计算机组成原理

课内大作业报告

学　　号_____18021006_____

姓　　名_____吴天鹤_____

指导教师_____朱文军_____

提交日期_____2020.5.17_____

成绩评价表

报告内容	报告结构	报告最终成绩
□丰富正确 □基本正确 □有一些问题 □问题很大	□完全符合要求 □基本符合要求 □有比较多的缺陷 □完全不符合要求	
报告与 Project 功能一致性	报告图表	总体评价
□完全一致 □基本一致 □基本不一致	□符合规范 □基本符合规范 □有一些错误 □完全不正确	

教师签字:_____

目录

1. 总体数据通路结构设计图

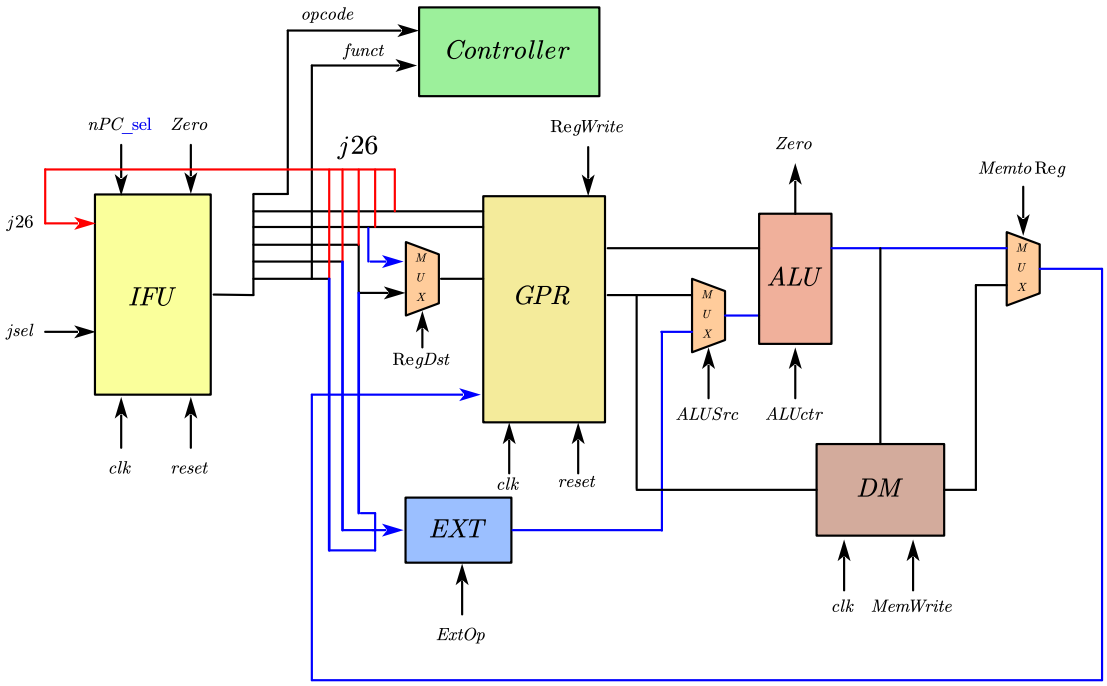

图 1 总体数据通路结构设计图

其中 **lui** 指令是走蓝色线路，**j** 指令是走红色线路

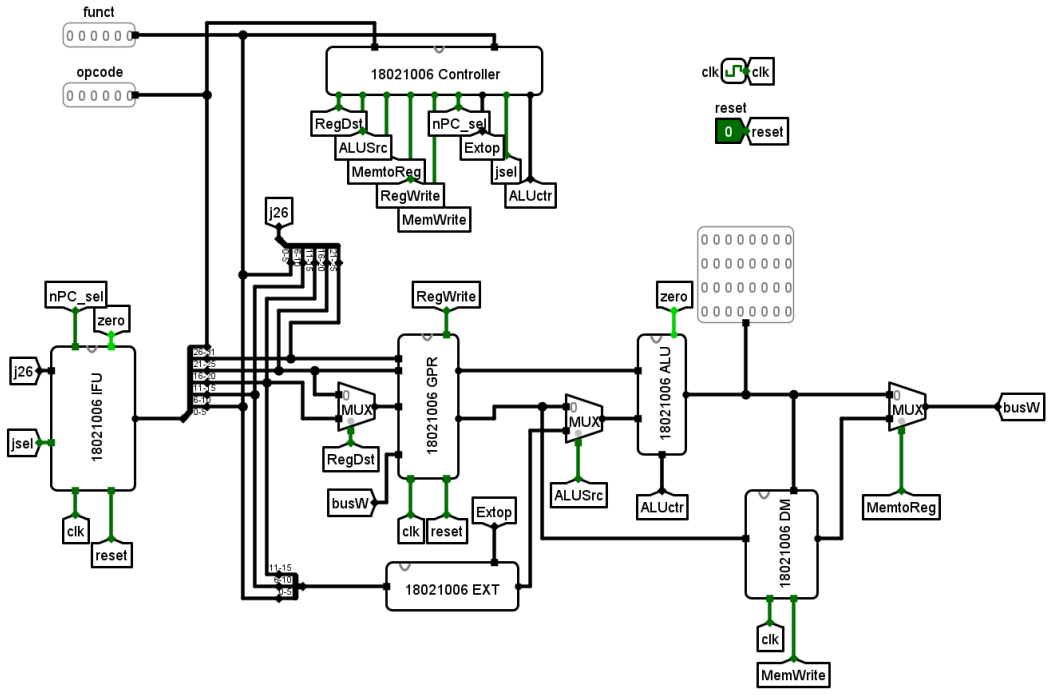

图 2 Logisim 主电路截图

2. 各个模块定义

2.1 IFU 模块定义

(1)基本描述

IFU 主要功能是完成取指令功能。IFU 内部包括 PC、IM(指令存储器)以及其他相关逻辑。IFU 除了能执行顺序取指令外，还能根据 BEQ 指令以及 J 指令的执行情况决定顺序取指令还是转移取指令。

(2)模块接口

信号名	方向	描述
npc_sel	I	当前指令是否为 beq 指令标志。 1：当前指令为 beq 0：当前指令非 beq
zero	I	ALU 计算结果为 0 标志。 1：计算结果为 0 0：计算结果非 0
clk	I	时钟信号
reset	I	复位信号。 1：复位 0：无效
jsel	I	1: 当前指令为 J 指令 0：当前指令非 J 指令
