计算机组成原理

期末大作业报告

学　　号_____20020203_____

姓　　名_____王思哲_____

指导教师_____魏坚华_____

提交日期_____2022 年 6 月 14 日___

成绩评价表

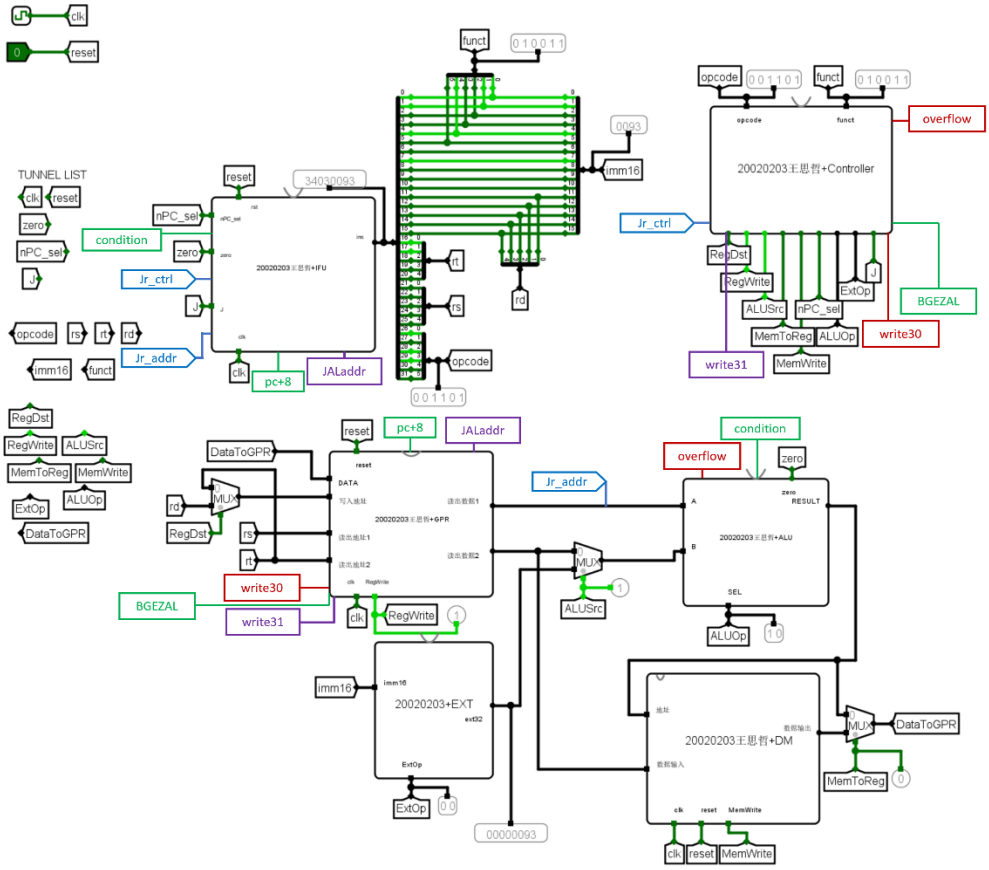
报告内容	报告结构	报告最终成绩
☐丰富正确 ☐基本正确 ☐有一些问题 ☐问题很大	☐完全符合要求 ☐基本符合要求 ☐有比较多的缺陷 ☐完全不符合要求	
报告与大作业功能一致性	**报告图表**	**总体评价**
☐完全一致 ☐基本一致 ☐基本不一致	☐符合规范 ☐基本符合规范 ☐有一些错误 ☐完全不正确	

教师签字:_____

目录

一、 总体数据通路结构设计图

本次单周期处理器的数据通路设计图(MIPS-Litel 指令集)，是在之前的数据通路图(MIPS-Lite 指令集)上进行的扩充，具体信号通路设计如下图所示。



二、 模块定义

2.1 ALU 模块

2.1.1 基本描述

ALU 模块为运算模块，主要功能是根据控制信号指定的运算类型，对两个输入的 32 位操作数进行相应的运算（也有可能只用到了一个操作数）并输出运算结果。除了对操作数进行相应的运算外，ALU 还支持溢出检测、判断结果是否为 0 以及判断结果是否大于 0。

