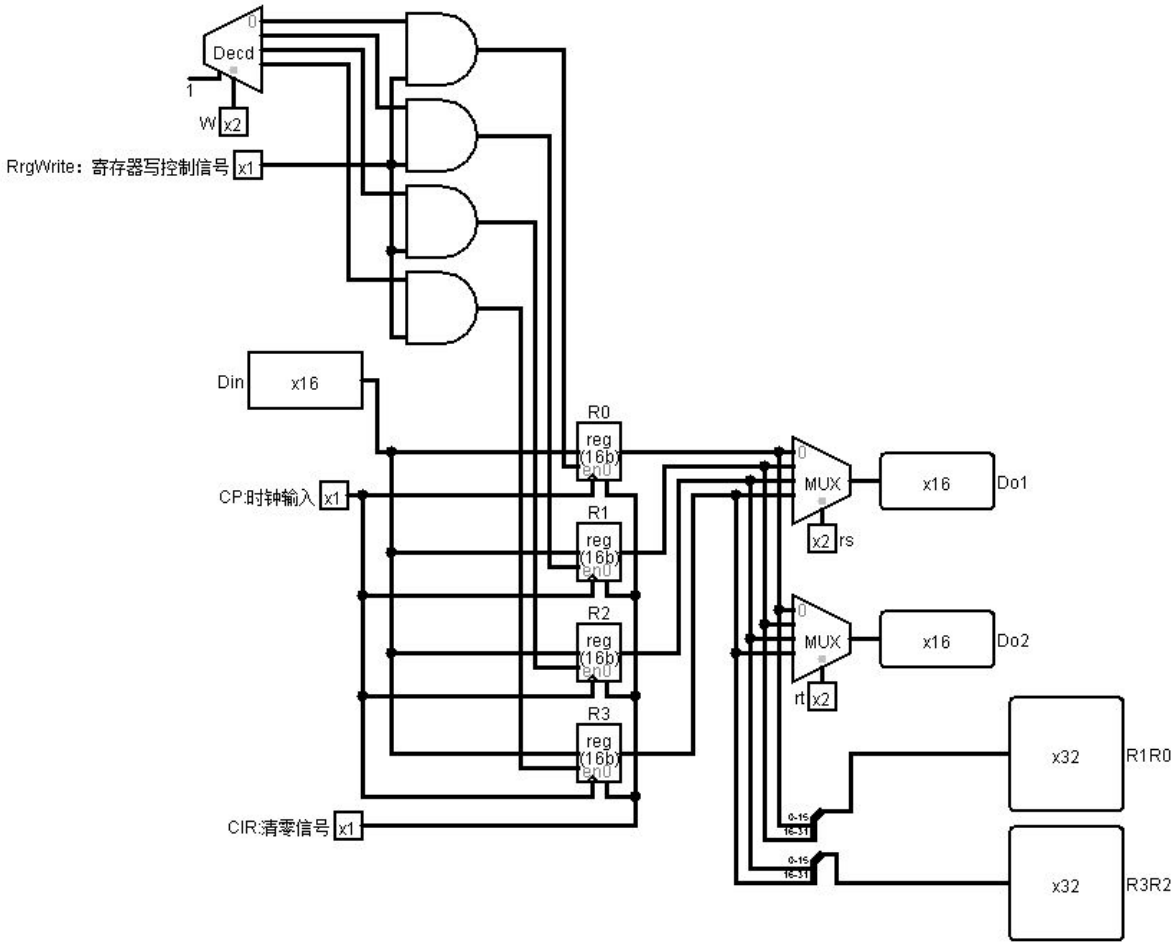


图 4. 3 寄存器堆 RF

4．指令译码器

根据 表格 3.4 指令译码器输入输出表 设计出的指令译码器的电路图如下图所示：

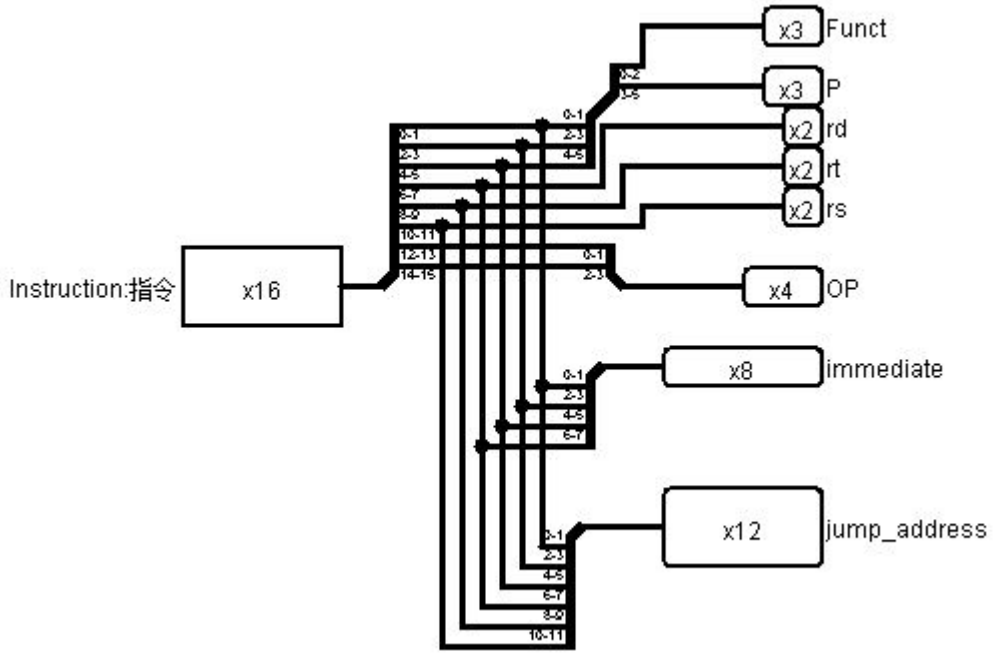


图 4. 4 指令译码器

4.1.2 跳转地址形成部件

根据 表格 3.5 跳转地址形成部件 ，得到的电路图为：

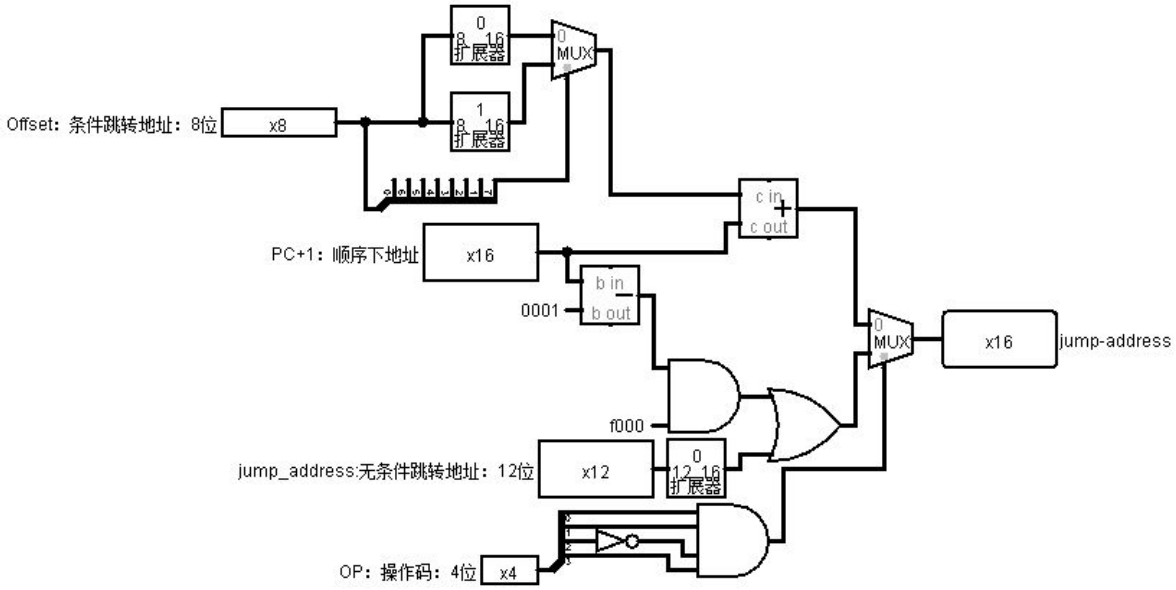


图 4. 5 跳转地址形成部件

4.1.3 slt 写回结果形成部件

根据 表格 3.6 slt 写回结果形成部件，得到的电路图为：

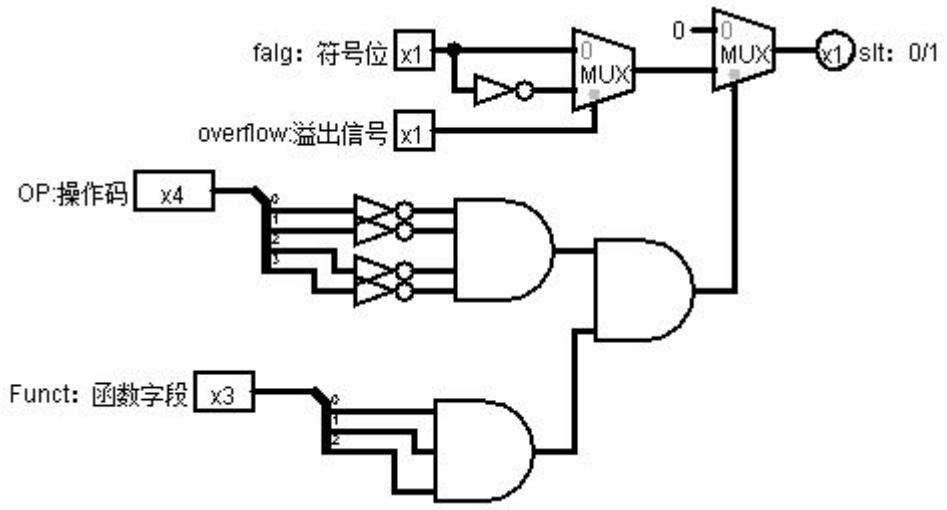


图 4. 6 slt 写回结果形成部件

4.1.4 跳转信号形成部件

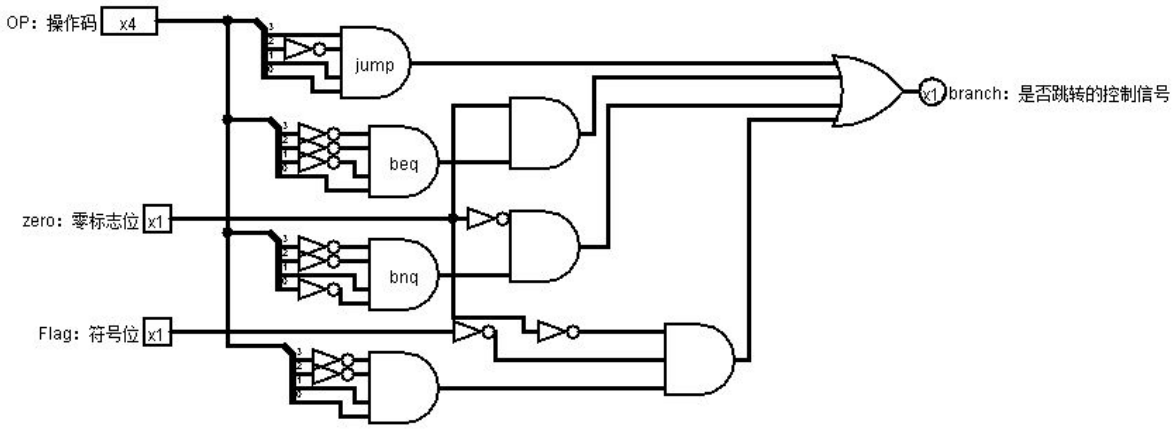


图 4. 7 跳转控制信号

4.1.5 ALU 控制信号形成部件

输入：指令的操作码字段 OP，指令的函数字段 Funct。

输出：ALUop

16 位运算器的 op 对应的运算关系表，如下表所示。

S3S2S1S0	十进制	功能	输入输出关系
0000	0	加	F ←A+B
0001	1	减	F ←A-B
0010	2	乘	F←(A*B)低 16 位
0011	3	除	F ← （A/B）不完全商
0100	4	逻辑与	F ← A and B
0101	5	逻辑或	F ← A or B
0110	6	逻辑非	F ←~A
0111	7	异或	F ←（A xor B）
1000	8	逻辑左移	F A 逻辑左移 B 低 4 位
1001	9	逻辑右移	F←A 逻辑右移 B 低 4 位
1010	10	算数右移	F←A 算术右移 B 低 4 位