J26	I	26 位 target address
ins[31:0]	O	32 位 MIPS 指令

(3)功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，PC 被设置为 0x00000000。
2	取指令	根据 PC 从 IM 中取出指令。
3	计算下一条指令地址	如果当前指令不是 beq 或 J 指令，则 PC←PC+4 如果当前指令是 beq 指令，并且 zero 为 0，则 PC←PC+4 如果当前指令是 beq 指令，并且 zero 为 1，则 PC←PC+4+(sign_ext(ins[15:0])<<2) 如果当前指令是 J 指令，则 PC = {(PC+4)[31:28], target address, 00}

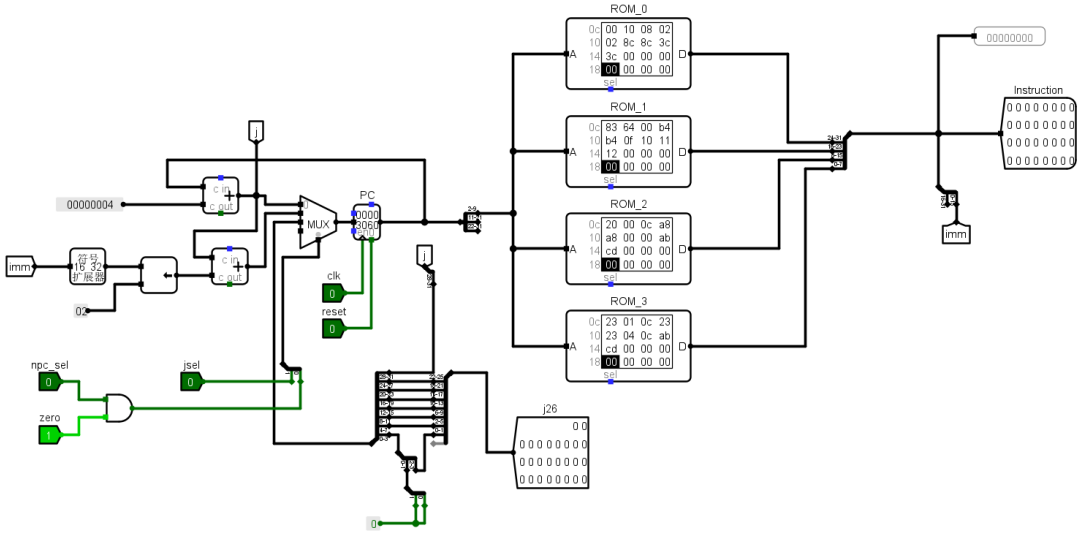

图 3 IFU Logisim 电路截图

2.2 GPR 模块定义

(1) 基本描述

GPR 是通用寄存器堆，里面有 32 个寄存器用来存储值，还有相应的 DXM 进行 rt 和 rd 寄存器的选取。主要功能是向特定的寄存器内写入具体的值，从特定的寄存器中取出值，传递给 ALU，进行相应的指令功能的操作。

(2) 模块接口

信号名	方向	描述
ra	I	接收当前指令的 25-21 位，选择对应的寄存器，取出对应的值
rb	I	接收当前指令的 20-16 位，选择对应的寄存器，取出对应的值
rw	I	选择对应的寄存器，作为结果存储的寄存器
clk	I	时钟信号
reset	I	复位信号。 1：复位 0：无效
busW	I	接收通过 ALU 之后的值存入由 rw 决定的寄存器中
busA	O	输出对应的寄存器内部的值
busB	O	输出对应的寄存器内部的值