2.1.2 模块接口

信号名	方向	描述
A[31:0]	I	第一个操作数

B[31:0]	I	第二个操作数
ALUOp[2:0]	I	控制运算类型 000：加法　001：减法　010：逻辑或 011：小于置一 100：加立即数 101：大于等于 0 跳转
zero	O	判断 alu_res 是否为 0 0：否　1：是
alu_res	O	32 位运算结果
overflow	O	判断 addi 指令第一个操作数加立即数（有符号）是否产生溢出 0：否　1：是
condition_jdg	O	判断第一个操作数是否大于等于 0 0：否　1：是

2.1.3　功能定义

序号	功能名称	功能描述
1	计算	根据运算控制信号完成相应运算： 加法：计算操作数 1+操作数 2 减法：计算操作数 1-操作数 2 逻辑或：计算操作数 1 || 操作数 2 小于置一：若操作数 1<操作数 2，结果为 1，否则为 0 加立即数：计算操作数 1+操作数 2，同时判断是否产生溢出 大于等于 0 跳转：比较操作数 1 与数字 0 的大小
2	判断	判断运算结果是否为 0 判断运算结果是否大于等于 0 判断 addi 运算时是否产生了溢出

2.2 IFU 模块

2.2.1　基本描述

IFU 主要完成的是取指功能。IFU 内部结构关键部件有 PC 和指令寄存器。指令寄存器负责存储程序需要运行的所有指令，位宽 8 位，长度 1024。PC 由寄存

器存储，在每次时钟沿到来时更新，更新为下一条指令的地址。
PC 地址的更新分为 5 种情况，分别是 j 指令的更新、beq 指令的更新、顺序执行下一条指令、jr 指令的更新、bgezal 指令的更新。

2.2.2 模块接口

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号 0：不复位 1：复位
npc_sel	I	选择 pc 的下一个地址是否为 beq 成立时的地址
zero	I	判断 alu_res 是否为 0，与 npc_sel 同时作用，用于选择下一个地址是否为 beq 成立时的地址
j	I	如果 j==1，则 pc 更新为 j 指令跳转的地址
jr_ctrl	I	如果 jr_ctrl==1，则 pc 更新为 jr 指令需要跳转的地址
condition_jdg	I	如果 condition_jdg==1，则 pc 更新为 bgezal 需要跳转的地址
insout[31:0]	O	输出 pc 所指向的当前指令
opcode[5:0]	O	opcode 操作码
rs[4:0]	O	寄存器地址
rt[4:0]	O	寄存器地址
rd[4:0]	O	寄存器地址
funct[5:0]	O	funct 功能码
imm16[15:0]	O	立即数
jalAddr[31:0]	O	pc+4
pc_p8[31:0]	O	pc+8

2.2.3 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 置 0
2	取指令	根据 PC 从 IM 中取出对应的指令
3	计算下一条指令地址	如果不是以下任何指令，pc←pc+4 如果是 beq 指令，且 zero 为 1，pc←pc+4+(sign_ext(insout[15:0])<<2) 如果是 j 指令，pc←{(pc+4)[31:28], insout[25:0], 2’b00} 如果是 jr 指令，pc←jrAddr 如果是 bgezal 指令，pc←pc+4+sign_ext(offset||0^2)

2.3 Controller 模块

2.3.1 基本描述

根据每一条指令的 opcode 和 funct 以及 alu 产生的溢出信号 overflow，产生与之对应的控制信号。

2.3.2 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
overflow	I	alu 加法(addi)溢出标志　0：未溢出　1：溢出
RegDst	O	选择寄存器写入地址　0：rt　1：rd
RegWrite	O	寄存器写使能信号　0：无效　1：有效
ALUSrc	O	选择 ALU 第二个操作数来源 0：gpr 第二个输出数据 1：立即数拓展结果
MemtoReg	O	选择传给寄存器的写入数据 0：alu 计算结果 1：dm 指定存储单元的数据
MemWrite	O	数据存储器写使能信号　0：无效　1：有效
Branch	O	是否为 beq 指令　0：否　1：是
J	O	当前是否为 j 指令，跳转信号　0：否　1：是
ALUOp[2:0]	O	alu 运算类型控制信号 000：加　001：减　010：逻辑或 011：小于置一 slt　100：addi　101：bgezal default：alu_res = 0
ExtOp[1:0]	O	立即数扩展类型 00：0 拓展 01：符号拓展 10：16 位立即数拓展至高 16 位，低位补 0
WriteToGPR_30	O	30 号寄存器写入标志　0：不写入　1：写入
jr_ctrl	O	jr 指令标志　0：不是 jr 指令　1：是 jr 指令
write_31	O	31 号寄存器写入标志　0：不写入　1：写入
bgezal_31	O	bgezal 专用的 31 号寄存器写入标志 0：不写入　1：写入

