计算机组成原理

课内大作业报告

学　　号_____20020203_____

姓　　名_____王思哲_____

指导教师_____魏坚华_____

提交日期_____2022 年 5 月 9 日_____

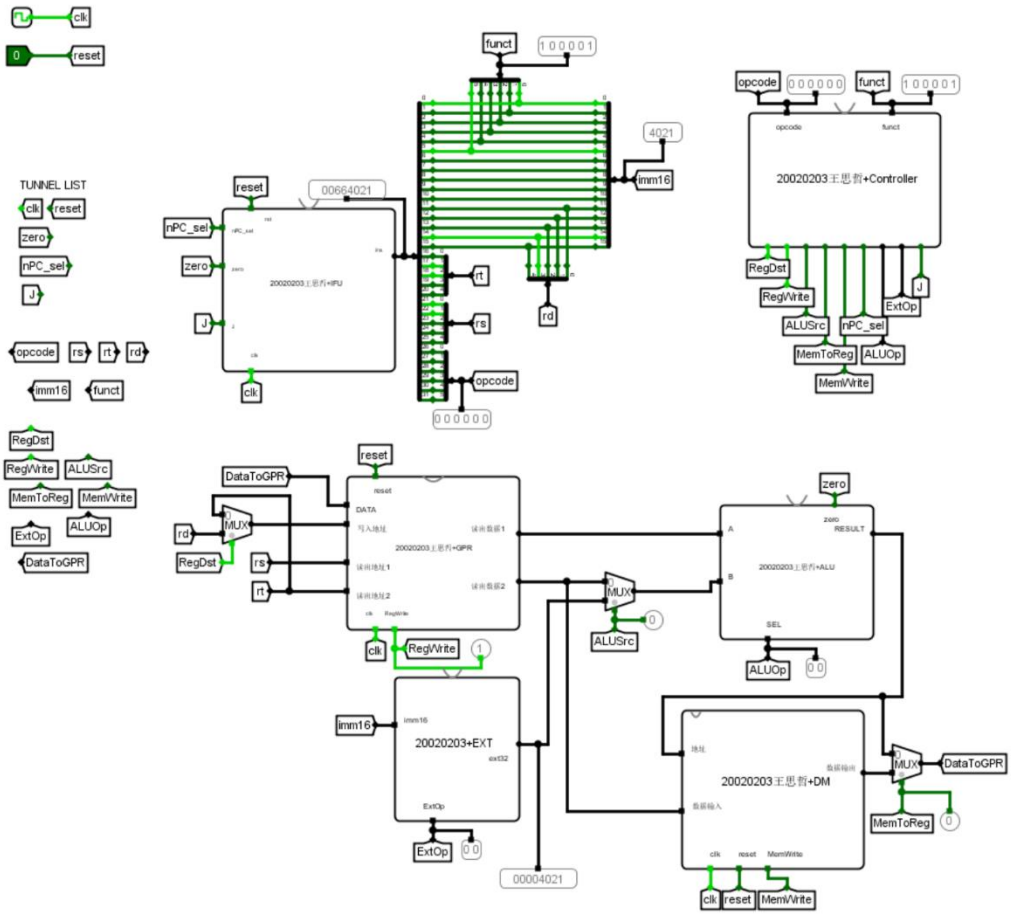
成绩评价表

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

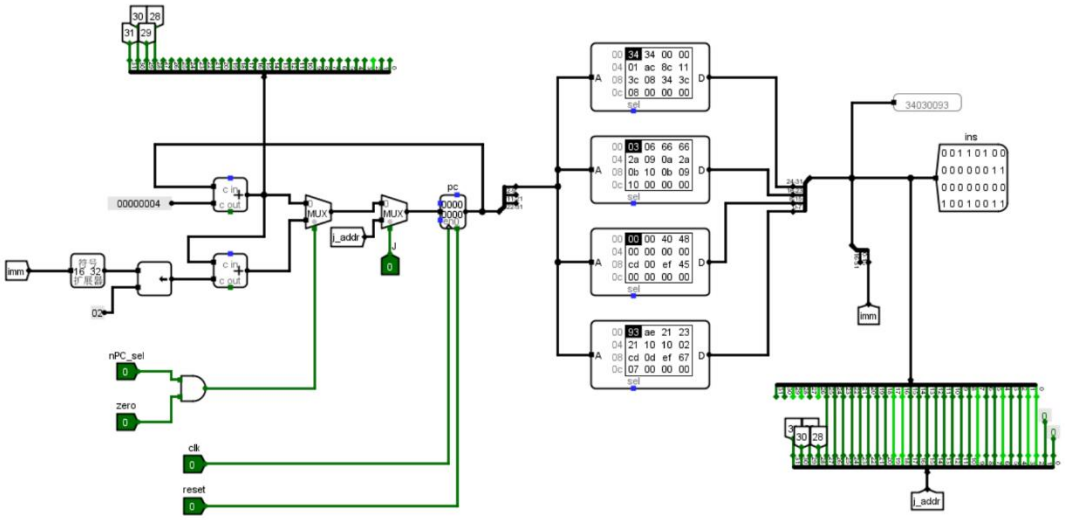
一、 总体数据通路结构设计图

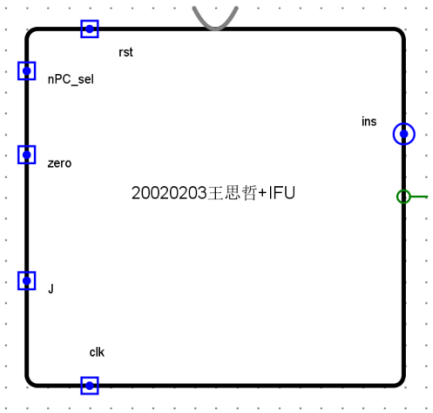


二、 模块定义

2.1 IFU 模块

2.1.1 设计总览&模块封装





2.1.2 基本描述

IFU 在数据通路中充当的是取指部件的角色，主要完成取指令的功能。
IFU 内部结构关键部件有 PC、指令存储器。其中，PC 使用寄存器实现，位宽为 32 位。IM 由 4 个 ROM 组成，其输入地址为 8 位，数据位宽也为 8 位。
IFU 在每个时钟沿到来时进行更新，分为两种情况：当为一般情况时，PC=PC+4；当为 lui 或 j 等跳转指令时，PC 会根据跳转地址进行更新。

2.1.3 模块接口

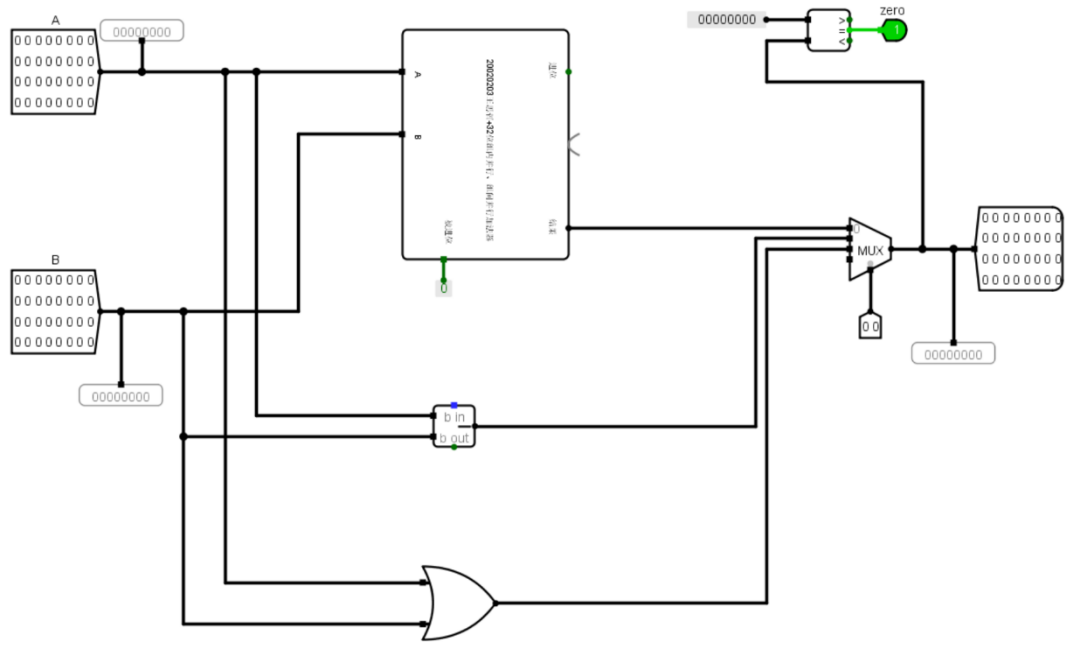