(3) 功能定义

序号	功能名称	功能描述
1	取值	根据 rs, rt 从特定寄存器取值
2	存值	根据 rt 或 rd 向特定寄存器写入值

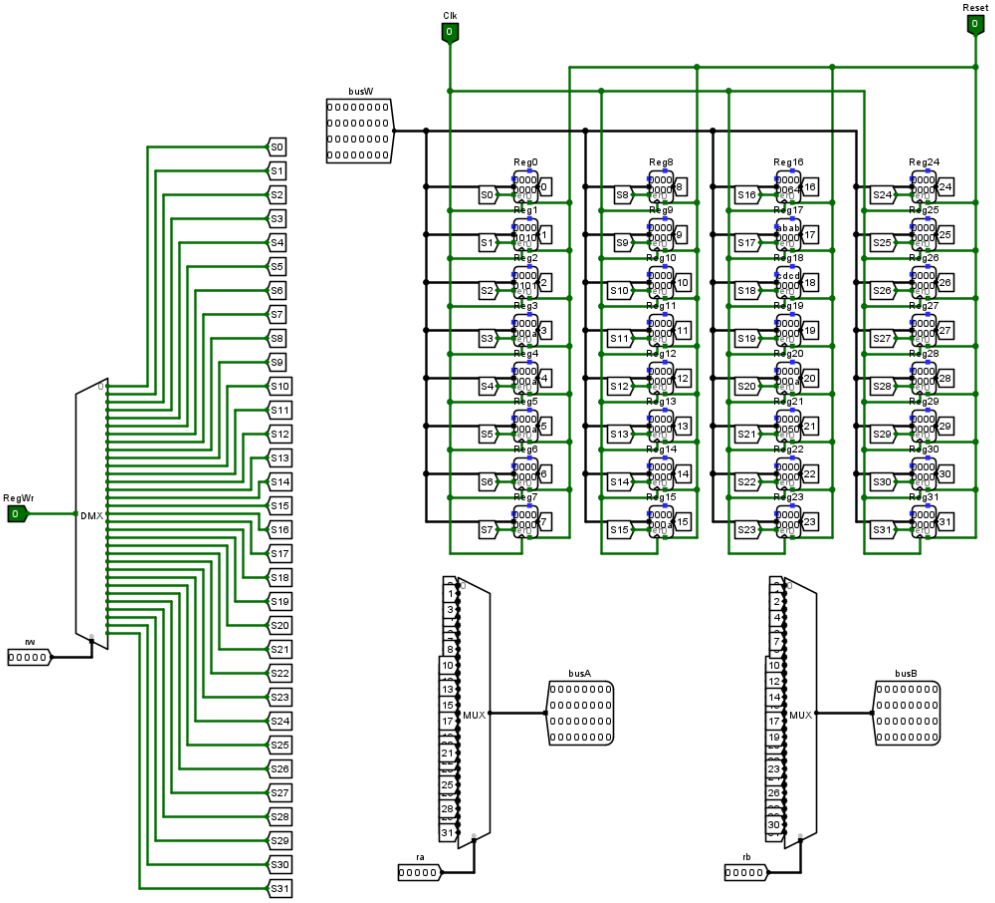

图 4 GPR Logisim 电路截图

2.3 ALU 模块定义

(1) 基本描述

ALU 是算术逻辑单元，在 ALU 中可以实现加、减，或运算。对应 addu, subu, ori 指令，同时能判定相减后是否为 0，如果为 0 且指令为 beq 指令，则根据 beq 后的指令进行跳转。

(2) 模块接口

信号名	方向	描述
busA	I	输入第一个值 busA
busB	I	输入第二个值 busB
ALUctr	I	选择信号，对应的只输出 busB(lui 指令)，输出两个值相加，输出两个数相减，输出与立即数或运算
zero	O	判断两数相减是否为 0
out	O	输出运算后的结果

(3) 功能定义

序号	功能名称	功能描述
1	相加	实现传入两数相加（要经过 ALUSrc 选择）
2	相减	实现传入两数相减（要经过 ALUSrc 选择）
3	与立即数或	实现与 32 位立即数相或（要经过 ALUSrc 选择）