2.3.3 功能定义

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

2.4 GPR 模块

2.4.1 基本描述

寄存器模块，主要功能是数据的临时存储。含有 32 个 32 位寄存器，当对应编号寄存器的写使能有效的时候，可以完成数据的写入功能；任何时刻都可以完成数据的读取功能。可以将 32 个寄存器清零。0 号寄存器不能被写入。

2.4.2 模块接口

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
RegWrite	I	写使能信号(所有)　0：不能写入　1：写入
WriteToGPR_30	I	30 号寄存器写入标志　0：不写入　1：写入
writeData[31:0]	I	写入数据
writeAddr[4:0]	I	写入地址
readAddr_1[4:0]	I	读出数据 1 所在的地址
readAddr_2[4:0]	I	读出数据 2 所在的地址
write_31	I	jal 指令时 31 号寄存器写入标志　0：不写入　1：写入
bgezal_31	I	bgezal 指令时 31 号寄存器写入标志 0：不写入 1：写入
pc_p8[31:0]	I	bgezal 指令时的写入数据
jalAddr[31:0]	I	jal 指令时的写入数据
dataOut_1[31:0]	O	读出数据 1
dataOut_2[31:0]	O	读出数据 2

2.4.3 功能定义

序号	功能名称	功能描述
1	复位	复位信号有效时，32 个寄存器清零
2	写数据	30 和 31 号寄存器需要在特定的使能信号有效时写入数据；其余寄存器仅需在 RegWrite 写使能有效时，根据写

		入地址将数据写入寄存器中即可。
3	读数据	根据读取数据的地址，将数据读出并输出

2.5 DM 模块

2.5.1 基本描述

该模块主要功能为存储数据，容量为 1kb，位宽 8 位，长度 1024，采用小端序的方式存储数据。输入地址 32 位（只取低 10 位），输入数据 32 位，输出数据 32 位。

2.5.2 模块接口

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
MemWrite	I	写使能信号 0：不能写入 1：写入
din[31:0]	I	被写入的数据
addr[31:0]	I	写入/读出地址
dout[31:0]	O	读出数据

2.5.3 功能定义

序号	功能名称	功能描述
1	复位	复位信号有效时，数据存储器数据清零
2	写数据	写使能有效时，根据数据写入地址，将数据写入对应的存储器单元，采用小端序存储。
3	读数据	根据读取数据的地址，将数据读出并输出

2.6 MUX 模块

2.6.1 基本描述

该模块实现的主要功能为数据选择器，根据选择信号，选择对应的输入数据进行输出。

定义了两种数据选择器，分别是 32 位二选一数据选择器、5 位二选一数据选择器。

2.6.2 模块接口

信号名	方向	描述
sel	I	选择信号
din_0[4:0]/[31:0]	I	输入数据 1
din_1[4:0]/[31:0]	I	输入数据 2
dout[4:0]/[31:0]	O	输出数据

2.6.3 功能定义

序号	功能名称	功能描述
1	选择数据	根据选择信号，选择正确的输入数据并输出

2.7 EXT 模块

2.7.1 基本描述

根据选择信号指示的拓展类型，将 16 位立即数扩展至 32 位。扩展类型包括 0 扩展，符号扩展以及立即数扩展至高位、低位补 0 三种。

2.7.2 模块接口

信号名	方向	描述
imm16[15:0]	I	16 位立即数
ExtOp[1:0]	I	扩展类型选择信号 00：0 扩展 01：符号扩展 10：立即数扩展至高位，低位补 0
ext32[31:0]	O	扩展结果