表 4. 1 运算器的操作与控制信号对应表

再根据 表格 3.1 指令系统表 得到 OP，Funct，ALUop 的关系表。如下表所示。

指令	OP（15~12）	Funct(2~0)	运算	ALUop(3-0)
or	0	0	或	0101
and	0	1	与	0100
add	0	2	加	0000
sub	0	3	减	0001
sllv	0	4	逻辑左移	1000
srlv	0	5	逻辑右移	1001
srav	0	6	算术右移	1010
slt	0	7	减	0001
beq	1	xxxx	减	0001
bne	2	xxxx	减	0001
bgt	3	xxxx	减	0001
DISP	4	xxxx	无	0101（或）
lui	5	xxxx	逻辑左移	1000
ori	6	xxxx	或	0101
andi	7	xxxx	与	0100
addi	8	xxxx	加	0000
lw	9	xxxx	加	0000
sw	10	xxxx	加	0000
jump	11	xxxx	无	0101（或）
halt	12	xxxx	无	0101（或）

表 4. 2 指令与运算对应表

因此：该译码器需要两个部件，一个完成 Funct 到 ALUop 的映射，（当 OP== 0 才能使用这个）

另一个是 OP 到 ALUop 的映射，（当 OP 非 0 时使用）。

其中：OP-->ALUop 的译码器的真值表如下：

OP3	OP2	OP1	OP0	ALUop3	ALUop2	ALUop1	ALUop0
0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	1
0	0	1	0	0	0	0	1
0	0	1	1	0	0	0	1
0	1	0	0	0	1	0	1
0	1	0	1	1	0	0	0
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	1
1	1	0	0	0	1	0	1
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

图 4. 8 OP->ALUop 真值表

得到的电路为：

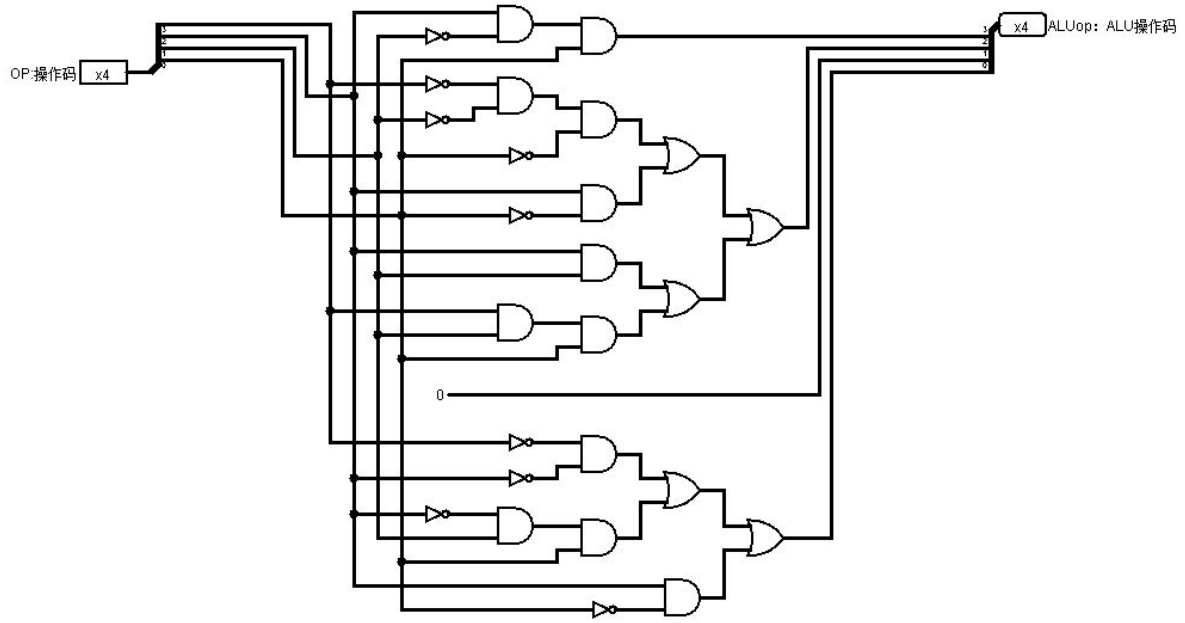


图 4. 9 OP->ALUop

Funct-ALUop 真值表如下：

Funct2	Funct1	Funct0	ALUop3	ALUop2	ALUop1	ALUop0
0	0	0	0	1	0	1
0	0	1	0	1	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	1
1	0	0	1	0	0	0
1	0	1	1	0	0	1
1	1	0	1	0	1	0
1	1	1	0	0	0	1

得到的电路如下：

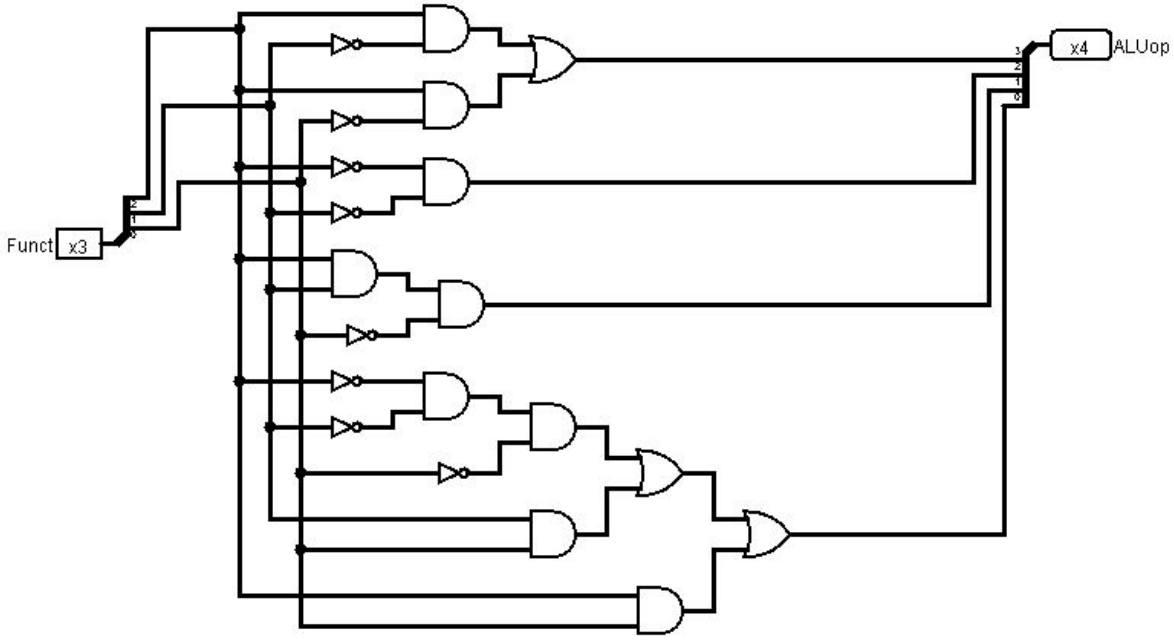


图 4. 10 Funct->ALUop

两个部件结合构造一个 Funtc，OP 结合计算 ALUop 的译码器，两个部件分别处理自己的输入，然后将结果输出到一个二路选择器，二路选择器的选择信号由操作码（OP）的各 bit 为相或给出。

如下图所示：

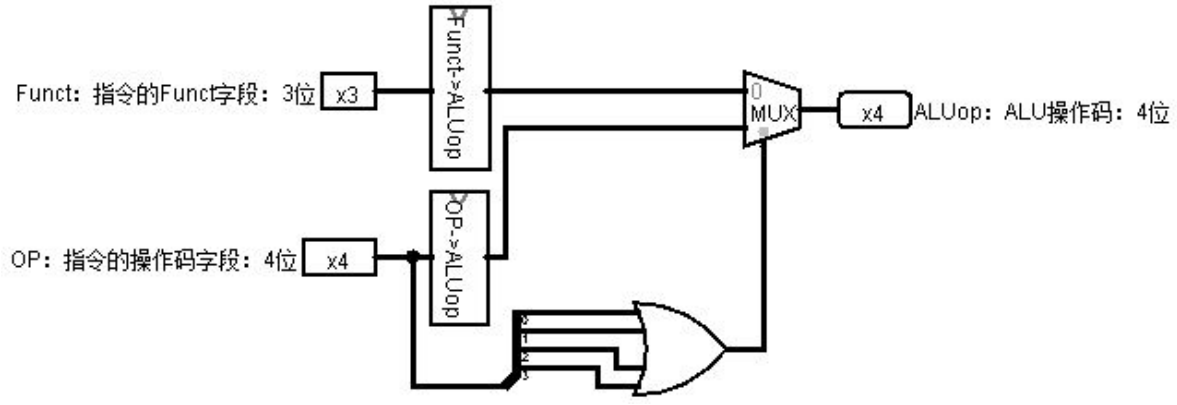


图 4. 11 ALUop 译码器

4.1.6 DISP 部件

根据 表格 3.9 DISP 部件 ，可得到输入输出关系，另外因为输入到 DISP 的结果需要保存，直到下一条 DISP 指令出现，因此需要两个寄存器，一个寄存 DISP-0，一个寄存 DISP-1.

实现后的电路如下图所示：

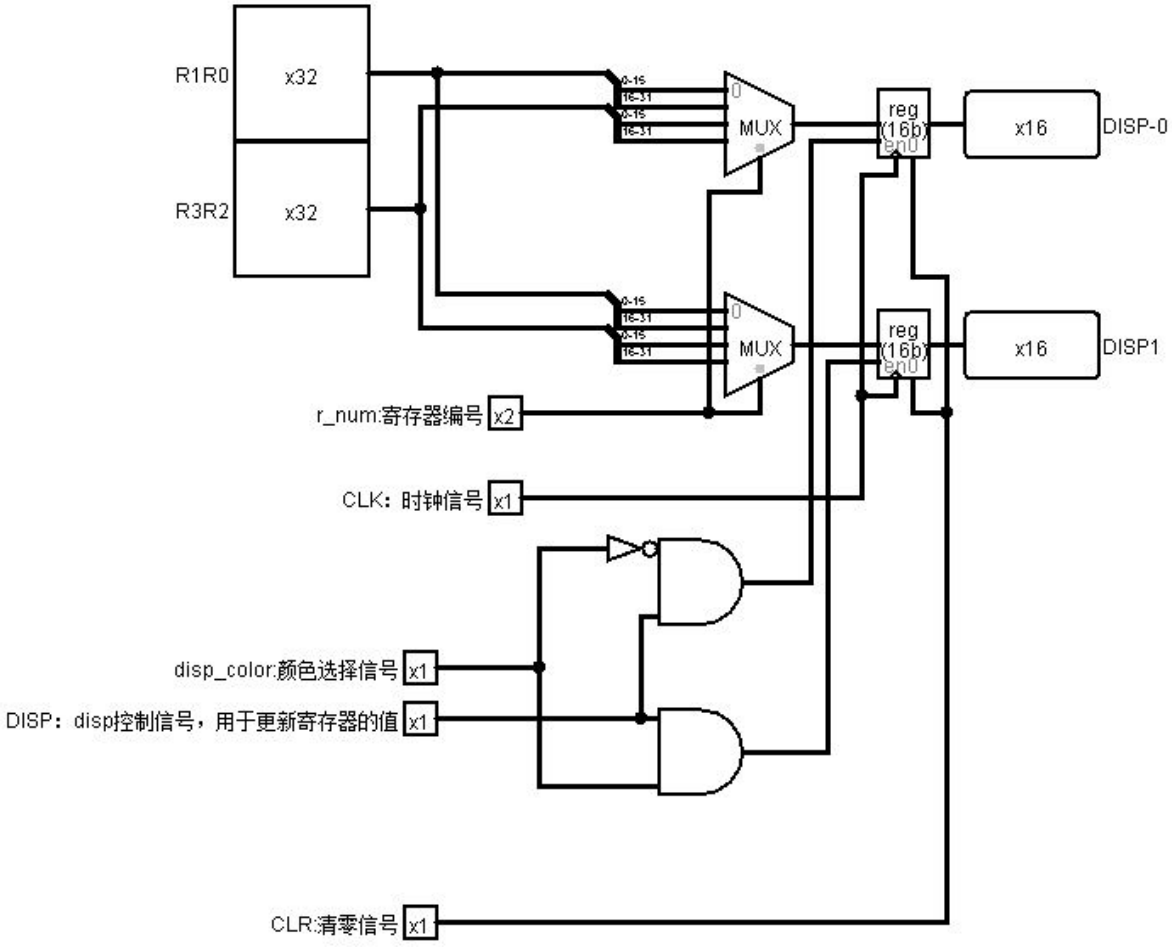


图 4. 12 DISP 部件

4.1.7 寄存器写控制信号

RegWrite 信号是控制寄存器堆的写控制信号。

列出需要 RegWrite 信号的指令。可以看出它只与操作码有关。

#	指令	OP(15~12)	rs(11~10)	rt(9~8)	7~6	5~3	2~0	指令功能
1	or	0	rs	rt	rd	0	0	$rd = $rs | $rt（5~3 位无用）
2	and	0	rs	rt	rd	0	1	$rd = $rs & $rt
3	add	0	rs	rt	rd	0	2	$rd = $rs + $rt
4	sub	0	rs	rt	rd	0	3	$rd = $rs - $rt
5	sllv	0	rs	rt	rd	0	4	$rd = $rs << $rt
6	srlv	0	rs	rt	rd	0	5	$rd = $rs >> $rt
7	srav	0	rs	rt	rd	0	6	$rd = $rs >> $rt 算术右移
8	slt	0	rs	rt	rd	0	7	$rd = ($rs < $rt) ? 1 :0
13	lui	5	0	rt	immediate-u			$rt = imm << 8
14	ori	6	rs	rt	immediate-u			$rt = $rs | imm
15	andi	7	rs	rt	immediate-u			$rt = $rs & imm
16	addi	8	rs	rt	immediate-s			$rt = $rs + imm
17	lw	9	rs	rt	immediate-s			$rt = MEM[$rs + imm]