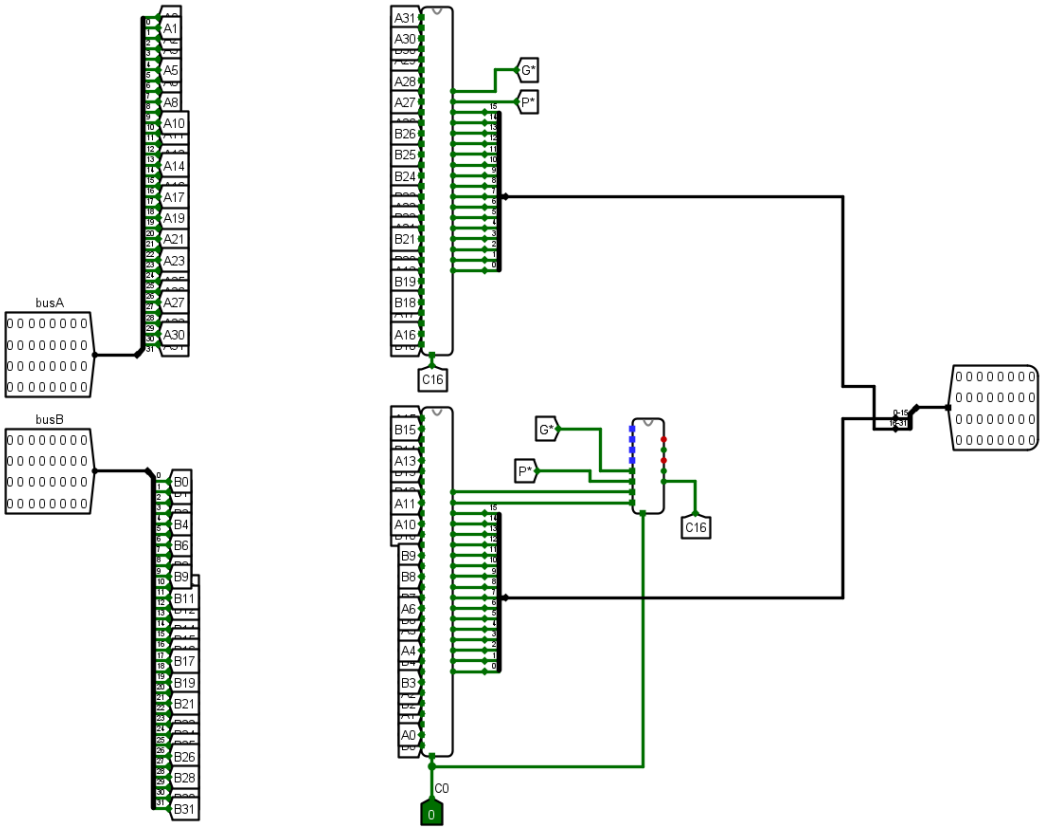

图 5 32bit_paraller_adder Logisim 电路截图

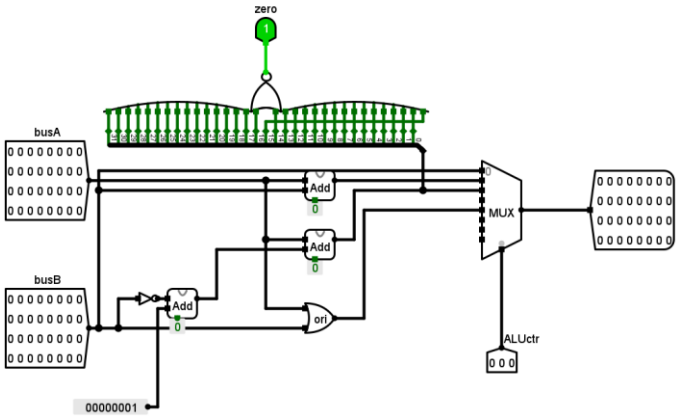

图 6 ALU Logisim 电路截图

2.4 EXT 模块定义

(1) 基本描述

EXT 是扩展模块。该模块里能够实现 16 位到 32 位的 0 扩展，符号扩展，还有低 16 位 0 的扩展，高位为该立即数的功能。

(2) 模块接口

信号名	方向	描述
Imm16	I	输入 16 位立即数
Extop	I	选择信号，选择进行高位 0 扩展，符号扩展，低 16 位 0 扩展高 16 位为立即数即 lui 指令
Output32	O	输出扩展后的结果

(3) 功能定义

序号	功能名称	功能描述
1	高位 0 扩展	将 16 位立即数进行高位 0 扩展
2	符号扩展	将 16 位立即数进行符号扩展
3	高位为立即数，低 16 位为 0	lui 指令操作

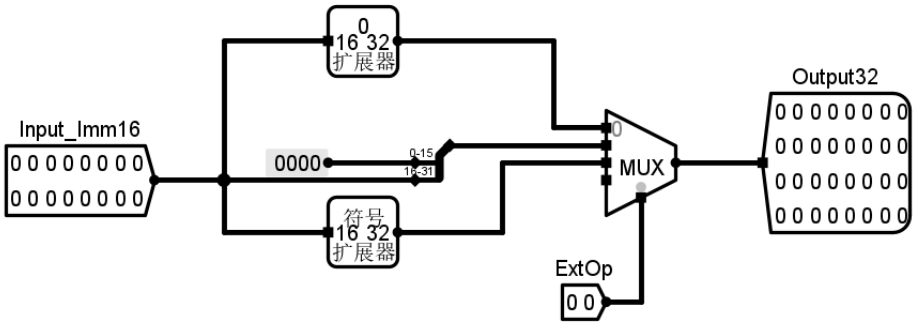

图 7 EXT Logisim 电路截图

2.5 DM 模块定义

(1) 基本描述

DM 是数据存储器模块。该模块是向具体的数据段地址存储相应的数据，和从相应的地址取出数值。

(2) 模块接口

信号名	方向	描述
busB	I	输入 rt 对应寄存器的值
busA	I	输入对应的地址
Out	O	输出从寄存器取出的值
clk	I	时钟信号
MemWrite	I	存储使能信号