2.7.3 功能定义

序号	功能名称	功能描述
1	扩展立即数	高 16 位 0 扩展、高 16 位符号扩展、立即数扩展至高 16 位并低 16 位补 0

三、指令描述

序号	助记符	opcode	funct	指令功能
1	addu	000000	100001	无符号加法
2	subu	000000	100011	无符号减法
3	ori	001101		或立即数
4	lw	100011		加载字
5	sw	101011		存储字
6	beq	000100		等于时转移
7	lui	001111		立即数加载至高位
8	j	000010		无条件跳转
9	addi	001000		加立即数（有符号）
10	addiu	001001		加立即数（无符号）
11	slt	000000	101010	小于时置1
12	jal	000011		跳转并链接
13	jr	000000	001000	跳转寄存器
14	bgezal	000001		大于等于0时跳转，并链接

四、测试程序

4.1 MIPS-Lite1 指令集的测试程序

```
ori $16, $0, 1  # 把1与0号寄存器进行或运算，结果存16号寄存器
ori $17, $0, 3  # 把3与0号寄存器进行或运算，结果存17号寄存器
ori $8, $0, 1   # 把1与0号寄存器进行或运算，结果存8号寄存器
ori $12, $0,0xabab      # 十六进制数abab与0号寄存器相或，结果存12号寄存器
lui $13, 10     # 把10扩展到32位数的高位，低位补0，存13号寄存器
start:addu $4, $0,$16   # 0号寄存器内容 和 16号寄存器内容 进行无符号加法，结果存4号寄存器
addu $5, $0,$8  # 8号寄存器内容 和 0号寄存器内容 进行无符号加法，结果存5号寄存器
jal newadd      # 跳转到newadd处，同时 （当前地址+4---下一条指令地址）的结果 存入31号寄存器
addu $16, $0, $2        # 2号寄存器内容 和 0号寄存器内容 进行无符号加法，结果存16号寄存器
subu $17,$17,$8         # 17号寄存器内容 和 8号寄存器内容 进行无符号减法，结果存17号寄存器
beq $16, $17, start     # 比较 16号寄存器内容 和 17号寄存器内容 ： 相等则跳转到start处
ori $8, $0,4            # 把4与0号寄存器相或，结果存8号寄存器
addiu $24,$0,0x7fffffff # 0号寄存器内容加立即数7fffffff（十六进制）的结果，存入24号寄存器 （无符号加）
addiu $9,$24,3          # 24号寄存器内容与3相加的结果（无符号加法）存9号寄存器
addiu $10,$24,5         # 24号寄存器内容与5进行无符号加法的结果，存10号寄存器
addu $0,$0,$0           # 0号寄存器与0号寄存器进行无符号加法的结果存0号寄存器。（理论上0号寄存器不能被写入）
#addi $22,$24,6         # 6与24号寄存器内容进行有符号加法，结果存入22寄存器，若产生溢出，30号寄存器存1 （实验要求）
start2:sw $9, 0($8)     # 以8号寄存器中的内容为基地址，加上偏移量0后的数据存储器地址的存储单元，存9号寄存器内容
lw $14, 0($8)           # 以8号寄存器中的内容为基地址，加上偏移量0后 的数据存储器地址的存储单元的内容，存入14号寄存器
sw $10,4($8)            # 以8号寄存器的内容为基地址，加上偏移量4后的数据存储器地址的存储单元，存10号寄存器内容
lw $15,4($8)            # 以8号寄存器的内容为基地址，加上偏移量4后的数据存储器地址的存储单元的内容，存入15号寄存器
sw $4, -4($8)           # 以8号寄存器的内容为基地址，加上偏移量-4后的数据存储器地址的存储单元，存4号寄存器内容