表 4. 3 用到 RegWrite 信号的指令

寄存器堆 RF 的寄存器改成下降沿触发。

得到的电路图如下图所示：

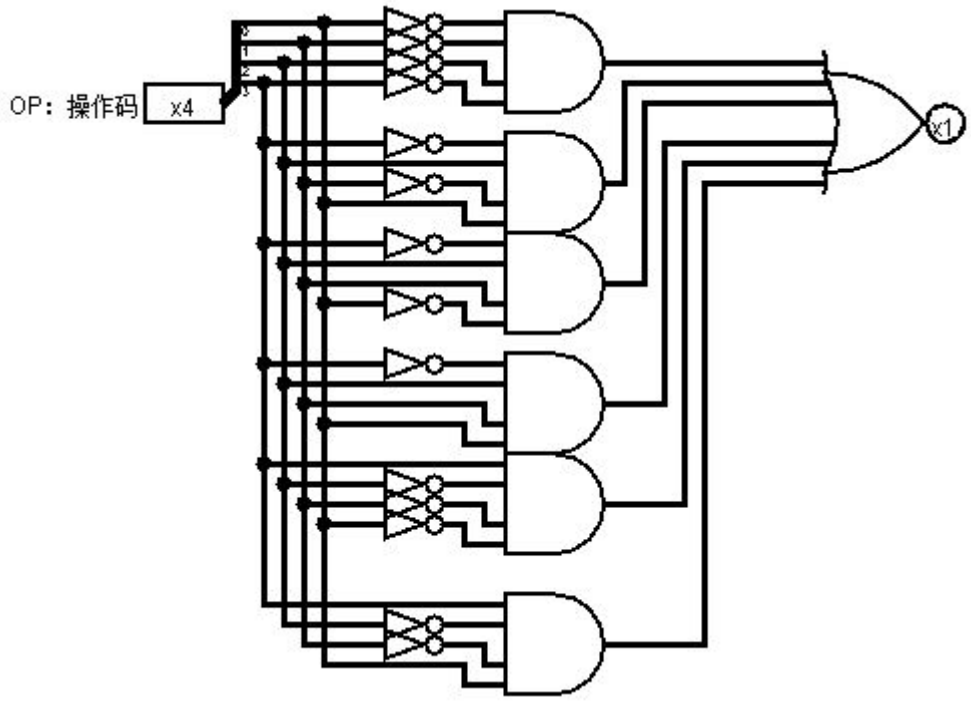


图 4. 13 RegWrite 控制信号

4.1.8 运算器输入选择控制信号

因为运算器的其中一端的输入有可能是立即数也有肯是寄存器的数据，该控制部件用于选择参与运算器的其中一端的输入信号。

以下是需要用到运算器的指令：排除 13 号指令 lui，该指令的运算器另外实现。原因是 immediate 字段放在运算器的移位端（当进行移位运算的时候），所以用运算器实现起来太复杂。

#	指令	15~12	11~10	9~8	7~6	5~3	2~0	指令	指令功能
1	or	0	rs	rt	rd	0	0	or	$rd = $rs | $rt（5~3 位无用）
2	and	0	rs	rt	rd	0	1	and	$rd = $rs & $rt
3	add	0	rs	rt	rd	0	2	add	$rd = $rs + $rt
4	sub	0	rs	rt	rd	0	3	sub	$rd = $rs - $rt
5	sllv	0	rs	rt	rd	0	4	sllv	$rd = $rs << $rt
6	srlv	0	rs	rt	rd	0	5	srlv	$rd = $rs >> $rt
7	srav	0	rs	rt	rd	0	6	srav	$rd = $rs >> $rt 算术右移
8	slt	0	rs	rt	rd	0	7	slt	$rd = ($rs < $rt) ? 1 :0
9	beq	1	rs	rt	offset-s			beq	beq =?
10	bne	2	rs	rt	offset-s			bne	bne != ?
11	bgt	3	rs	rt	offset-s			bgt	bgt >?（有符号比较）
13	lui	5	0	rt	immediate-u			lui	$rt = imm << 8
14	ori	6	rs	rt	immediate-u			ori	$rt = $rs | imm
15	andi	7	rs	rt	immediate-u			andi	$rt = $rs & imm
16	addi	8	rs	rt	immediate-s			addi	$rt = $rs + imm
17	lw	9	rs	rt	immediate-s			lw	$rt = MEM[$rs + imm]
18	sw	10	rs	rt	immediate-s			sw	MEM[$rs+imm] = $rt