(3) 功能定义

序号	功能名称	功能描述
1	输入地址	选择数据存储器具体的地址
2	存储数据	向相应的地址里写入数据
3	取出数据	从相应的地址里取出数据

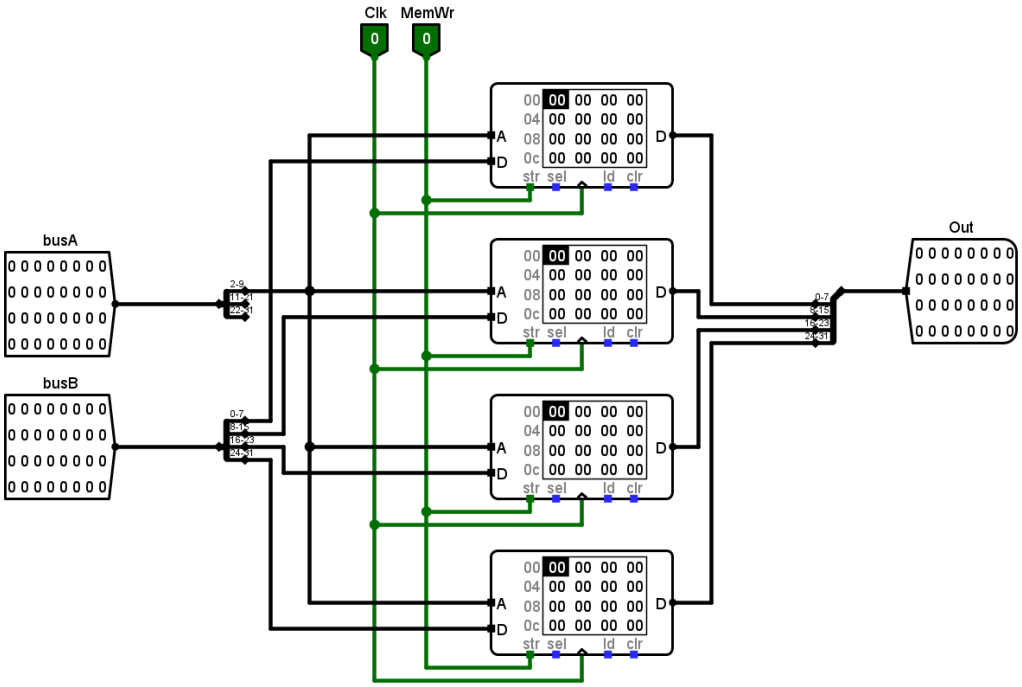

图 8 DM Logisim 电路截图

2.6 Controller 模块定义

(1) 基本描述

Controller 是控制器模块。该模块是控制数据通路里选择信号的一个模块。通过对 opcode 和 funct 进行与或操作来实现一条指令输入，哪些选择信号打开的功能。

(2) 模块接口

信号名	方向	描述
RegDst	O	对 rt 和 rd 寄存器进行选择
ALUSrc	O	选择 busB 和 16 位立即数扩展后的值
MemtoReg	O	选择 DM 和 ALU 计算后的值
RegWrite	O	寄存器堆写使能信号
MemWrite	O	数据存储器写使能
nPC_sel	O	检测 Beq 指令
Extop	O	选择扩展的方式
jsel	O	J 指令使能信号
ALUctr	O	ALU 执行操作选择信号
opcode	I	输入指令的 opcode 码
funct	I	输入指令的 funct 码

(3) 功能定义

序号	功能名称	功能描述
1	选择	选择相应的功能操作，当某些指令执行时相应的选择信号进行选择
2	使能信号	对相应的模块的功能进行使能判断，即什么时候该模块执行相应操作