信号名	方向	描述
nPC_sel	I	当前指令是否为 beq 指令。 1：指令为 beq 指令 0：指令非 beq 指令
zero	I	ALU 计算结果是否为 0 1：是 0：否
clk	I	时钟信号
reset	I	复位信号，使 PC 为 0。 1：复位 0：无效
J	I	当前指令是否为 J 指令 1:指令为 J 指令 0:指令非 J 指令
Ins[31:0]	O	PC 指针指向指令存储器对应地址的指令。

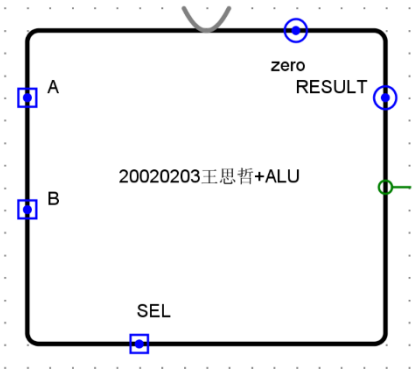
2.1.4 功能描述

序号	功能名称	功能描述
1	复位	当复位信号有效时，PC 置 0。
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 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) 如果当前是 J 指令，则 PC[31:28]由 PC+4 高四位提供，PC[27:2]由 ins 低 26 位提供，PC[1:0]补 0.

2.2 ALU 模块

2.2.1 设计总览&模块封装





2.2.2 基本描述

根据选择信号指示的运算类型，将两个输入的数据进行对应的运算后，输出结果。同时，如果结果为 0，则将 zero 信号置 1.