表 4. 4 用到运算器的指令

（如果 lui 单独实现后在写会阶段，也就是赋值给$rt 阶段还要做一个二路选择，这同样会给电路增加相当的复杂度。）

尝试在 ALU 输入端增加二路选择器，再加控制器的控制信号。

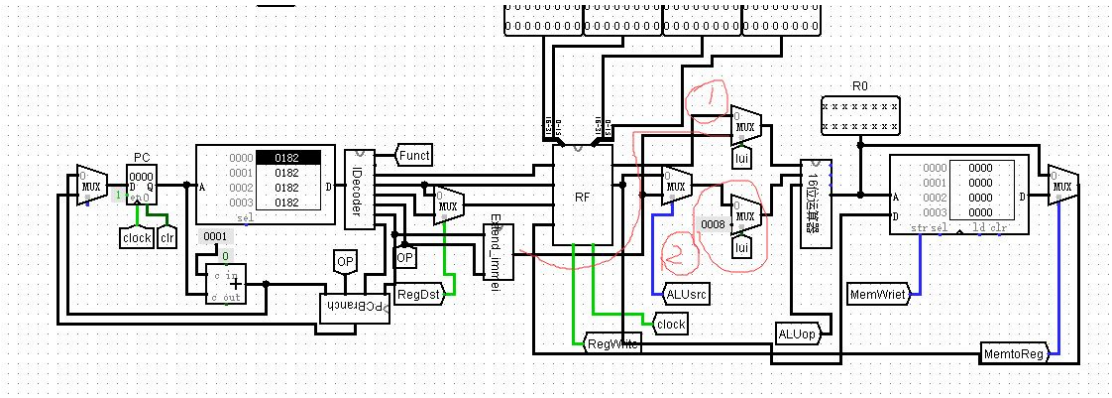


图 4. 14 lui 单独实现电路

在输入端增加数据选择，控制信号为 lui。因为 lui 指令的 OP 为 5(0101)，所以 lui 控制电路部分为

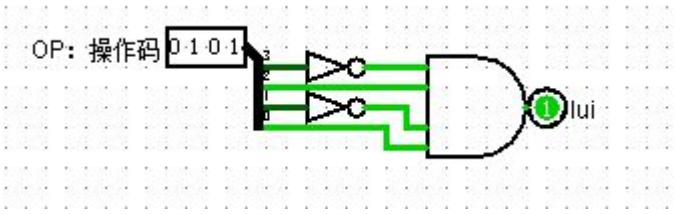


表 4. 5 lui 控制信号

在 ALUSrc 看是部分的表格可以看出，当 OP 输入{6,7,8}时选择立即数作为运算器的 B 端输入。

因此可得真值表：

OP3	OP2	OP1	OP0	ALUSrc
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

表 4. 6 OP -> ALUsrc 映射真值表

得到电路。

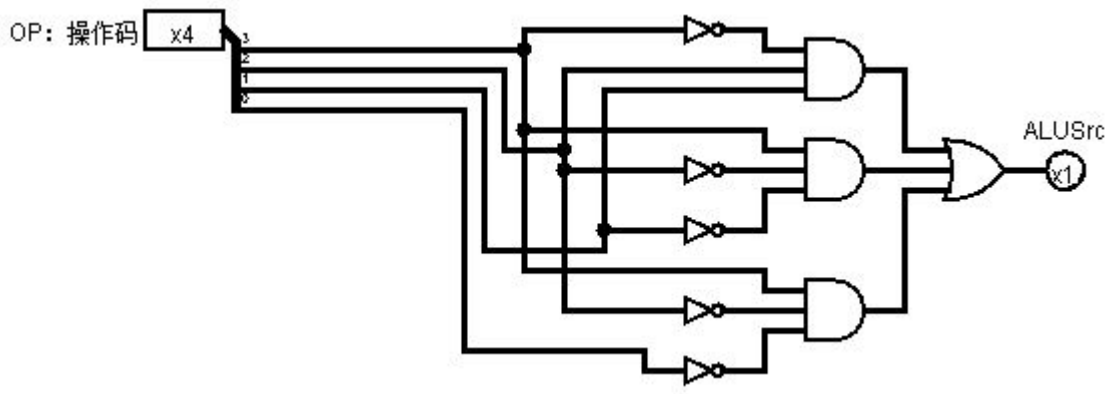


图 4. 15 ALUsrc 控制信号

4.1.9 RAM 写控制信号

由 sw 指令的 OP 可以得到其控制信号电路图：

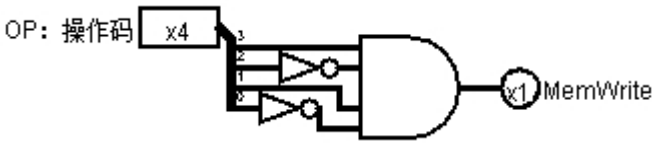


图 4. 16 RAM 写控制信号

4.1.10 启停控制器

参考教材第六章中央处理器的启停逻辑的设计思想，自己设计了一个针对该电路的启停逻辑。启停逻辑如下图所示。

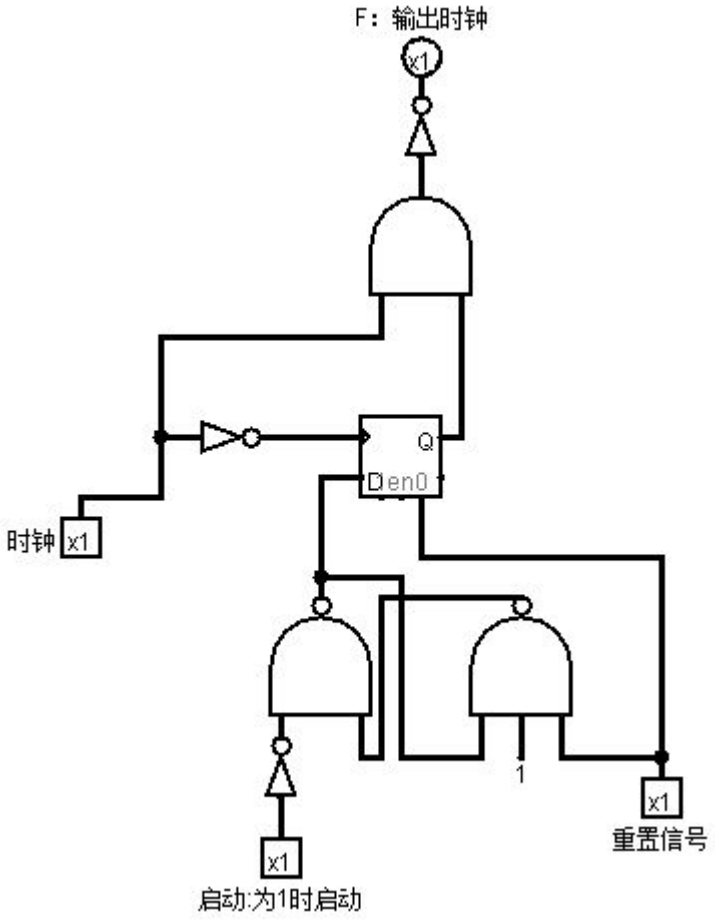


图 4. 17 启停控制器

4.1.11 MemtoReg

由 lw 指令得 OP 可以得到其控制信号电路图：

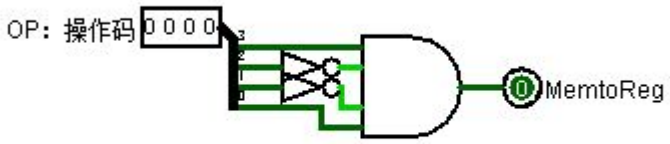


图 4. 18 MemtoReg 控制信号

4.1.12 disp

由 DISP 指令的 OP 可以得到其控制信号电路图：

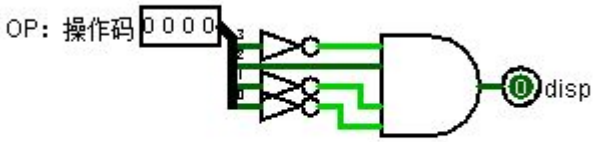


图 4. 19 disp

4.1.13 halt

由 halt 指令的 OP 可以得到其控制信号电路图：

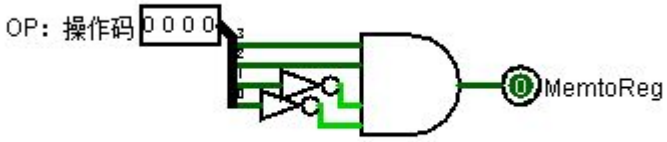


图 4. 20 halt

4.1.14 slt

由 slt 指令可以得到其控制信号电路图：

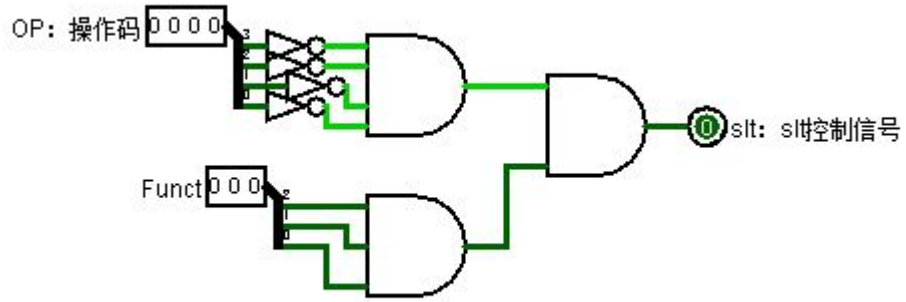


图 4. 21 slt 控制信号

4.1.15 控制器封装

依据表格 3. 14 控制器封装 可得到控制器的电路图（注：其中用到的很多电路都封装成了子电路）：

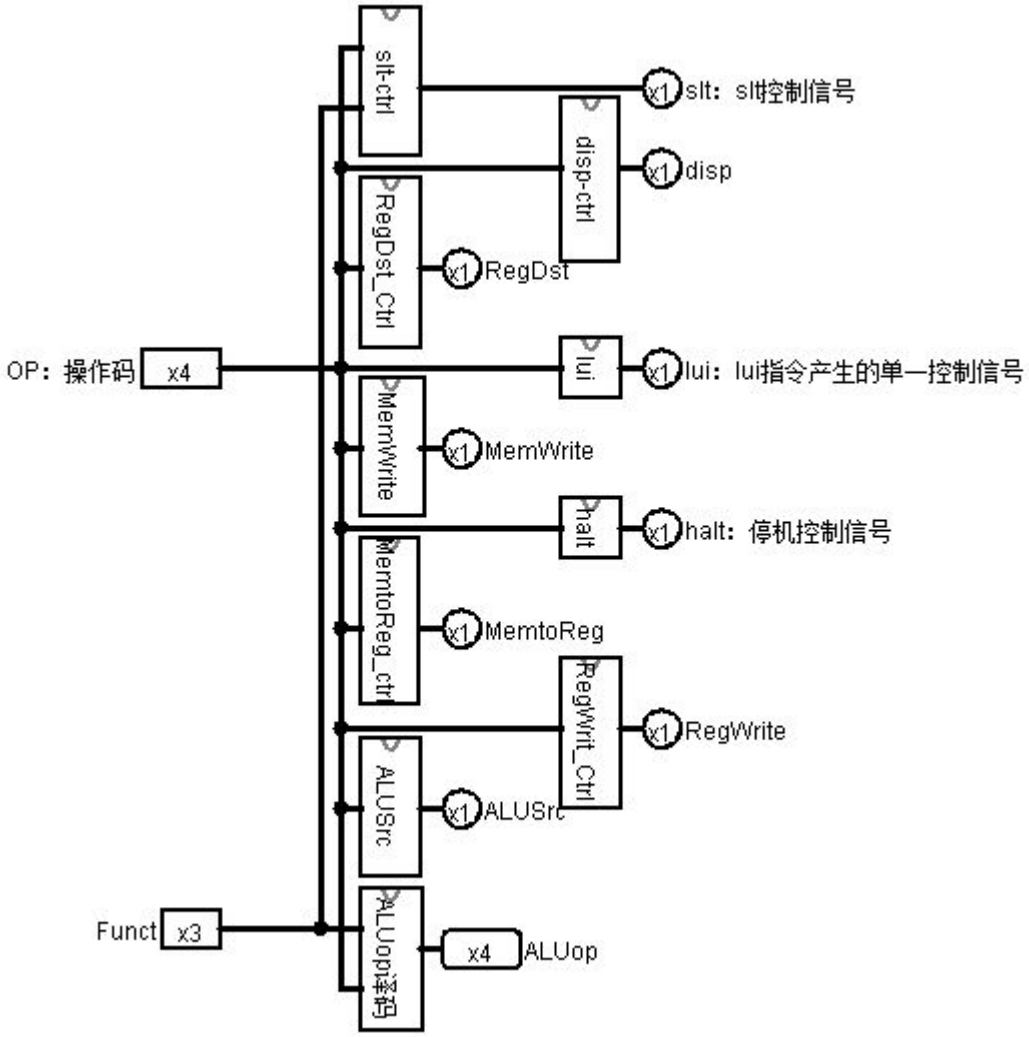

slt-ctrl
slt：slt控制信号
disp-ctrl
disp
RegDst_Ctrl
RegDst
OP：操作码
x4
lui
lui：lui指令产生的单一控制信号
MemWrite
MemWrite
halt
halt：停机控制信号
MemtoReg_ctrl
MemtoReg
RegWrit_Ctrl
RegWrite
ALUSrc
ALUSrc
Funct
x3
ALUop译码
x4
ALUop