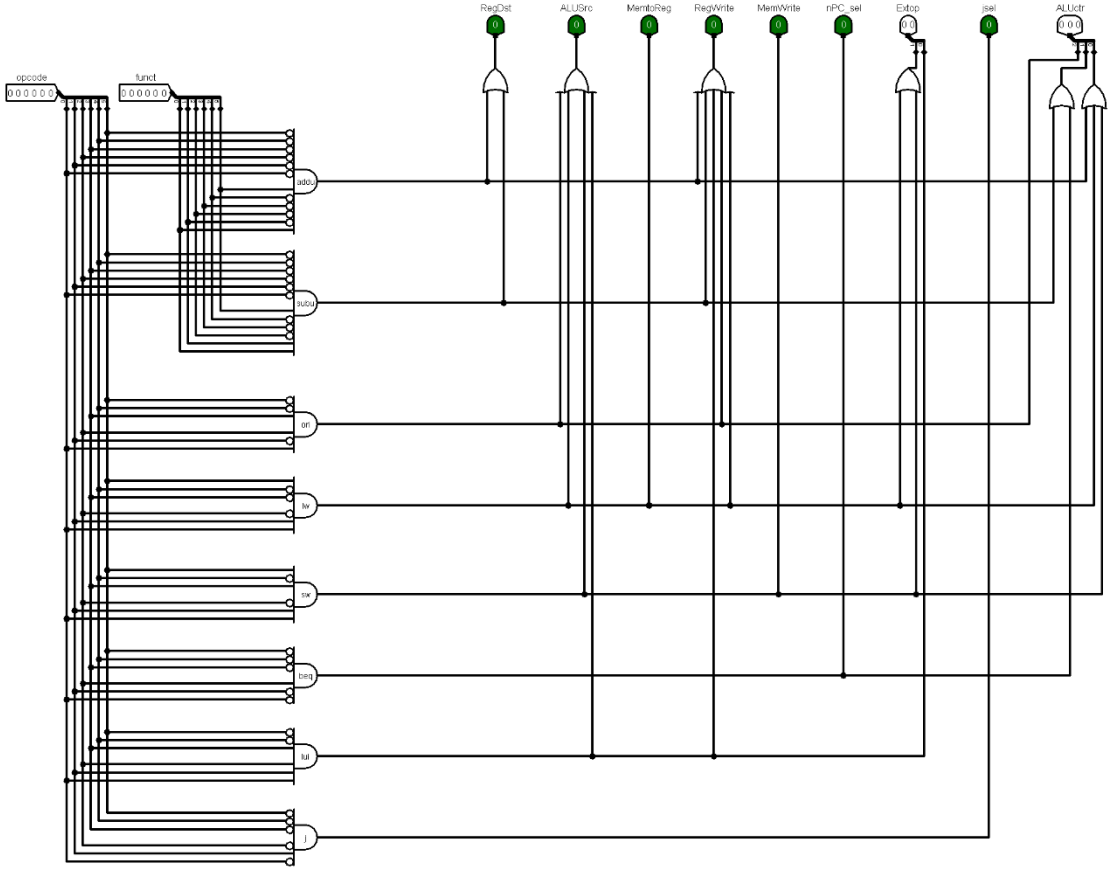

图 9 Controller Logisim 电路截图

3. 机器指令描述

助记符	Opcode	rs	rt	rd	shamt	Funct
ADDU	000000	5 位	5 位	5 位	00000	100001
SUBU	000000	5 位	5 位	5 位	00000	100011

助记符	Opcode	rs	rt	immediate
ORI	001101	5 位	5 位	16 位
LW	100011	5 位	5 位	16 位

SW	101011	5 位	5 位	16 位
BEQ	000100	5 位	5 位	16 位
LUI	001111	5 位	5 位	16 位
J	000010	26 位 instr_index		

ADDU:

实现两个 32 位数相加，不考虑溢出

GPR[rd] <- GPR[rs] + GPR[rt]

SUBU:

实现两个 32 位数相减，不考虑溢出

GPR[rd] <- GPR[rs] - GPR[rt]

ORI：

与一个 16 位立即数做完高位 0 扩展后进行或操作

GPR[rt] <- GPR[rs] or immediate

LW:

从数据存储器内提取相应的值存入寄存器中

GPR[rt] <- memory[GPR[base] + offset]

SW:

向数据存储器内相应的地址写入值

Memory[GPR[base] + offset] <- GPR[rt]

LUI:

将一个 16 位常量赋值给寄存器中的高 16 位，低 16 位清 0

GPR[rt] <- immediate || 0^{16}

BEQ:

比较 rs 对应寄存器和 rt 对应寄存器的内容，如果相等，则跳转至标号位置，不相等则继续执行 PC + 4

If GPR[rs] = GPR[rt] then branch

J:

无条件跳转至 256MB 的区间内的任意指令位置

4. 测试程序

```
ori $1,0x1010              #测试 ori，向$1 中写入 0x1010
ori $2,0x0101              #测试 ori，向$2 中写入 0x0101

ori $3,10                  #给$3 赋值为 10
ori $4,100                 #给$4 赋值为 100
ori $5,10                  #给$5 赋值为 10

addu $20,$20,$3            #测 addu $20=10
addu $21,$21,$4            #测 addu $21=100

beq $3,$5,then             #测试 beq 会成功跳转时的情况
circle1:
addu $6,$6,$3              #$6 永远为 0
j end                      #不会走这个 j 指令

then:                      #通过 beq 执行 then 标号下指令
sw $3,4($0)                #测试 sw，将$3 处的值写入数据存储器地址为 4 的地方，值为 10
sw $4,12($0)               #测试 sw，将$4 处的值写入数据存储器地址为 12 的地方，值为 100

end:
subu $4,$4,$3              #测试 subu 指令，令$4 内的值 100 不断减$3 内的值 10
beq $3,$4,circle2          #测试 beq 不跳转的情况，进入 PC+4
j end                      #测试 j 指令

circle2:
subu $21,$21,$20           #测试 subu 指令$21 内的值 100 减$20 内的值 10
subu $21,$21,$20           #测试 subu 指令$21 内的值 100 减$20 内的值 10
lw $15,4($0)               #测试 lw 指令，从 4 位置处取出刚才存的值 10，赋值给寄存器$15
lw $16,12($0)              #测试 lw 指令，从 4 位置处取出刚才存的值 100，赋值给寄存器$16

ori $17,0x1111             #给寄存器$17 低位赋值 0x1111
ori $18,0xabcd             #给寄存器$18 低位赋值 0xabcd
lui $17,0xabab             #测试 lui 指令，给寄存器$17 的高位赋值 abab，低位清 0
lui $18,0xcdcd             #测试 lui 指令，给寄存器$18 的高位赋值 cdcd，低位清 0
```

5. 测试结果与说明

下图为测试程序对应的结果

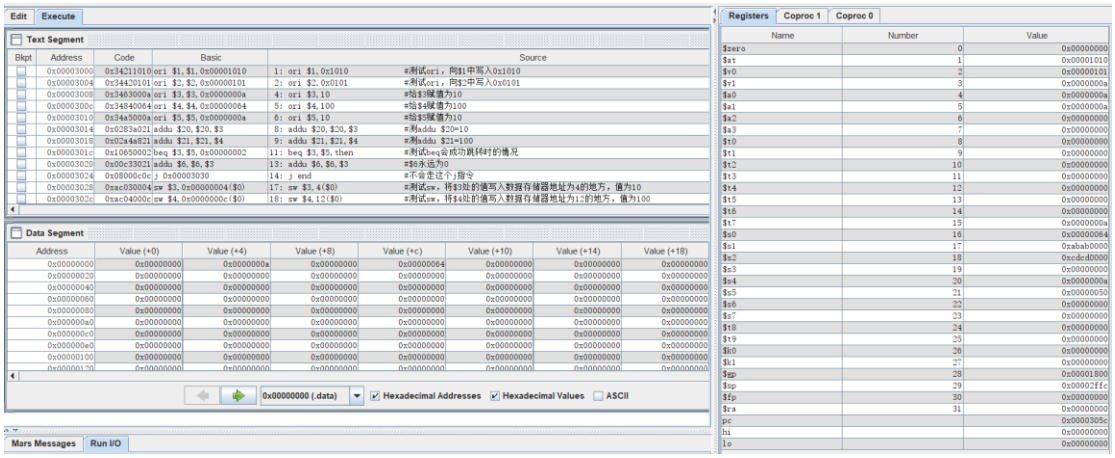

图 10 测试程序结果截图

只需要把 MARS 中运行后的寄存器数据的值，与电路中 GPR 里 32 位寄存器的值进行比较，若相等则可认定该电路连接正确，反之则电路存在问题，无法认证成功实现 8 条指令。

在我写的测试程序中，每一条指令都执行了至少两遍，且符合要求程序行数超过 20，也测试了 BEQ 指令跳转成功和无法跳转的两种情况，中间利用 BEQ 和 J 指令实现了一个小循环，循环中一直对$4 寄存器进行 subu 操作，最后两条指令，先将$17 和$18 寄存器的低位赋值，再利用 lui 指令对其高位进行赋值，低位清零，以此来检测清零过程是否正确。下图为 GPR 中 32 个寄存器存储情况。根据一一比对后，发现与 MARS 中运行结果一样，因此可判断该数据通路，连接正确，并且成功实现了 8 条指令。