2.2.3 模块接口

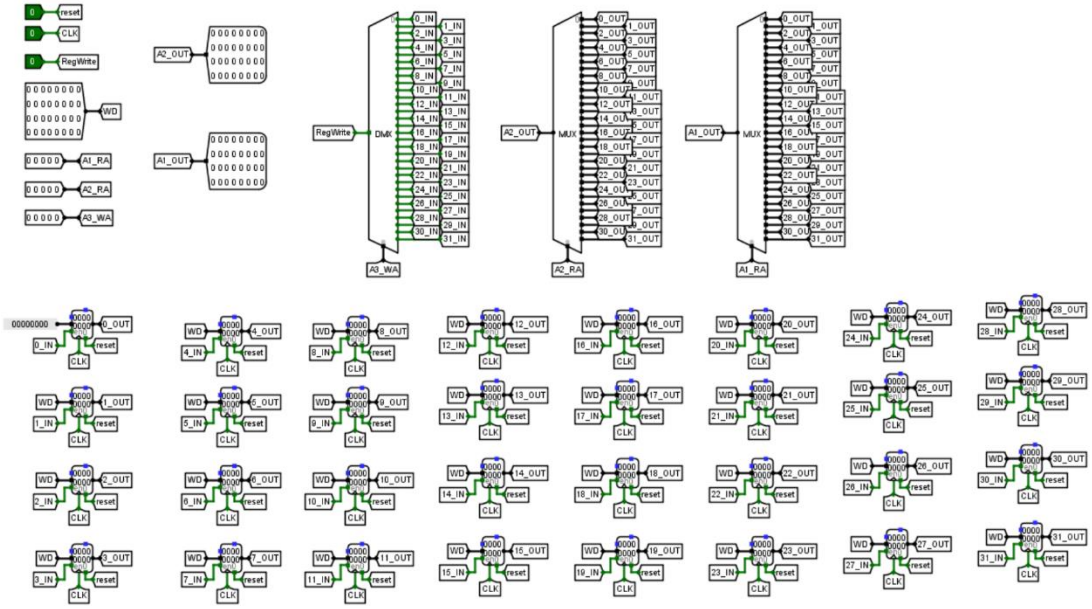
信号名	方向	描述
A [31:0]	I	操作数 1
B [31:0]	I	操作数 2
SEL[1:0]	I	运算类型 00：加法 01：减法 10：逻辑或
zero	O	判断运算结果是否为 0 1:为 0 0:非 0
RESULT[]	O	运算结果输出

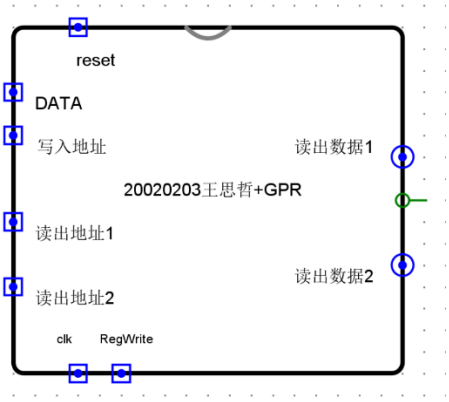
2.2.4 功能描述

序号	功能名称	功能描述
1	加法运算	计算操作数 1+操作数 2
2	减法运算	计算操作数 1-操作数 2
3	逻辑或运算	计算操作数 1 || 操作数 2
4	结果 0 判断	判断运算结果是否为 0

2.3 GPR 模块

2.3.1 设计总览&模块封装





2.3.2 基本描述

包括 32 个具有写使能的 32 位寄存器。写使能有效时可以根据数据写入地址以及输入的数据写数据，任何时刻都可以根据读数据地址读数据。可以异步置 0.

2.3.3 模块接口

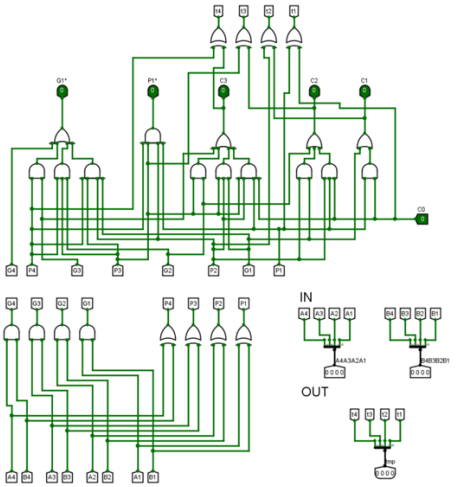
信号名	方向	描述
reset	I	寄存器数据是否置 0 1：置 0 0：无效
clk	I	时钟信号
RegWrite	I	写使能信号 1：写有效 0：写无效
DATA[31:0]	I	数据输入
写入地址[4:0]	I	数据应该写入的寄存器的地址
读出地址 1[4:0]	I	应该读数据的寄存器的地址
读出地址 2[4:0]	I	应该读数据的寄存器的地址
读出数据 1[31:0]	O	数据输出
读出数据 2[31:0]	O	数据输出