图 4. 22 控制器

4.2可支持理想流水线多周期 CPU

1. IF/ID

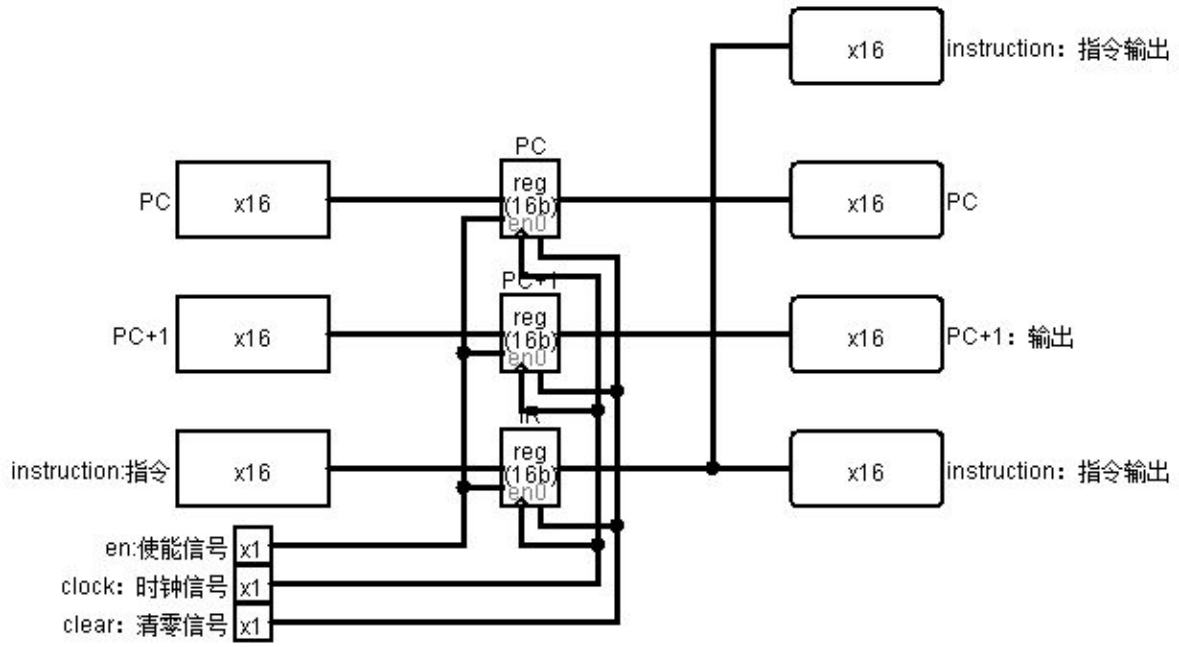


图 4. 23 IF/ID 实现

2. ID/EX

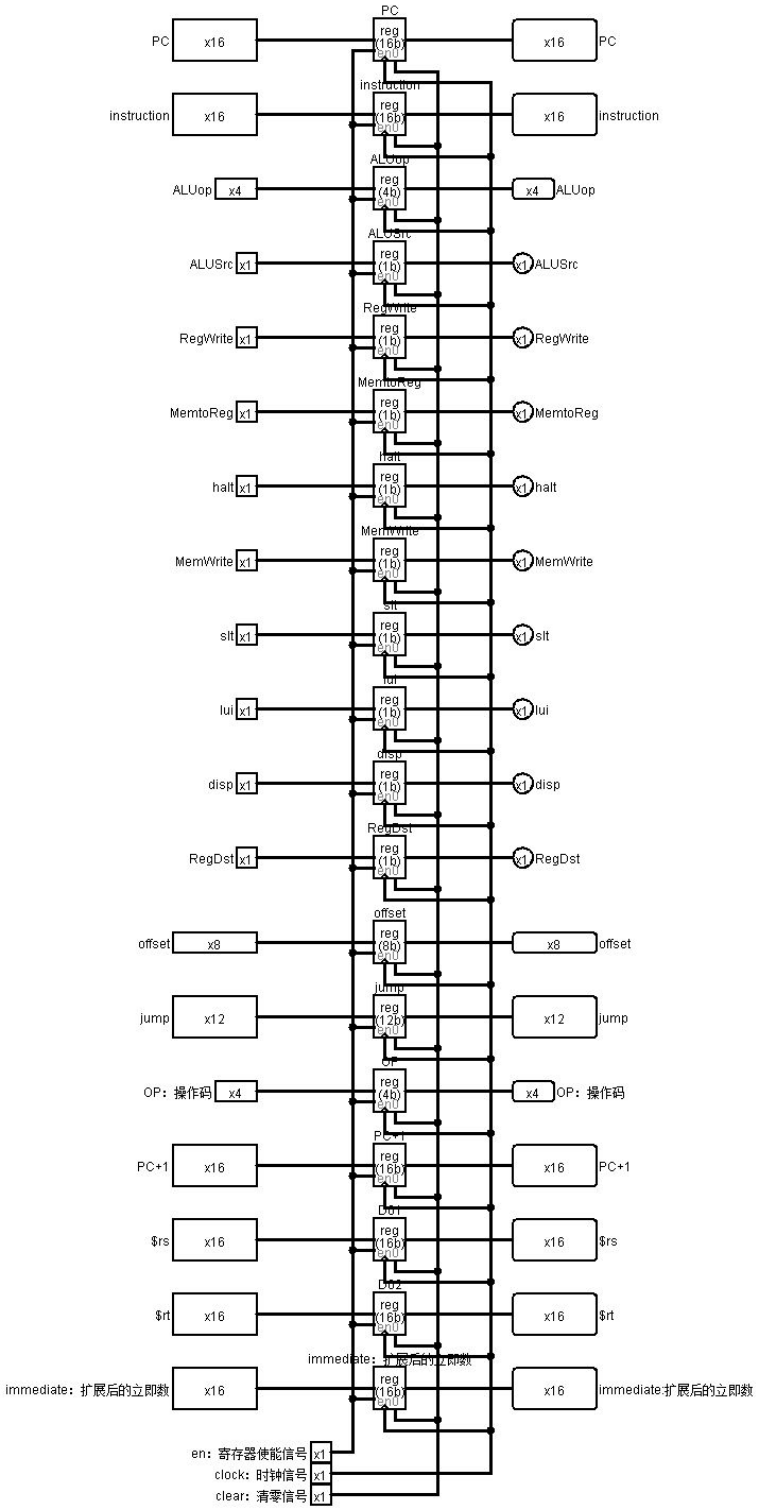


图 4. 24 ID/EX 实现

3. EX/MEM

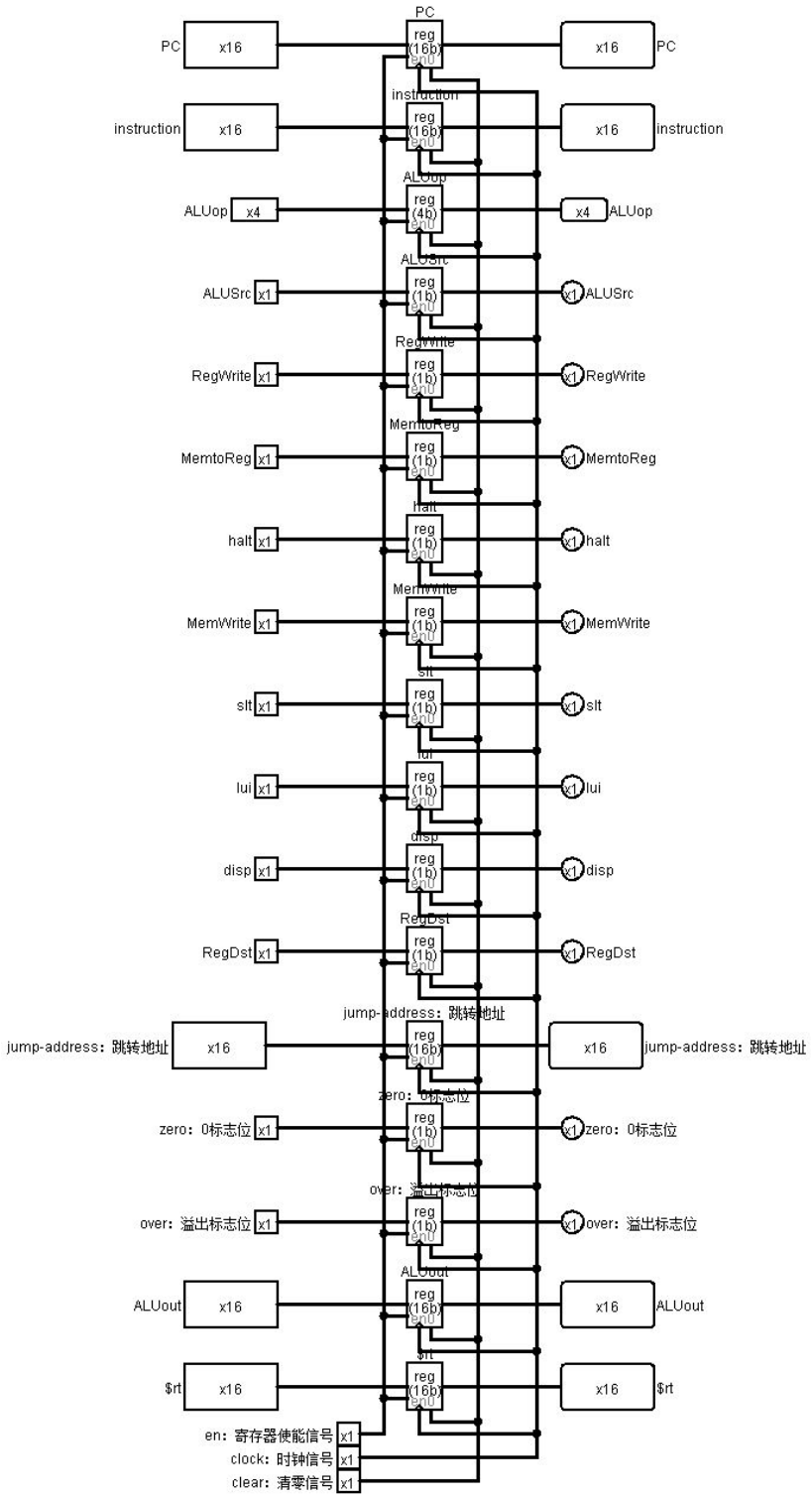


图 4. 25 EX/MEM 实现

4. MEM/WB

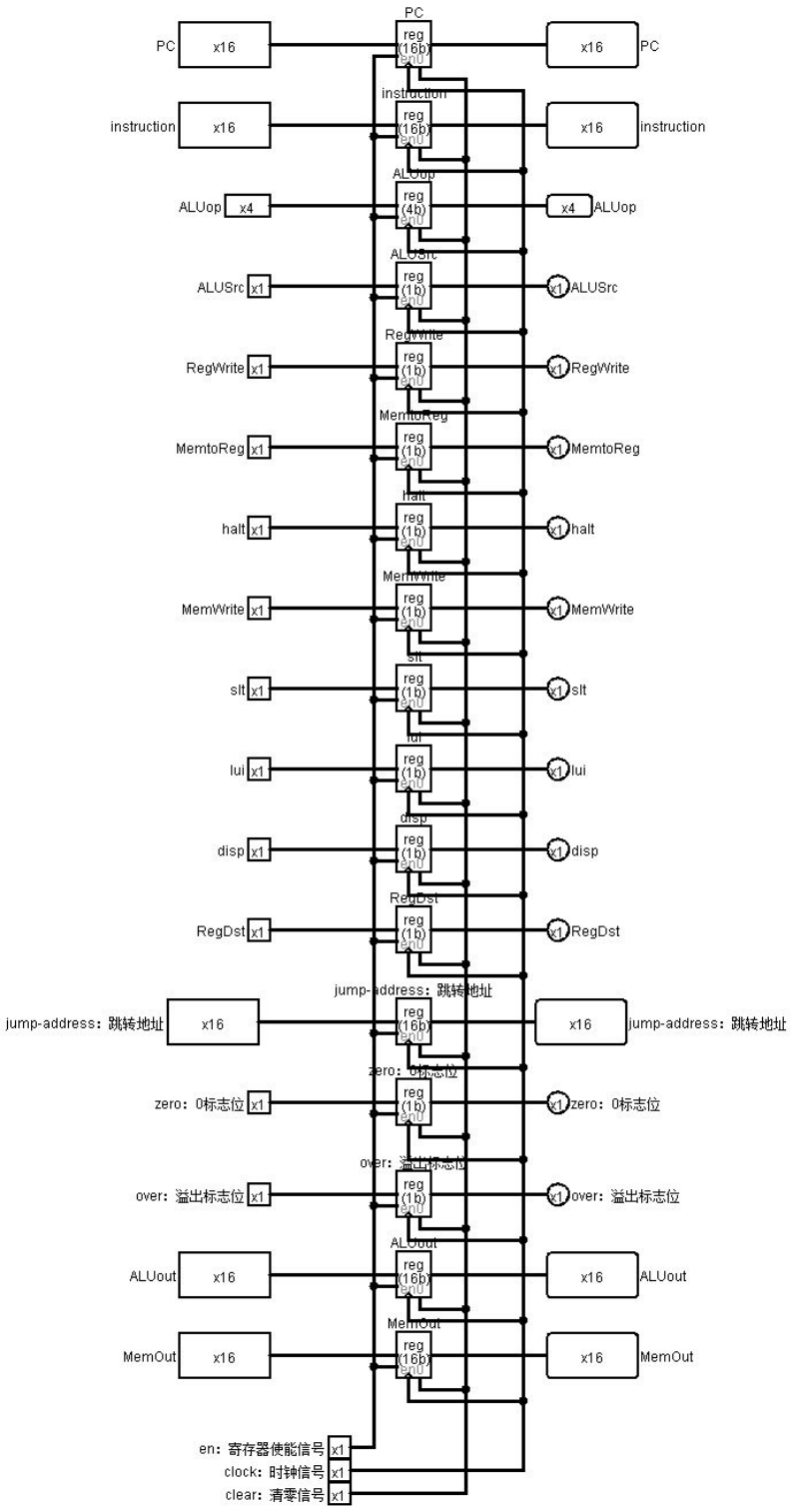


图 4. 26 MEM/WB 实现

5. 理想流水电路实现

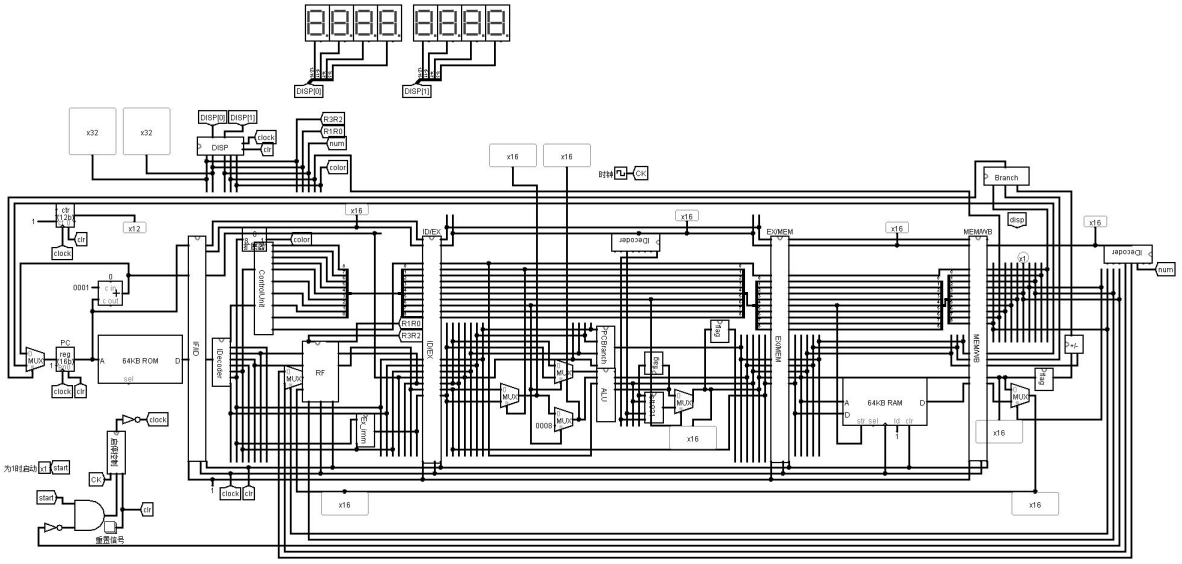


图 4. 27 理想流水完整电路图

4.3流水线冲突检测器

冲突有两种方式，一个是数据冲突，一个是控制冲突，因为该 CPU 的数据和指令分离，因此不会产生结构冲突。

数据冲突：

当前指令要写回的寄存器与下一条指令的要用到的寄存器有交集，造成冲突。

控制冲突：

需要跳转的情况如果使用理想流水回出问题。

因此，冲突检测器应该是这样一个函数模型。

输入：当前指令，下一条指令

输出：是否冲突。

详细情况

数据冲突：

- 指令限制
  - 当前指令：{R 型指令（OP=1），lui，ori，andi，addi，lw}（需要将值写会寄存器），但是当指令为 0x0000 的时候
  - 下一条指令：{R 型指令（OP=1），beq，bne，bgt，ori，andi，addi，lw，sw}（需要寄存器参与运算）

当指令满足上述条件时，才有可能产生冲突。

然后在此基础上，在详细判断上一条指令要写回的寄存器是否与下一条指令用到的寄存器是否重合。

在上述条件满足条件下

当当前指令的 OP 为 0000 的时候，写回的寄存器为$rd,否则写回寄存器的值为$rt,

当下一条指令的 OP∈{0,1,2,3,10}的时候,指令同时的寄存器为$rs,$rt.当 OP∈{6,7,8,9}的时候，指令用到的寄存器只是$rs.

分支冲突：

- 指令限制：
  - 当前指令：{beq，bne，bgt，jump}

根据上述的分析：

得到的冲突分析电路如下图所示：

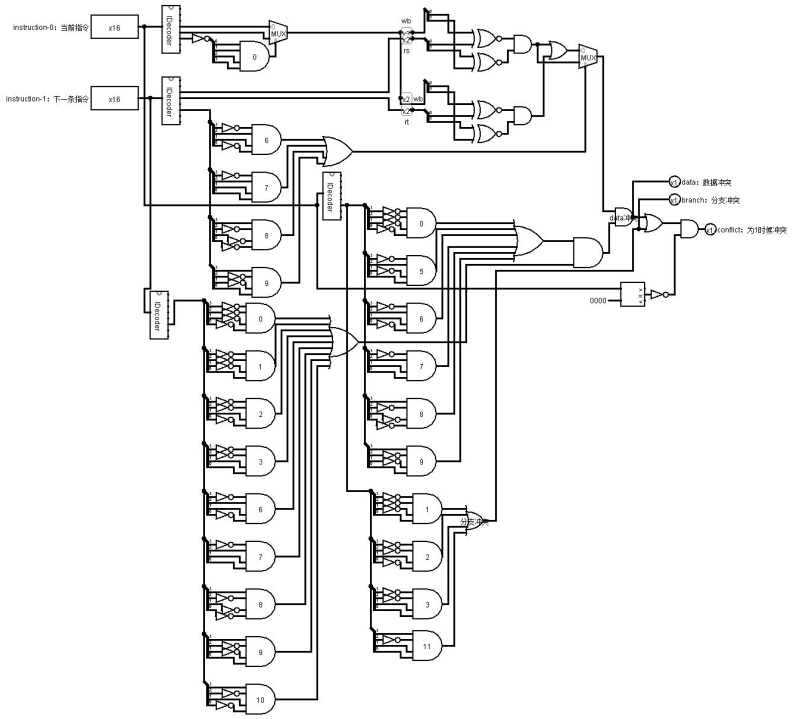


图 4. 28 两指令冲突检测

因为当前指令不仅可能和前一条指令冲突，还有可能和之前的几条指令相冲突，因此要一起检测。因此下一条进入电路的指令需要和 IF,EX,MEM 端的指令比较，得到是否冲突。若产生冲突，则插入气泡。