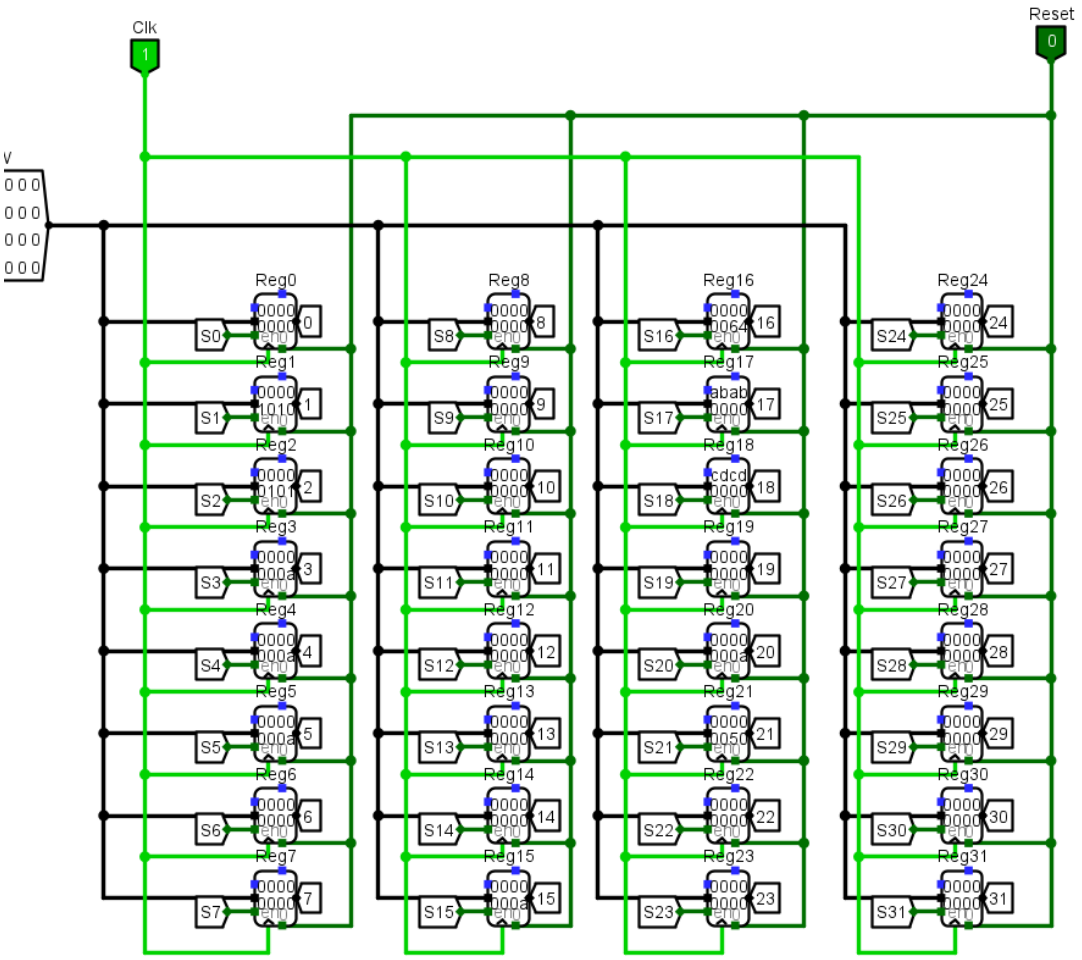

图 11 GPR 存储情况截图

6. 收获、体会、总结

这次单周期数据通路大作业做的可谓是一波三折。我不是一个聪明的学生，不像别的大佬们一样，上完课，拿到大作业就知道从哪里开始，就知道如何去做。借着这个机会我想讲一下自己是如何做大作业的，同时也写一写自己的感悟。

起初自己完全不知道从哪里入手去做，知道了每一条指令的功能，也知道了数据通路的具体流程，还是不明白如何起步。后来是看了日新学堂的 IFU 连接后，才慢慢有了头绪。

这里总结一下我的思路过程。首先要有自顶向下的思维方式，明确数据通路的整体布局是什么，对于每一个模块你要实现的东西是什么，这些一定要在设计之前想好，不然在设计的时候会遇到没有头绪，没有思路的情况。明确好输入输出，想好如果要实现这个功能，那么我应该需要些什么。这些看似很简单的问题，其实是最重要的。例如一开始，我的 LUI 指令就是没有办法正确实现，后来发现了电路连接的 bug，然后重新设计了 controller 后才改好。

其实最能体现这次大作业的整体思想的模块便是 controller 模块，它是对选

择信号的一个控制模块，在上课的时候我就在想，那么多选择信号，我要怎么一个个控制，结果后来老师讲到要用一个控制器。如果要明确每一个选择信号是什么，那当然就要对整个通路的流程非常清楚，因此，在我的设计中，我在遇到不知道该干什么的时候，选择先去连控制器。

在连接好控制器以后，我便对整个数据通路有了更深入的认识。我知道了我的 ALU 应该具备这些这些功能，我的 DM 应该要干什么，具体的细节，再在我连接具体模块的时候去研究，但是有了大方向后，就真的容易很多，也不会出现我不知道该从哪里开始诸如此类的问题。

经过这次连接数据通路，我更加的佩服最初设计计算机的人了，尤其是我在研究 ALU 组间组内并行的时候，如果利用传统的思维方式，那么我肯定会以串联的方式去想问题，而并行的方法却打破了固有思路，同时产生进位，同时产生结果，通过数学表达式，去实现“并行”的方式。

总之，这次的单周期数据通路连接，让我了解了电路设计是什么样的，也再次开拓了我的眼界，和知识面。有一句话说的特别的对“纸上得来终觉浅，绝知此事要躬行”，只有自己亲自去实践了，才能真正理解所学的知识，我想这次的大作业，也便是这个道理吧。