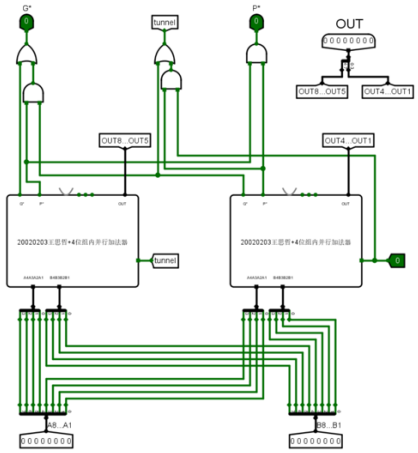
2.3.4 功能描述

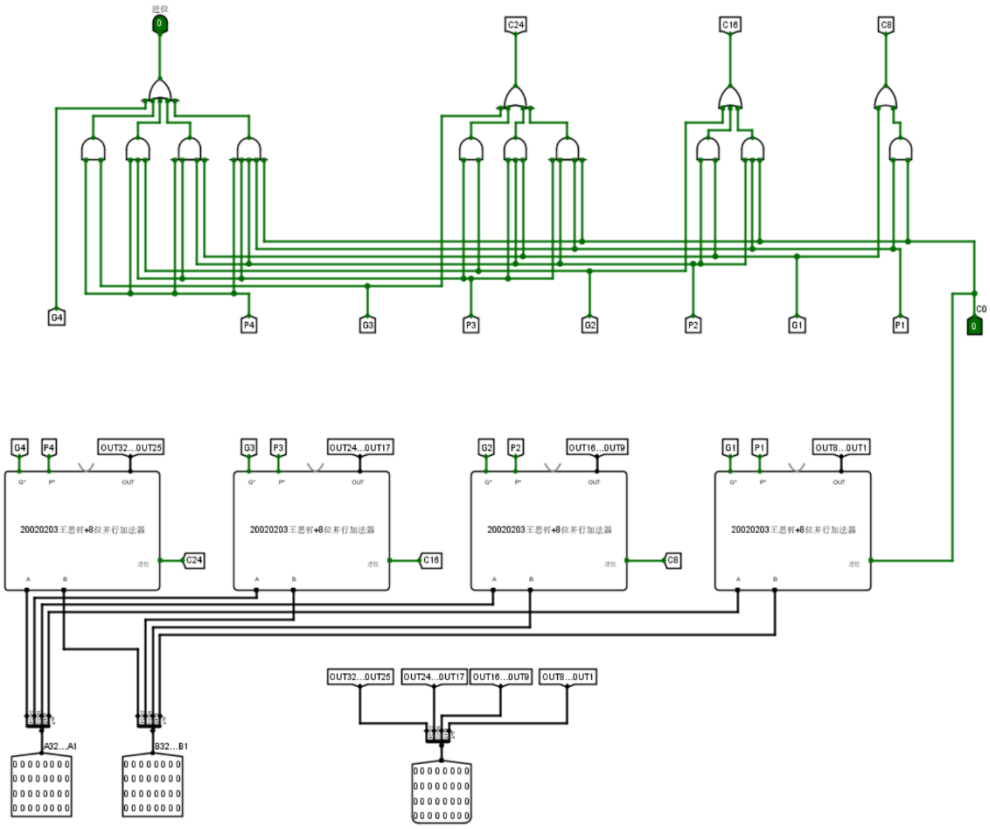
序号	功能名称	功能描述
1	复位	当复位信号有效时，32 个寄存器内容均置 0.
2	写数据	根据写入地址，将数据写入寄存器中。
3	读数据	根据读出地址，将数据读出并输出。

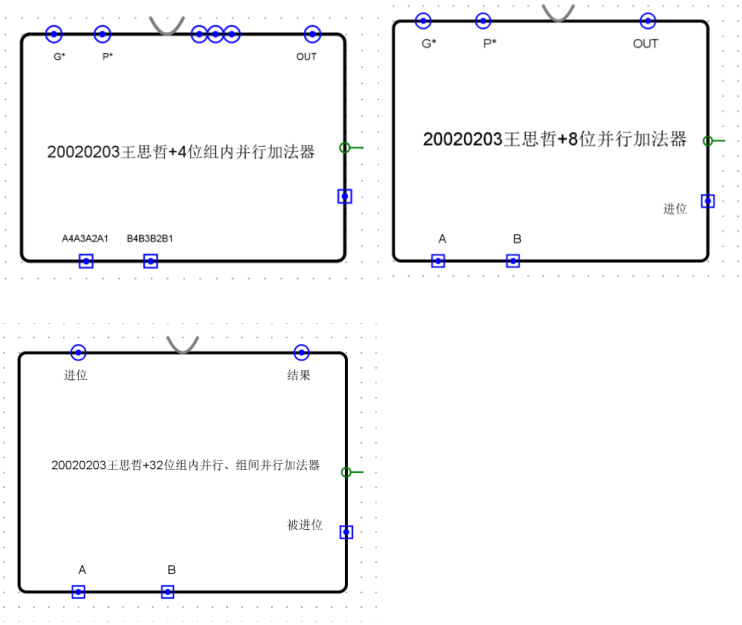
2.4 并行加法器模块

2.4.1 设计总览&模块封装









2.4.2 基本描述

32 位组内组间全并行加法器实现两个操作数的加法运算，支持溢出功能。
32 位并行加法器是由 4 个 8 位加法器组间并行组成，其中 8 位由两个 4 位加法器构成，4 位加法器实现组内并行的加法运算。
设计参考：计算机组成原理与汇编语言 P43-P45 内容