实现后的检测器如下图所示。

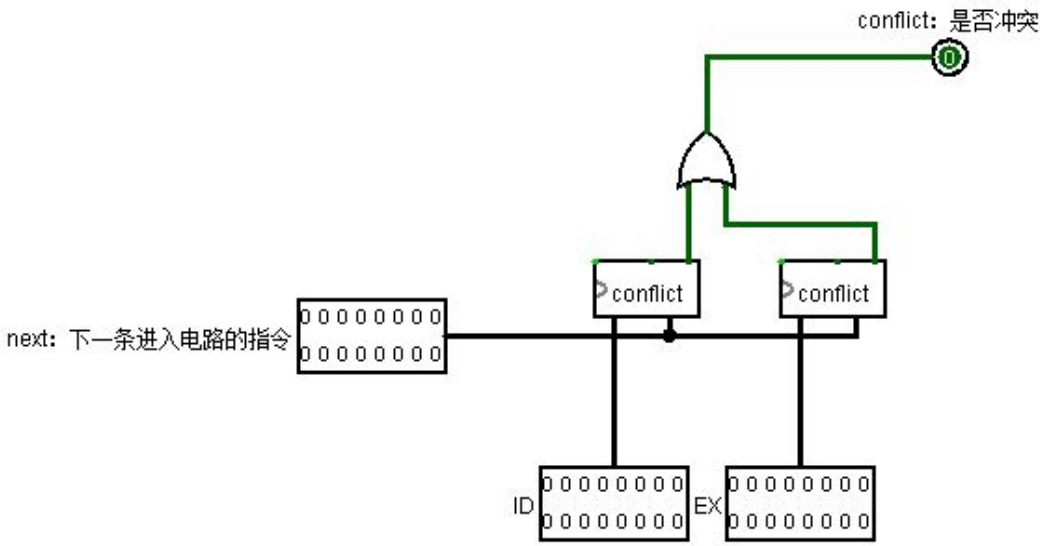


图 4. 29 三指令冲突检测

4.4 插入气泡的流水冲突处理

在检测到冲突之后就可以进行气泡插入处理了，插入气泡需要以下两个处理。

3. 向前阻塞

向前提供阻塞信号，使得 PC 暂停取下一条指令，这个可以用冲突检测器的输出信号取反后接入到 PC 的使能端。

4. 向后插入气泡

向后提插入气泡，插入的气泡必须与所有指令都没有冲突的指令。

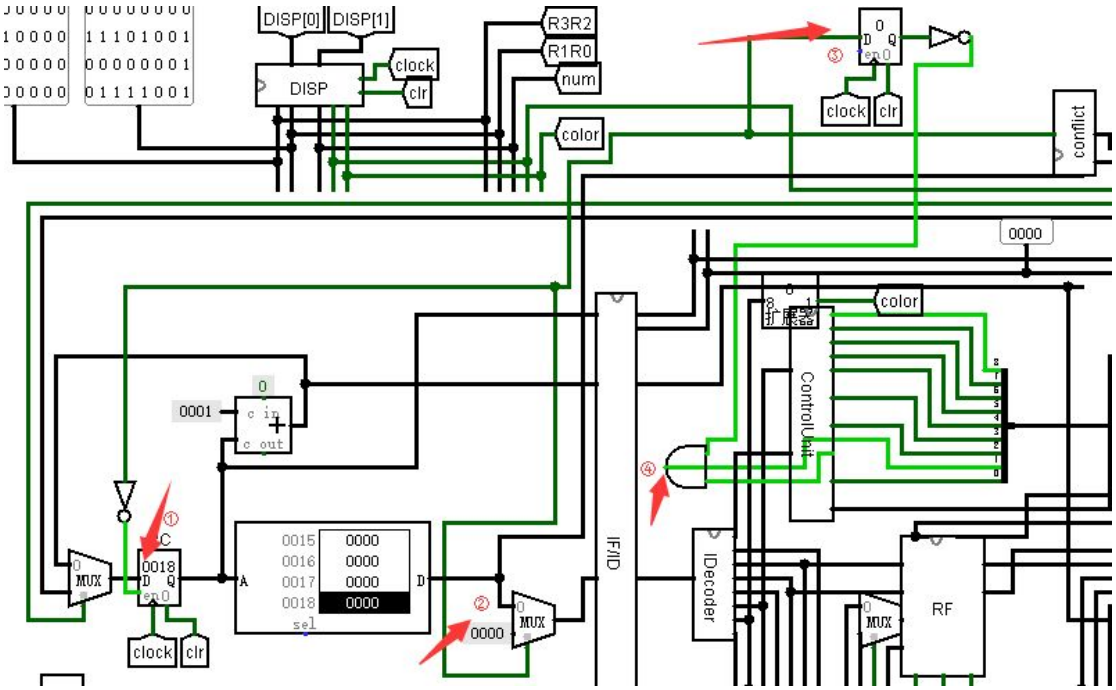
可以考虑的指令为：

1) 0x0000

气泡插入方法：

向前阻塞之后，将要插入的气泡输入到 IF/ID 的输入端。这与 ROM 的输出冲突，故做一个数据选择器，以冲突检测器的输出信号作为选择信号。另外，需要将 0x0000 的 RegWrite 控制信号置为 0.

实现后的效果如图所示：



如上图所示：

①：冲突信号取反后接到 PC 的使能端，用于暂停 PC

②：冲突后插入气泡，气泡指令为 0x0000

③：因为过半个时钟周期后气泡才进入 ID 段，所以用寄存器缓存冲突信号，使得其与气泡同步到达 ID 段。

④：冲突信号取反后与 RegWrite 信号相与，得到的结果传到下一级。这样，插入气泡之后，气泡对应的 RegWrite 信号就被置 0 了。达到真正的气泡效果。

但是上图的冲突处理未能很好的处理分支存在的情况，如果有条件跳转，而且假设最后不跳转，理想的指令流应该是检测数据冲突，进行处理之后在进入电路，因为不跳转，所以照常流下去。而如果需要跳转，则跟着进入电路的跳转指令的后面部分都需要清零，相当于插入两个气泡，电路在 EX 期间执行跳转，因此需要将接口 IF/ID，ID/EX 都清零，上图能实现 IF/ID 清零，而且接口的清零信号指令进行异步清零。故上图不能满足需求。需要对接口部件进行修改，使之能够进行同步清零。这里用常量加二路选择器实现。

实现的结果如下图所示：

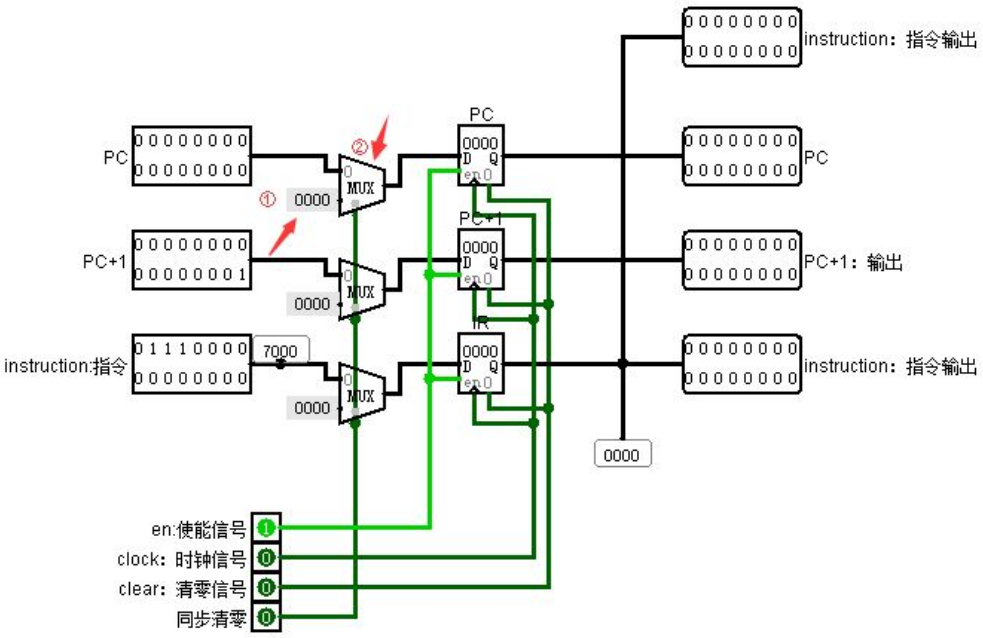


图 4. 30 同步清零接口

其它接口跟这个接口类似。

所以当跳转信号有效的时候需要将 IF/ID,ID/EX 清零，用改进后的接口实现的冲突处理电路为：

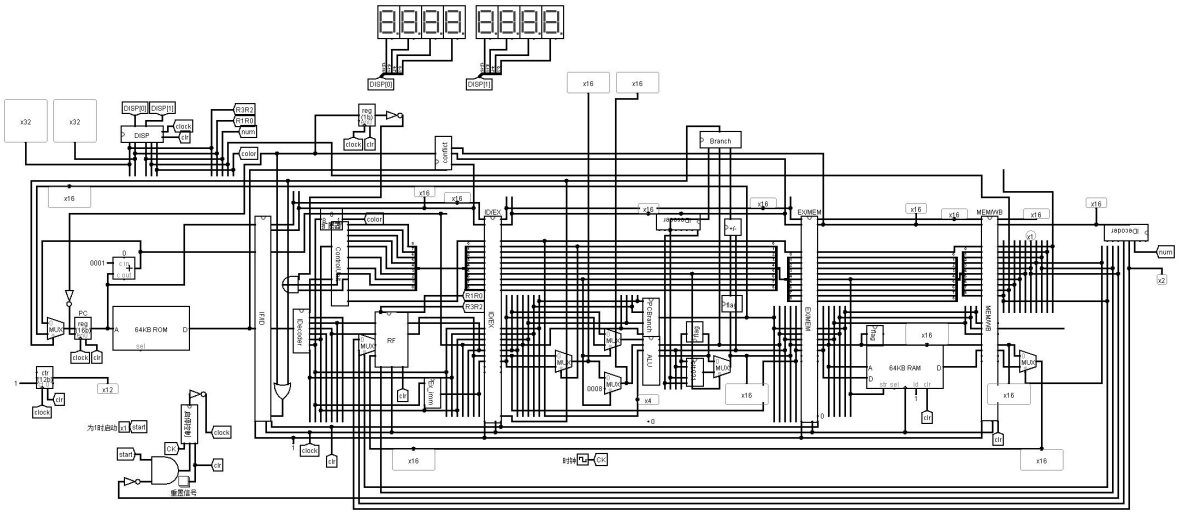


图 4. 31 气泡冲突处理的流水线

可以看到，跳转信号能够使 IF/ID,ID/EX 清零，同时冲突信号也可以对 IF/ID 清零，因此在 IF/ID 的同步清零信号端的两个清零信号进行了或操作。

4.5 数据重定向的流水冲突处理

4.5.1 运算输入的重定向

根据之前的输入输出模型，我们的 TO_A 只能是 MEM-MEMout，MME-ALUout，WB-in，original-a 中的一个，TO_B 只能是 MEM-MEMout，MME-ALUout，WB-in，original-b 中的一个。因此可以做两个数据选择器，一个选择数据输出到 TO_A，一个选择数据输出到 TO_B，现在关键是怎样确定选择器的选择信号，设 TO_A 的选择信号为 choose-a，TO_B 的选择信号为 choose-b。

现在将重心放在如何确定 choose-a 和 choose-b 上。

1) choose-a:

choose-a 的值与选择的数据的对应关系如下：

数据来源	choose-a（二进制）
original-a	00
MEM-ALUout	01
MEM-MEMout	10
WB-in	11

图 4. 32 choose-a 输入输出表

首先定义 choose-a 的输入输出模型：

输入	说明	输出	说明
ex-mem-conf	EX 指令和 MEM 指令是否冲突	choose-a	TO_A 的选择信号
ex-wb-conf	EX 指令和 WB 指令是否冲突		
op-9	MEM 阶段指令的 OP 是否为 9		
MEM-wb==WB-wb	MEM 的写会寄存器与 WB 的写回寄存器是否相等		
MEM-wb==rs	MEM 的写回寄存器与 EX 阶段指令的 rs 相等		
WB--wb == rs	WB 的写回寄存器与 EX 阶段指令的 rs 相等		

图 4. 33 choose-a 输入输出模型

在一下情况下，TO_A 应该为 MEM-ALU-out：

EX 冲突 MEM	EX 冲突 WB	MEM_OP ==9	MEM _wb== =WB_ wb	MEM-wb == rs	WB-wb ==rs	选择	Choose-A
1	0	0	x	1	x	MEM-alu-out	01
1	1	0	1	1	x		
1	1	0	0	1	0	MEM-alu-out	