lw $18, -4($8)          # 以8号寄存器的内容为基地址，加上偏移量-4后的数据存储器地址的存储单元的内容，存入18号寄存器
addu $4,$0,$8           # 8号寄存器内容 与 0号寄存器内容 进行无符号加法，结果存入4号寄存器
addu $5,$0,$9           # 9号寄存器内容 与 0号寄存器内容 进行无符号加法，结果存入5号寄存器
```

```
jal newadd              # 将当前指令地址加4的内容（下一条指令地址）存入31号寄存器，跳转到newadd
slt $25,$10,$8          # 如果10号寄存器内容 小于 8号寄存器内容， 25号寄存器内容置1， 否则置0
beq $25, $0,end2        # 比较25号寄存器 与 0号寄存器内容（0）， 相等则跳转end2处，否则顺序执行
slt $20,$12,$4          # 如果12号寄存器内容 小于 4号寄存器内容， 20号寄存器内容置1， 否则置0
beq $20, $0, end1       # 比较20号寄存器 与 0号寄存器内容（0）， 相等则跳转end1处，否则顺序执行
lui $12, 65535          # 将十进制65535加载到32位的高16位，低16位补0， 扩展结果存入12号寄存器
end1:ori $0, $0,1       # 0号寄存器内容 与 1 相或， 结果存0号寄存器
lui $19, 0xefef         # 十六进制数efef加载至32位中的高16位，低位补0， 结果存入19号寄存器
addiu $3,$0,0xababcdcd  # 0号寄存器内容 无符号加立即数 ababcdcd（十六进制）， 结果存入3号寄存器中
start3:addiu $4, $3, 2  # 3号寄存器内容 无符号加立即数2， 结果存入4号寄存器中
addi $23, $3, 5         # 3号寄存器内容 加 5， 有符号数的加法， 结果存23号寄存器。若结果溢出，30号寄存器存1，否则为0
jal newadd              # 将下一条指令地址存入31号寄存器， 跳转到newadd处
addu $8, $0, $2         # 0号寄存器内容 与 2号寄存器内容 进行无符号加法， 结果存8号寄存器
addu $4, $0, $8         # 0号寄存器内容 与 8号寄存器内容 进行无符号加法， 结果存4号寄存器
addu $5, $0, $9         # 0号寄存器内容 与 9号寄存器内容 进行无符号加法， 结果存5号寄存器
jal newadd              # 将下一条指令地址存入31号寄存器， 跳转到newadd处
addu $9, $0, $2         # 将2号寄存器内容与0号寄存器内容进行无符号加法，结果存入9号寄存器
addu $9, $8, $0         # 将0号寄存器内容与8号寄存器内容进行无符号加法，结果存入9号寄存器
lui $10, 0x69           # 将十六进制数69加载到32位数的高16位，低位补0，结果存入10号寄存器
beq $8, $9, start4      # 比较8号寄存器 和 9号寄存器的内容， 相等跳转start4处，否则顺序执行
beq $0, $0, start3      # 比较0号寄存器 和 0号寄存器的内容， 相等跳转start3处，否则顺序执行
start4:j end            # 无条件跳转end处
newadd:addu $2, $4, $5  # 4号寄存器内容 无符号加 5号寄存器内容， 结果存入2号寄存器中
addi $0,$12,0x1234      # 十六进制数1234 和12号寄存器的内容 有符号加法， 结果存0寄存器（0寄存器理论上无法被写入）
jr $31                  # 跳转到 31号寄存器存储的内容 所指示的地址处
end2:addi $26,$0,0x5678 # 十六进制数5678 和 0号寄存器内容 有符号加法， 结果存26号寄存器
end:                    # end标志处
```