2.4.3 模块接口

只给出 32 位组内组间全并行加法器模块接口，8 位&4 位加法器略。

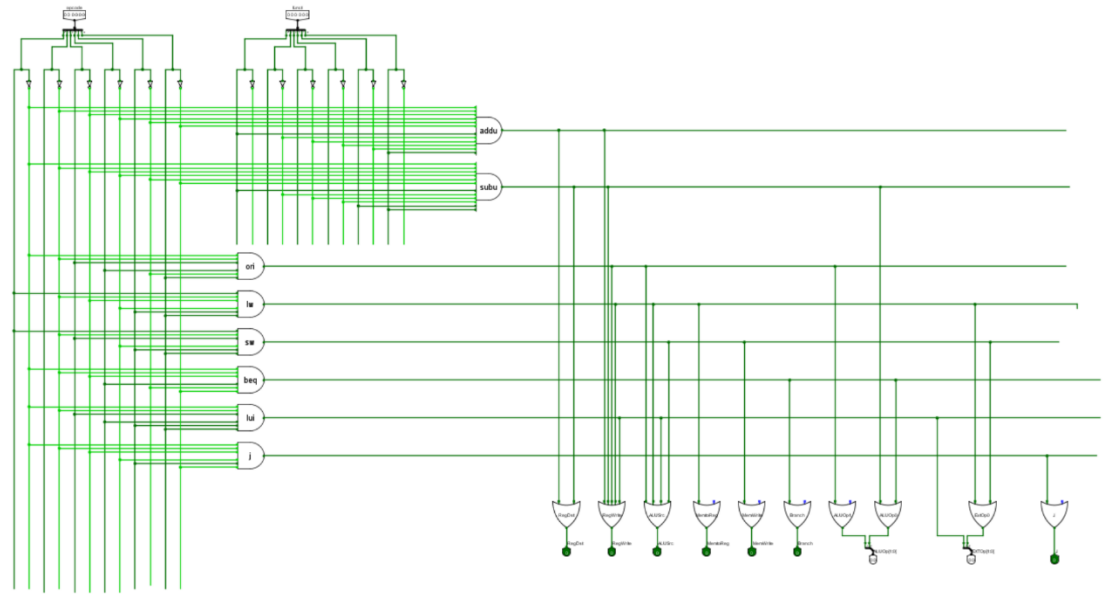
信号名	方向	描述
A[31:0]	I	操作数 1
B[31:0]	I	操作数 2
被进位	I	一般置常数 0
进位	O	向上进位，溢出判断。 1：有溢出 0：无溢出
结果[31:0]	P	运算结果：操作数 1+操作数 2

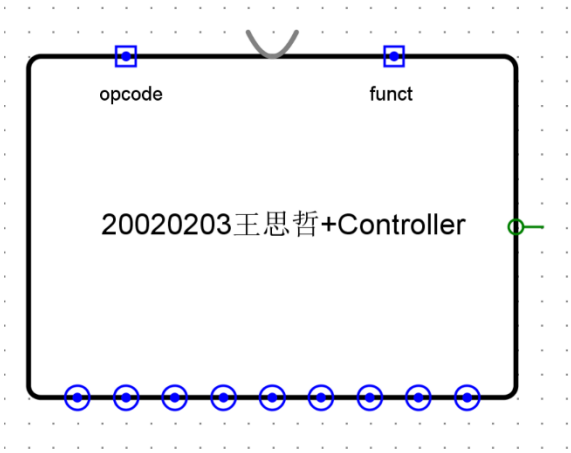
2.4.4 功能描述

序号	功能名称	功能描述
1	加法运算	将两个输入的操作数进行加法运算。

2.5 Controller 模块

2.5.1 设计总览&模块封装





2.5.2 基本描述

根据 opcode 和 funct 信号产生对应的控制信号。

2.5.3 模块接口

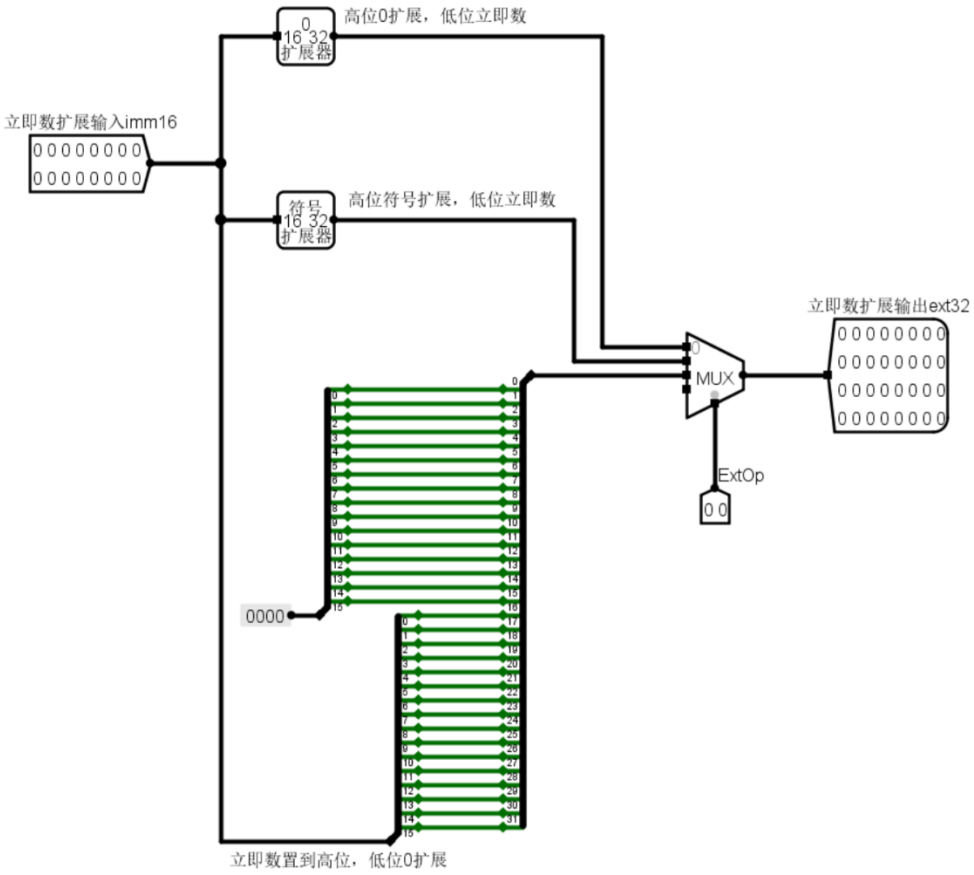
信号名	方向	描述
opcode[5:0]	I	操作码
funct[5:0]	I	功能码
RegDst	O	判断是否使用 rd 作为写入寄存器的地址 1：是 0：否
ALUSrc	O	判断是否使用扩展后的立即数作为 ALU 的第二操作数 1：是 0：否
MemToReg	O	判断是否使用 DM 中读出的数据输入寄存器 1：是 0：否
MemWrite	O	判断是否写入寄存器 1：是 0：否
RegWrite	O	判断是否写入寄存器 1：是 0：否
nPC_sel	O	判断是否为 beq 指令 1：是 0：否
ALUOp[1:0]	O	ALU 的操作类型 00：加法 01：减法 10：逻辑或
ExtOp[1:0]	O	立即数拓展的类型 00：0 拓展 01：符号拓展 10：将 16 位立即数扩展至高 16 位，低位补 0
J	O	判断是否为 j 指令 1：是 0：否