表 4. 7 choose-a==01

注：已经考虑到 EX 与 MEM，WB 同时冲突，且 MEM-wb==WB-wb 的情况，此时，选择 MEM 阶段的数据。因为此时的数据是最新的。

同理可以得到 choose-a 的其他情况，这里略去。

连接后的电路图为：

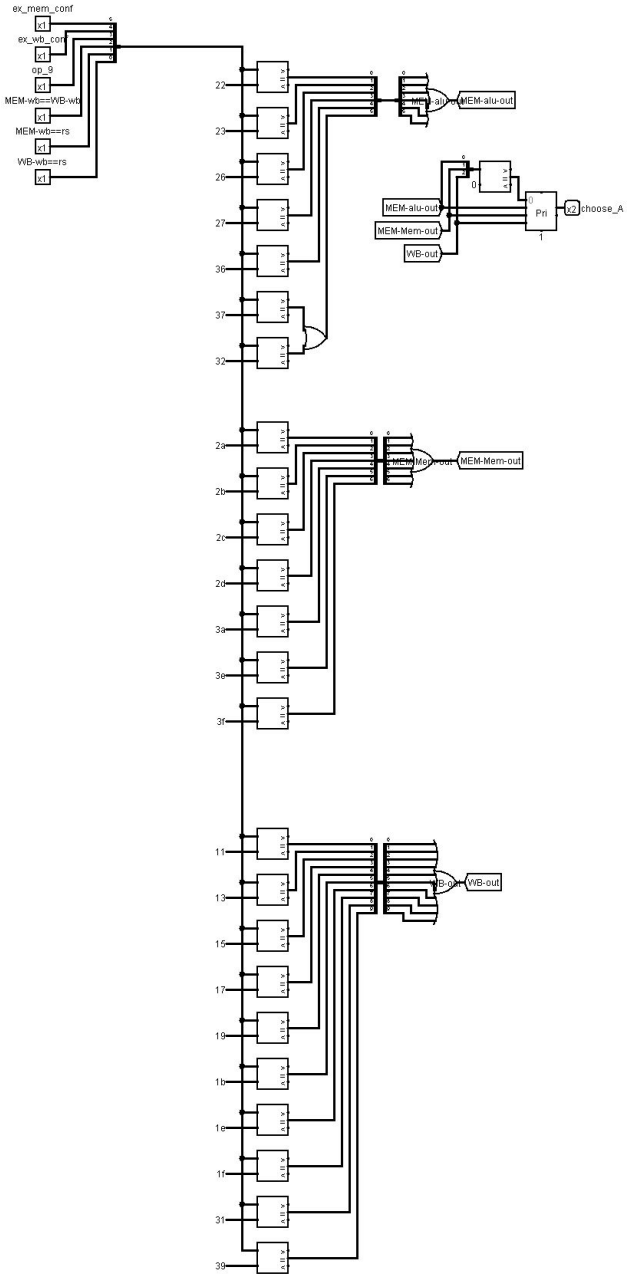


图 4. 34 choose-a

2) choose-b:

基本上和 choose-a 一样，唯一不同的地方是此时指令 EX 必须含有 rt。

因此可以在 choose-a 的基础上实现 choose-b，choose-b 就不用封装成子电路了。得到的电路图如下图所示：

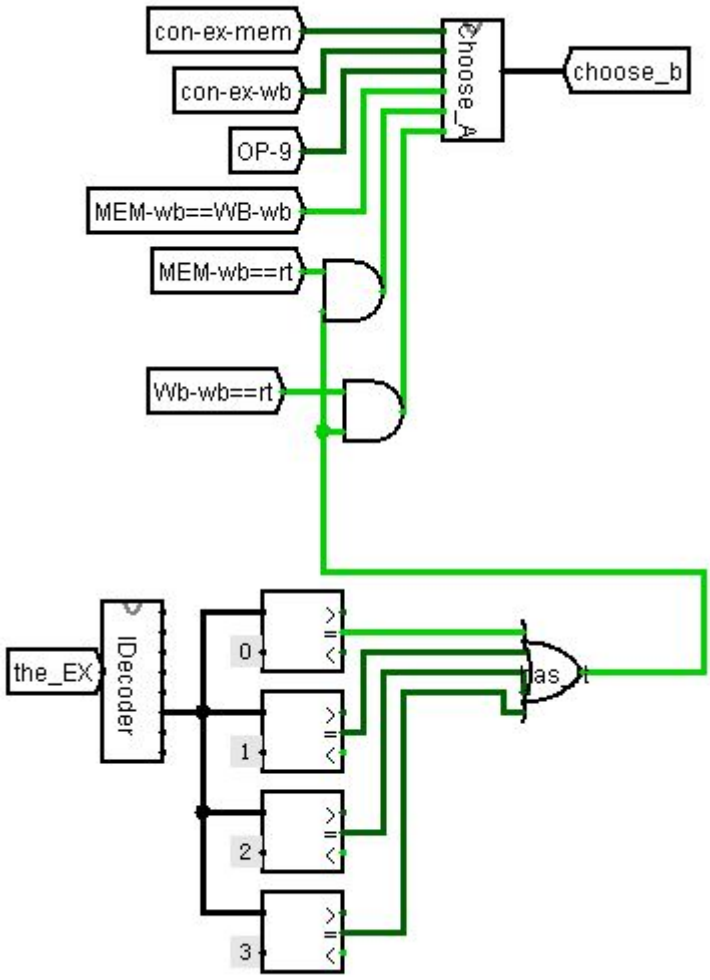


图 4. 35 choose-b

4.5.2 lw 指令输入的重定向

当 lw 的 rt 与前面不远的指令的写会的寄存器编号相等时，回产生数据冲突。

lw 需要一个电路来判断是否冲突，根据指令的设计容易的到该冲突检测电路的实现的电路图为：

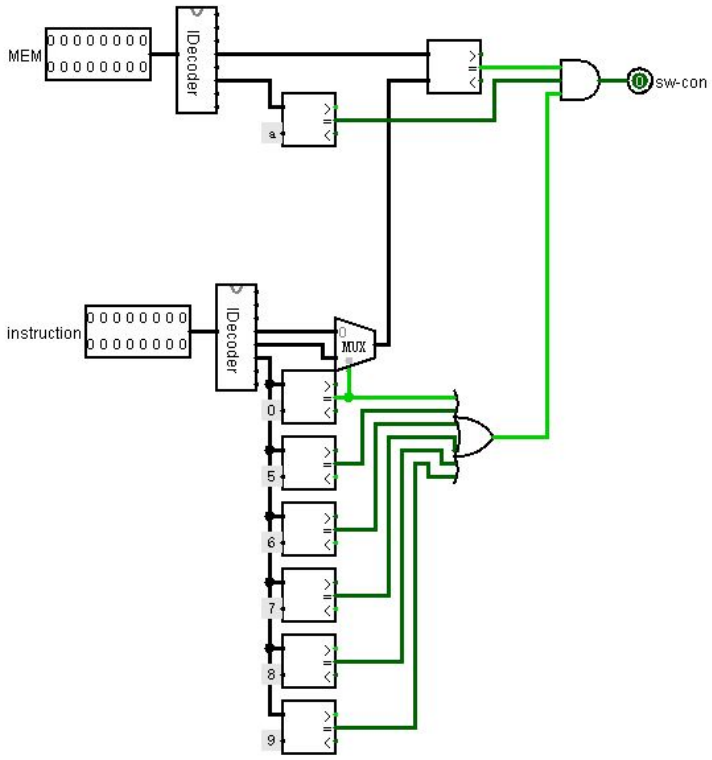


图 4. 36 lw 指令冲突检测电路

因为 MEM 的前一条指令为 WB 阶段的指令，而 MEM 有可能和 WB 的前一条以及 WB 的前前一条指令相冲突，所以对总电路做出修改，使之能够保存这些阶段的指令。在 WB 指令之后将两个寄存器就可以简单实现。

如图所示：

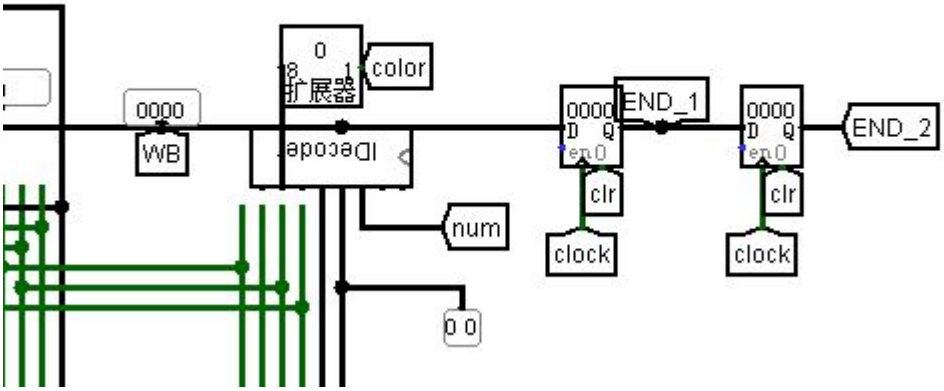


图 4. 37 指令活久一点

根据之前的 lw 重定向单元的输入输出表容易得到其电路实现：

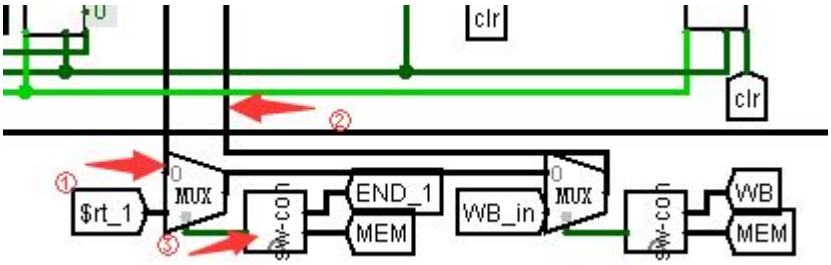


图 4. 38 lw 重定向单元

其中，①为 original-d ②为 TO_D ③为冲突检测单元的封装结果。

得到的数据重定向的完整电路图为：

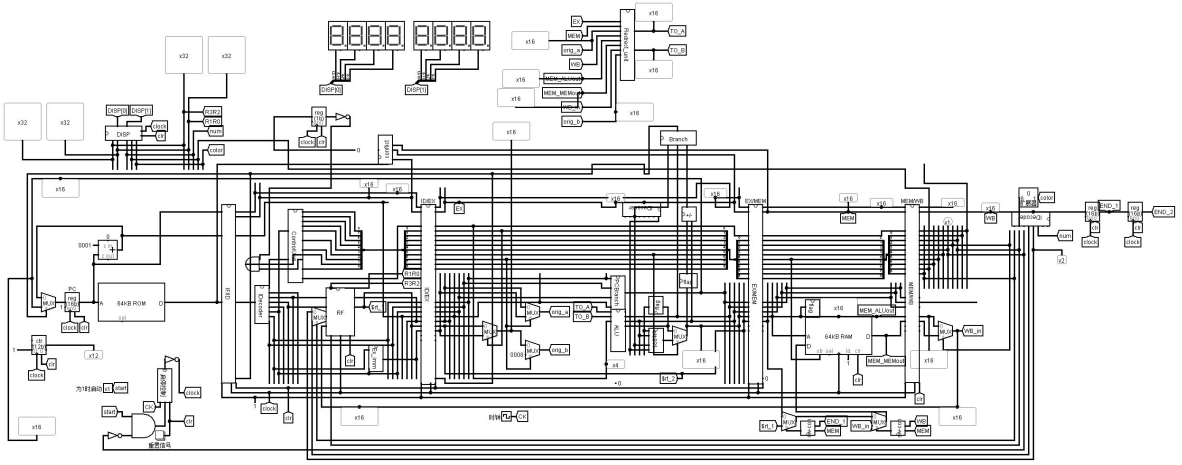


图 4. 39　数据重定向

5 实验过程与调试

5.1 测试用例和功能测试

5.1.1 测试用例 1 单周期

测试数据：benchmark（RAM 加载 benchmark）

周期数：1293（正确）

```
0000  7f00 7500 7000 7000  4c01 4c01 4400 4400  1801 0dc2 0487 0444  b008 8f01 8501 8001
0010  8001 7500 850f 0443  0442 01c4 4c00 7500  8503 0dc6 4c00 7500  8504 0dc6 4c00 0dc6
```

图 4. 40 单周期-RAM

5.1.2 测试用例 3 气泡

测试数据：benchmark（RAM 加载 benchmark）

周期数：2729（正确）

```
0000  7f00 7500 7000 7000  4c01 4c01 4400 4400  1801 0dc2 0487 0444  b008 8f01 8501 8001
0010  8001 7500 850f 0443  0442 01c4 4c00 7500  8503 0dc6 4c00 7500  8504 0dc6 4c00 0dc6
```