4.2 新增指令 BGEZAL 的测试程序

```
ori $1, $0, 3    # 3与0号寄存器内容相或，结果存1号寄存器
bgezal $1, jmpYes       # 如果1号寄存器内容大于等于0，则跳转jmpYes处
j end1          # 无条件跳转end1
jmpYes:ori $9, $0, 6  #【第一次对，$9 = 6】 0号寄存器与6相或，结果存9号寄存器
ori $2, $1, 5    # 1号寄存器内容与5相或，结果存2号寄存器
subu $1, $0, $2 # 0号寄存器内容 无符号减 2号寄存器内容，结果存1号寄存器
bgezal $1, jmpYess      # 如果1号寄存器内容大于等于0，则跳转jmpYess处
j end2          # 无条件跳转end2
jmpYess:ori $10, $0, 6 #【第二次对， $10=6】 0号寄存器与6相或，结果存10号寄存器
j end2          # 无条件跳转end2
end1:ori $11, $0, 9  #【第一次不对跳， $11=9】0号寄存器与9相或，结果存11号寄存器
j endd          # 无条件跳转endd
end2:ori $12, $0, 9  #【第二次不对跳， $12 = 9】0号寄存器与9相或，结果存12号寄存器
endd:
```

五、测试结果

5.1 MIPS-Lite1 指令集测试结果

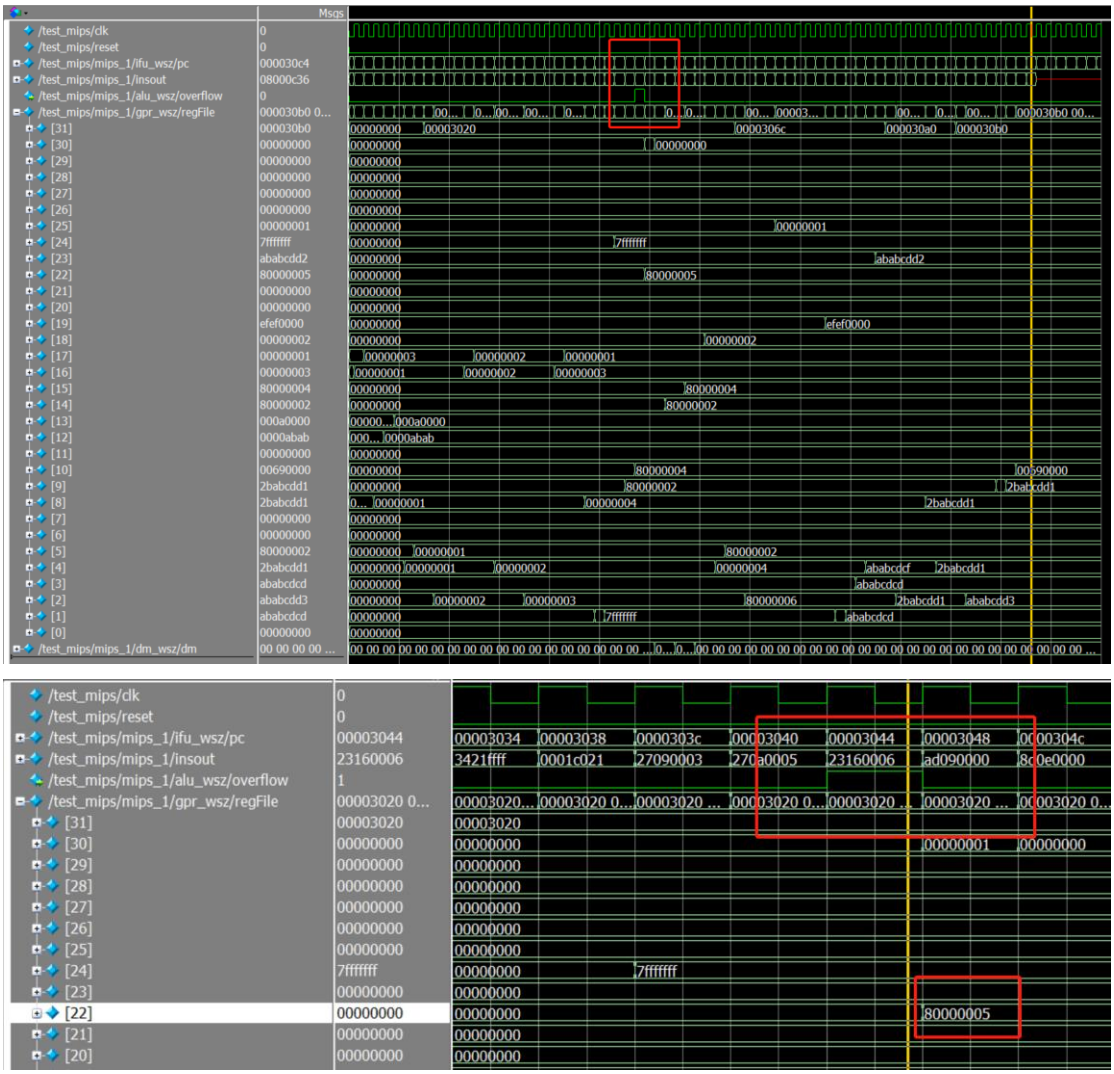
5.1.1 MARS 中结果

Name	Number	Value
$zero	0	0x00000000
$at	1	0xababcdcd
$v0	2	0xababcdd3
$v1	3	0xababcdcd
$a0	4	0x2babcdd1
$a1	5	0x80000002
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x2babcdd1
$t1	9	0x2babcdd1
$t2	10	0x00690000
$t3	11	0x00000000
$t4	12	0x0000abab
$t5	13	0x000a0000
$t6	14	0x80000002
$t7	15	0x80000004
$s0	16	0x00000003
$s1	17	0x00000001
$s2	18	0x00000002
$s3	19	0xefef0000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0xababcdd2
$t8	24	0x7fffffff
$t9	25	0x00000001
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x000030b0
pc		0x000030d8
hi		0x00000000
lo		0x00000000