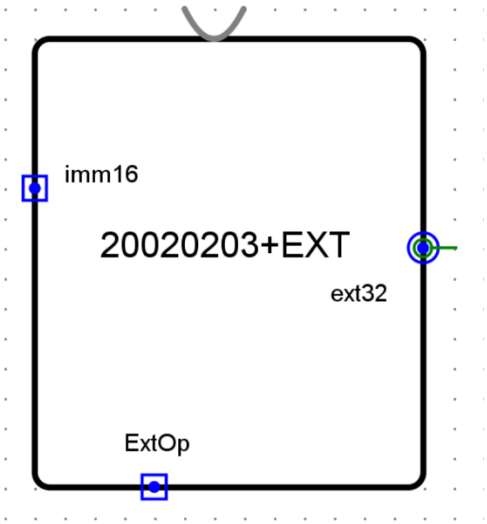
2.5.4 功能描述

序号	功能名称	功能描述
1	产生控制信号	根据 opcode & funct 信息，产生对应的控制信号。

2.6 EXT 模块

2.6.1 设计总览&模块封装





2.6.2 基本描述

根据选择信号 ExtOp[1:0]指示的类型，将 16 位立即数拓展到 32 位。类型包括 0 拓展，符号拓展以及立即数拓展至高位、低位补 0 三种。

2.6.3 模块接口

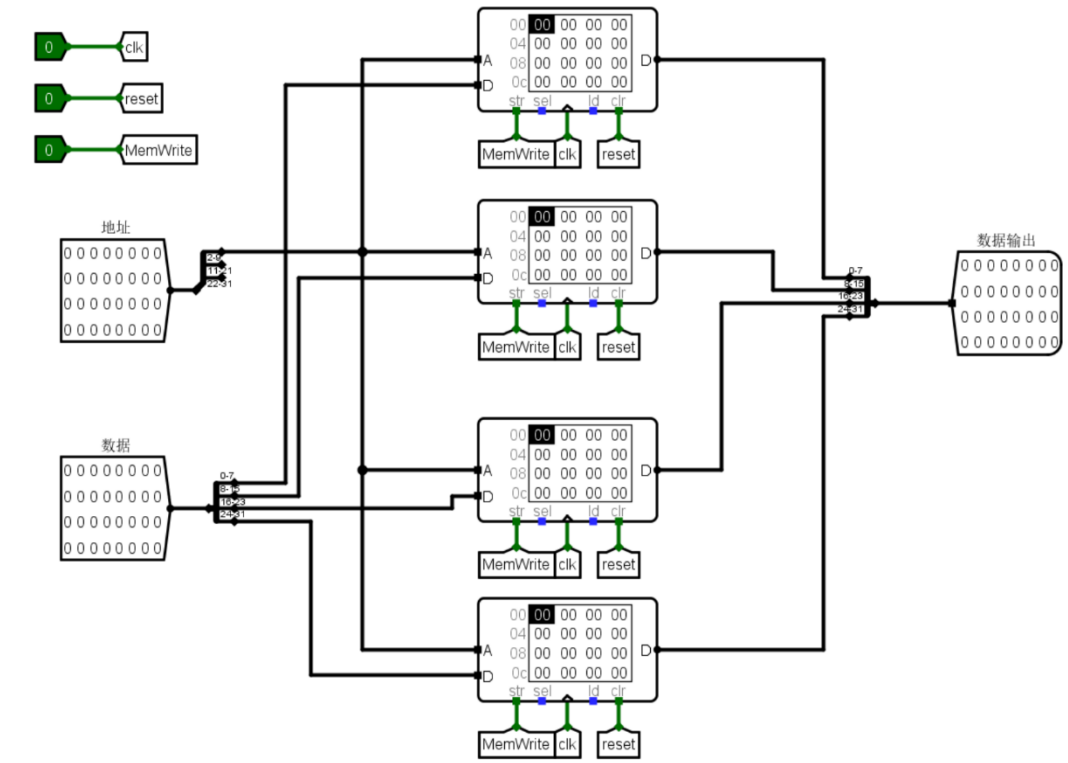
信号名	方向	描述
imm16[15:0]	I	16 位立即数
ExtOp[1:0]	I	选择立即数拓展的类型 00：0 拓展 01：符号拓展 10：立即数拓展至高位，低位补 0
ext32	O	拓展结果