图 4. 41 气泡-RAM 结果

5.1.3 测试用例 1 数据重定向

测试数据：benchmark（RAM 加载 benchmark）

周期数：1803（正确）

RAM 结果：

```
0000  7f00 7500 7000 7000  4c01 4c01 4400 4400  1801 0dc2 0487 0444  b008 8f01 8501 8001
0010  8001 7500 850f 0443  0442 01c4 4c00 7500  8503 0dc6 4c00 7500  8504 0dc6 4c00 0dc6
```

图 4. 42 数据重定向-RAM 结果

5.2 可自行安排章节

5.3 性能分析

分析不同方案时钟周期数差异

5.4 主要故障与调试

5.4.1 故障 1

走马灯测试：

问题：最后一条指令 bne $r0, $r2, Loop 的时候不能跳回前面的指令位置。

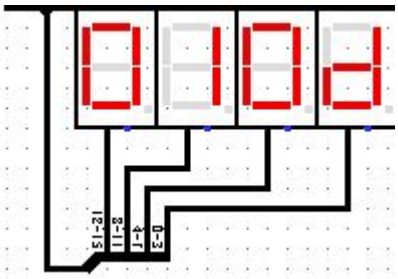


图 4. 43 跳转地址

如图所示，跳转地址形成部件形成的地址显然不符合情况。

进入跳转地址形成部件查看详细情况。

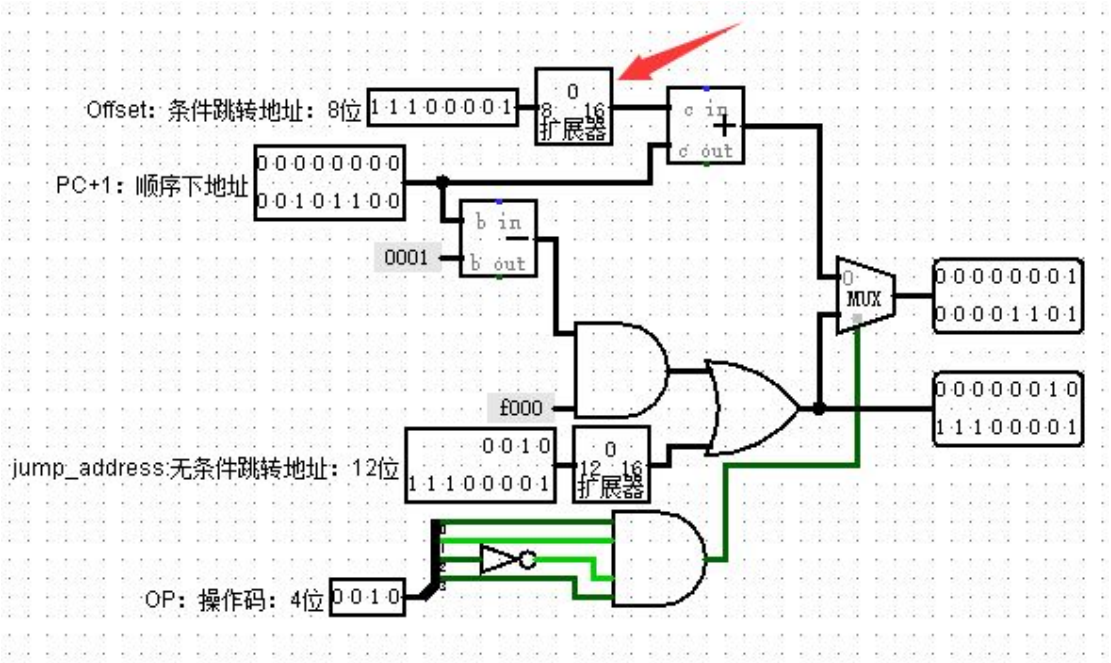


图 4. 44 跳转地址 有问题

可以看出，加法器应该还能进行减法，但是对 offset 进行了 0 扩展，因此只能进行加法操作，因此在要进行向前跳转的时候 offset 应该进行 1 扩展，此处没有，因此要在电路中添加这种情况。

修改如下：

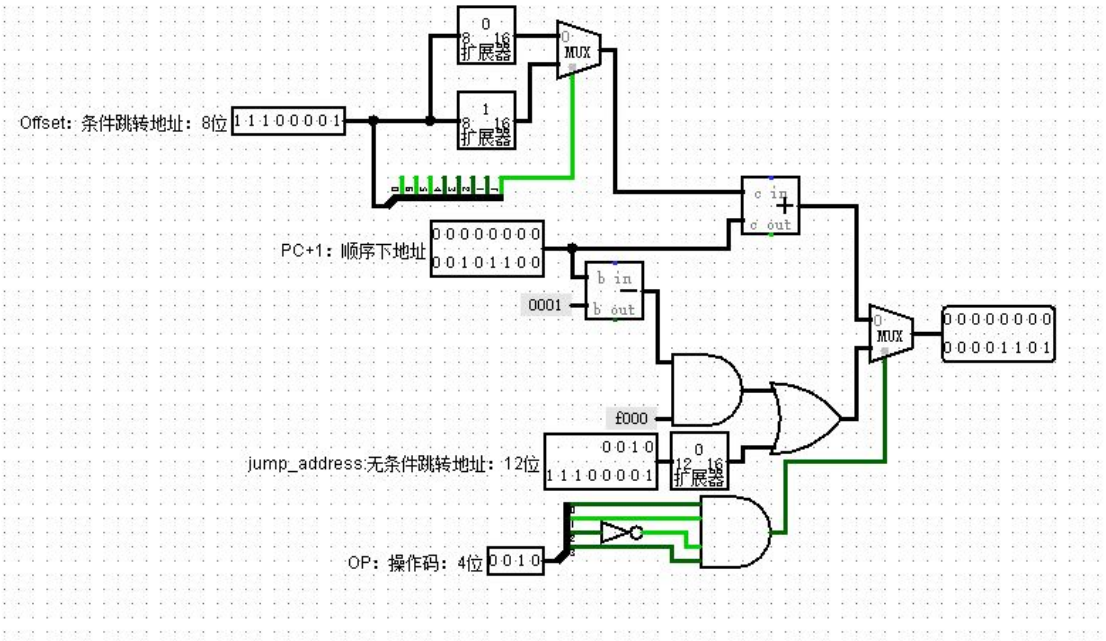


图 4. 45 跳转地址

可以看出，输出结果为 1101，的确是前面的地址，应该跳转到 Loop:addi $r0, $r0, 1，也就是说指令的 16 进制应该为 8001.

跳转的结果如图所示：

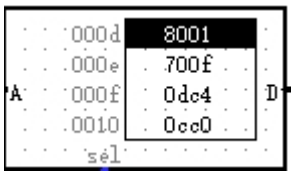


图 4. 46 跳转

跳转正确。

再测试一切正常。

5.4.2 故障 2

移位测试：

周期数 T = 140,和理想的情况不符(133)。该问题待解决。

找到问题：在执行到 110 步的时候，对应的指令是 slt。

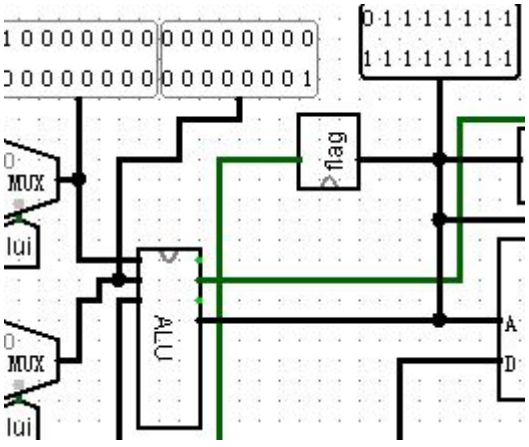


图 4. 47 110 步时运算器执行结果

两个操作数相减，操作数 A 是负数，操作数 B 是负数，得到的结果应该是负数，但是确得到一个正数，说明发生了溢出，此时用结果的符号位来作为两者大小的判断

依据显然不合理。因此关键部分是在 slt 指令的实现上。需要对 slt 做出修改。

修改思考：

slt 指令需要将 ALU 运算结果的符号位，溢出信号一起考虑。

因此得到的模型为

输入：

ALUout-flag：ALU 符号位

overflow：溢出信号

输出：1,或 0

当不发生溢出的时候，也就是说 overflow 为 0 的时候，以 flag 作为输出。

当溢出，也就是说 overflow 为 1 时，有两种情况

1. 正数减负数 flag = 1

2. 负数减正数 flag = 0

因此此时输出为 flag 取反。

追踪之后，发现减法器不能正确溢出，需要改进减法器。使之能够正确溢出。

修改之后得到的结果正确。

5.4.3 故障 3

排序测试：

排序结果混乱：

思考：根据上一个 slt 的修改可以直达，问题出在 bgt，也就是说我的跳转模块上。之前用输出结果的符号位来作为比较的依据，不能处理溢出的情况，因此需要新建一个模块使之能够正确判断 ALU 运算结果的理想的正负性。此外，ALU 加法也没有正确溢出，为避免不必要的麻烦，修改 ALU，使之能够正确溢出。

得到的判断正负性的模块的电路为：

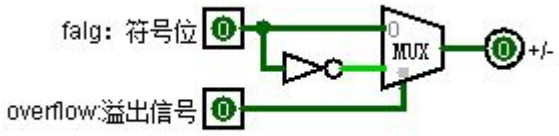


图 4. 48 判断运算结果的理想正负性

修改之后，能够正确排序。

5.5 实验流程

表 5.1 课程设计进度表

时间	进度
第一天	运动会
第二天	复习了组成原理教材，着重看了微程序设计和硬布线设计。仔细看了课设任务书。仔细安排了后续单周期 CPU 的设计实验工作。
第三天	上午：实现运算器，启停控制器，寄存器堆 GR。下午：实现指令译码器，ALUop 译码器，加法数据通路。晚上：实现跳转地址形成部件，单周期 CPU 整体数据通路总框架，未实现控制器。
第四天	上午：实现 RegWrite 控制部件，RegDst 控制部件，立即数符号扩展。下午：实现 ALUSrc 控制部件，MemWrite 控制部件，MemtoReg 控制部件。晚上：实现 DISP，修改 Branch 控制部件，halt 控制部件。
第五天	早上：测试 B 指令通过，下午：测试 JMP 指令，数据相关测试通过。实现 slt，晚上：修复 DSIP 的问题，改进寄存器堆 RF
第六天	调试完单周期，问题出在 ALU 没有正确溢出。开始画理想流水电路。
第七天	画完理想流水，问题很多，正在调试。
第八天	调通理想流水，画好冲突检测模块，正在实现插入气泡处理。
第九天	完成气泡
第十五天	完成写好数据重定向单元
第十六天	完成数据重定向

6 设计总结与心得

6.1 课设总结

基于对象的存储是为了克服当前基于块的存储存在的诸多难题，在存储接口和结构层次的重要发展。可以根据应用负载选择优化的存储策略。作了如下几点工作：

1） 完成方案总结

对于单周期部分的方案设计来说，整体设计主要是参考任务书的模型进行设计。最总要的部分是控制器的各个控制信号的设计。在这方面花了不少时间。

对于理想流水部分来说，接口部件的内容传递是很有挑战性的一部分。接口在调试的时候需要逐步对接口进行扩展，这对连线影响很大。

对于气泡解决冲突来说，最总要的部分是处理检测，冲突检测并不困难。

对于数据重定向来说，关键的地方是确定重定向数据来源。

2） 功能总结

单周期：完成给定功能，没有另外有趣的设计。

气泡解决冲突：完成给定功能，没有另外有趣的设计。

数据重定向解决冲突：完成给定的功能，没有魔幻的设计。

6.2 课设心得

1） 这次课设让我知道了什么叫磨刀不误砍柴工，我实现将教材 CPU 部分复习了一遍，之后在进行电路设计的时候能够做到胸有成竹。

2） 课设让我以超直接的方式了解的计算机的工作原理。当自己的 CPU 跑起来的时候，第一件想到的是平时看起来神秘无比的计算机，不过如此。

3） 我感觉 logisim 这个实验平台有很多让人不爽的地方，很多时候出错不是自己出错，而是 logisim 出错了，希望能有一个更好的 EDA 平台。

参考文献

[1] DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第 4 版).北京：机械工业出版社.

[2] David Money Harris(美).数字设计和计算机体系结构（第二版）. 机械工业出版社

[3] 秦磊华，吴非，莫正坤.计算机组成原理. 北京：清华大学出版社，2011 年.

[4] 袁春风编著. 计算机组成与系统结构. 北京：清华大学出版社，2011 年.

[5] 张晨曦，王志英. 计算机系统结构. 高等教育出版社，2008 年.

[6] 张志刚，FPGA 与 SOPC 设计教程-DE2 实践. 西安：电子科技大学出版社，2007