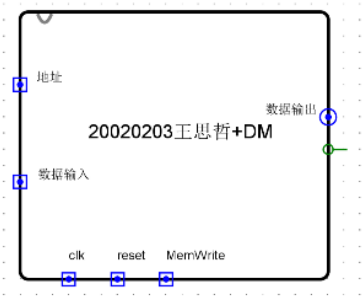
2.6.4 功能描述

序号	功能名称	功能描述
1	0 拓展	高 16 位 0 拓展
2	符号拓展	高 16 位符号拓展
3	lui 指令拓展	高 16 位置为原立即数，低 16 位补 0

2.7 DM 模块

2.7.1 设计总览&模块封装





2.7.2 基本描述

4 个存储体并行的小端序存储方式存储，容量 1KB，RAM 实现。输入地址 32 位，输入数据 32 位。取输入地址 2-9 位作为 RAM 存储地址的位置。

2.7.3 模块接口

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号，存储器中数据置 0。 1：置 0 0：无效
MemWrite	I	是否写入存储器中 1：是 0：否
地址[7:0]	I	写入的存储器单元地址
数据输入[31:0]	I	需要写入的数据
数据输出[31:0]	O	地址对应的存储器中读出的数据

2.7.4 功能描述

序号	功能名称	功能描述
1	复位	当复位信号有效时，存储器中数据置 0。
2	写入数据	根据地址将数据写入对应的存储器单元
3	读取数据	根据地址从存储器单元中读出数据。

三、 指令描述

3.1 addu 无符号加法

addu： 无符号加

编码	31 26	25 21	20 16	15 11	10 6	5 0
	special 000000	rs	rt	rd	0 00000	addu 100001
	6	5	5	5	5	6
格式	addu rd, rs, rt					
描述	GPR[rd] ← GPR[rs] + GPR[rt]					
操作	GPR[rd] ← GPR[rs] + GPR[rt]					
示例	addu $s1, $s2, $s3					
其他						

3.2 subu 无符号减法

subu：无符号减

编码	31 26	25 21	20 16	15 11	10 6	5 0
	special 000000	rs	rt	rd	0 00000	subu 100011
	6	5	5	5	5	6
格式	sub rd, rs, rt					
描述	GPR[rd] ← GPR[rs] - GPR[rt]					
操作	GPR[rd] ← GPR[rs] - GPR[rt]					
示例	sub $s1, $s2, $s3					
其他	subu 不考虑减法溢出。例如 0x0000_0000 - 0xFFFF_FFFF = 0x0000_0001，即结果为非负值。					

3.3 ori 或立即数

ori：或立即数

编码	31 26	25 21	20 16	15 0
	andi 001101	rs	rt	immediate
	6	5	5	16
格式	ori rt, rs, immediate			
描述	GPR[rt] ← GPR[rs] OR immediate			
操作	GPR[rt] ← GPR[rs] OR zero_extend(immediate)			
示例	ori $s1, $s2, 0x55AA			
其他				

3.4 lw 加载字

lw：加载字

编码	31 26	25 21	20 16	15 0
	lh 100011	base	rt	offset
	6	5	5	16
格式	lw rt, offset(base)			
描述	GPR[rt] ← memory[GPR[base]+offset]			
操作	Addr ← GPR[base] + sign_ext(offset)			

	GPR[rt] ← memory[Addr]
示例	lw $v1, 8($s0)
约束	Addr 必须是 4 的倍数(即 $Addr_{1..0}$ 必须为 00)，否则产生地址错误异常

3.5 sw 存储字

sw：存储字

编码	31 26	25 21	20 16	15 0	
	sw 101011	base	rt	offset	
	6	5	5	16	
格式	sh rt, offset(base)				
描述	GPR[rt] ← memory[GPR[base]+offset]				
操作	Addr ← GPR[base] + sign_ext(offset) memory[Addr] ← GPR[rt]				
示例	sw $v1, 8($s0)				
约束	Addr 必须是 4 的倍数(即 $Addr_{1..0}$ 必须为 00)，否则产生地址错误异常				

3.6 beq 相等时转移

beq：相等时转移

编码	31 26	25 21	20 16	15 0	
	beq 000100	rs	rt	offset	
	6	5	5	16	
格式	beq rs, rt, offset				
描述	if (GPR[rs] == GPR[rt]) then 转移				

操作	if (GPR[rs] == GPR[rt])
	PC ← PC + sign_extend(offset‖0^2)
	else
	PC ← PC + 4				

示例	beq $s1, $s2, -2				
其他					

3.7 lui 立即数加载到高位

lui：立即数加载至高位

编码	31 26	25 21	20 16	15 0
	lui 001111	0 00000	rt	immediate
	6	5	5	16
格式	lui rt, immediate			
描述	lui rt, immediate || 0^{16}			
操作	lui rt, immediate || 0^{16}			
示例	lui $s1, 0x55AA			
其他				

3.8 j 跳转

j：跳转

编码	31 26	25 0
	j 000010	instr_index
	6	26
格式	j target	
描述	j 指令是 PC 相关的转移指令。当把 4GB 划分为 16 个 256MB 区域，j 指令可以在当前 PC 所在的 256MB 区域内任意跳转。	
操作	PC ← PC31..28||instr_index||0^{2}	
示例	j Loop_End	
其他	如果需要跳转范围超出了当前 PC 所在的 256MB 区域内时，可以使用 JR 指令。	

四、 测试程序

```
ori $3, $0, 0x93   # 将0x93与0号寄存器进行或运算，将运算结果存入3号寄存器。
ori $6, $0, 0xae   # 将0xae与0号寄存器进行或运算，将运算结果存入6号寄存器。
addu $8, $3, $6    # 将3号寄存器内容与6号寄存器内容进行无符号加法，将运算结果存入8号寄存器。
subu $9, $3, $6    # 将3号寄存器内容与6号寄存器内容进行无符号减法，将运算结果存入9号寄存器。
addu $0, $9, $10   # 将10号寄存器内容与9号寄存器内容进行无符号加法，将运算结果存入0号寄存器。（但0号寄存器内容无法被修改）
sw $9, 16($0)      # 将9号寄存器内容存入：0号寄存器的数据偏移16后的地址 所指示的内存中。
lw $10, 16($0)     # 从： 0号寄存器中的数据偏移16后的地址 的内存单元中，取出数据，存入10号寄存器中
13:beq $9, $10, 11        # 若 9号寄存器 & 10号寄存器 中数据相等，则跳转到label 11处
lui $11, 0xcdcd    # 将0xcdcd加载到11号寄存器的数据的高16位，低位补0
j end              # 跳转到label end处
11:ori $11, $0, 0xefef     # 将0xefef与0号寄存器内容相或，结果存入11号寄存器
lui $9, 0x4567     # 将0x4567存入9号寄存器的数据的高16位，低位补0
j 13               # 跳转到label 13处
end:               # label end
```

五、 测试结果

将测试程序 test-m.asm 分别在 Mars 和 logisim 中仿真，观察 logisim 中的仿真结果是否与 Mars 中相同，从而验证电路设计的正确性。

5.1 Mars 中仿真结果

5.1.1 寄存器数据

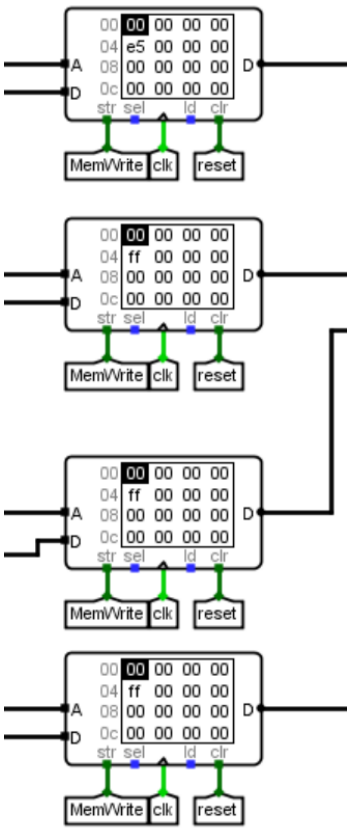
Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000093
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x000000ae
$a3	7	0x00000000
$t0	8	0x00000141
$t1	9	0x45670000
$t2	10	0xffffffe5
$t3	11	0xcdcd0000

5.1.2 数据存储器数据

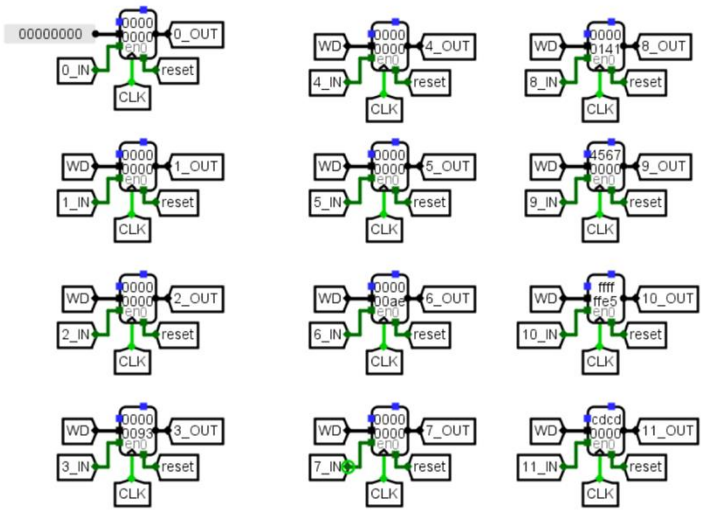
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0xffffffe5

5.2 logisim 中仿真结果

5.2.1 寄存器数据



5.2.2 数据存储器数据



5.3 结论

测试文件在 Mars 中与在 logisim 中仿真结果相同，即电路设计正确。

六、 总结与心得体会

通过这次开发单周期处理器，我了解了一个处理器是如何工作的，也对于 MIPS 指令系统每一条指令的构成、作用有了实际的体会。

开发一个处理器，首先需要明确的是需要什么样的硬件基础以及什么样的数据通路，然后再根据数据通路设置控制信号，来完成指定的功能。控制信号的生成依赖于当前指令，即根据当前指令的操作码或功能码来产生对应的指令，控制硬件设备的工作，控制数据传送的路径，最终实现指定的功能。

一开始听到自己要开发一个 cpu 的时候我感觉非常的不可思议，十分的困难。但听过老师的讲解与自己的思考过后，我从每一个基础的原件开始搭建：从加法器开始、到 IM 结束。每完成一个部件的搭建，我都对其有了更为透彻的理解和更深的记忆。最后，我使用了 3 天时间完成了单周期处理器的搭建与测试工作，搭建很成功，这给予了我很大的成就感，也让我对以后的学习更加的有了自信。

总结自己的工作，我认为搭建过程的难点有以下几点：

一是存储器数据位宽、地址位宽、地址来源的选择。

二是跳转指令需要跳转到的地址的计算，即 PC 的更新问题

三是比较容易忽略的点，即$0 寄存器需要指定为常数 0，其内容不应该被允许改变，这也正是我初次搭建时忽略、仿真时改正的错误。