Address	Value (+0)	Value (+4)	Value (+8)
0x00000000	0x00000002	0x80000002	0x80000004

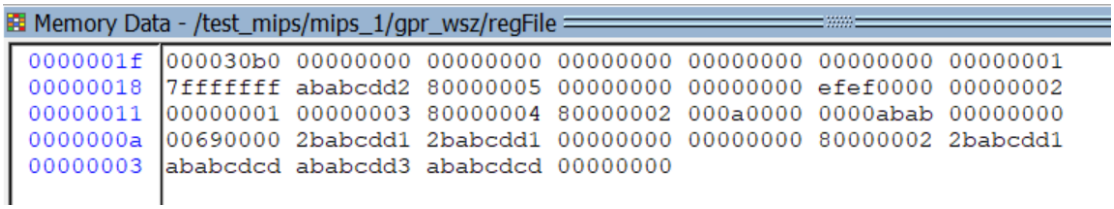
5.1.2 Logisim 中结果

（1）溢出波形

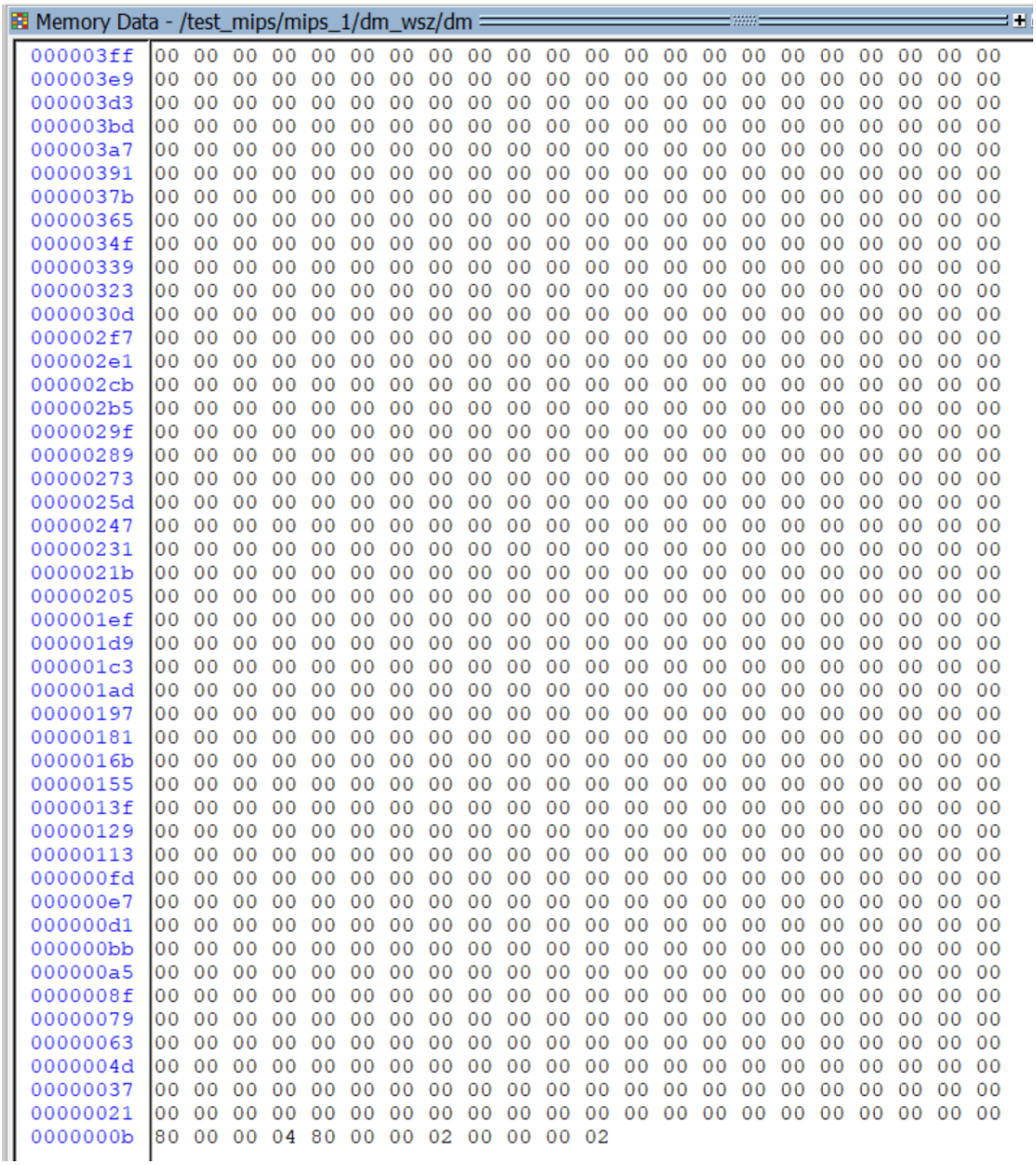


（2）寄存器 & 数据存储器内容

a. 寄存器



b. 数据存储器

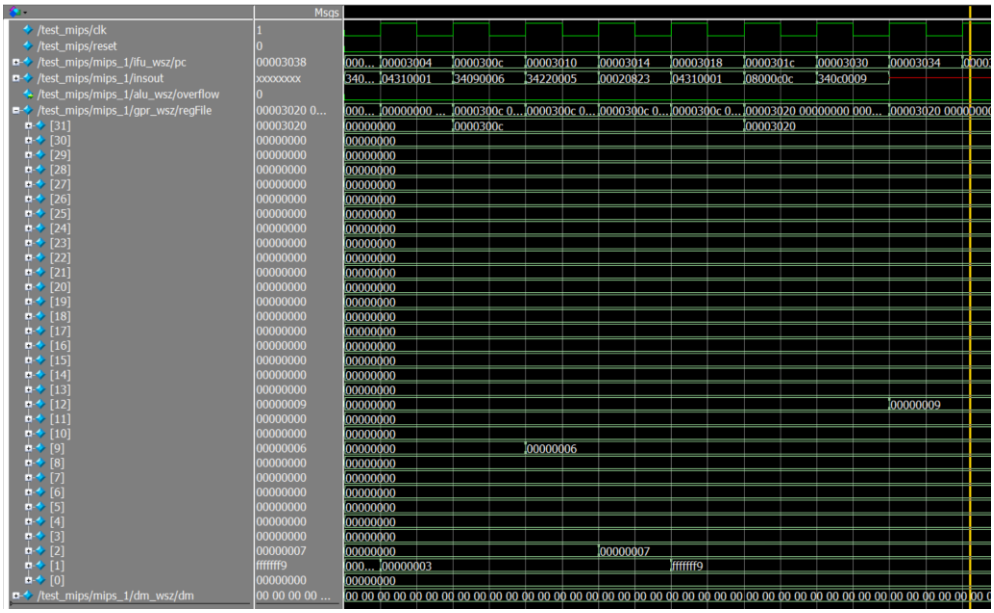

Memory Data - /test_mips/mips_1/dm_wsz/dm
000003ff 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000003e9 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000003d3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000003bd 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000003a7 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000391 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000037b 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000365 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000034f 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000339 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000323 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000030d 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000002f7 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000002e1 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000002cb 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000002b5 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000029f 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000289 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000273 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000025d 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000247 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000231 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000021b 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000205 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001ef 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001d9 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001c3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001ad 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000197 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000181 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000016b 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000155 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000013f 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000129 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000113 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000fd 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000e7 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000d1 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000bb 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000a5 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000008f 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000079 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000063 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000004d 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000037 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000021 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000b 80 00 00 04 80 00 00 02 00 00 00 02

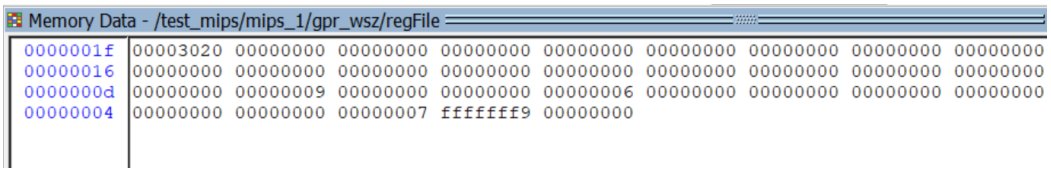
5.2 BGEZAL 指令测试结果

5.2.1 MARS 中结果

Name	Number	Value
$zero	0	0x00000000
$at	1	0xfffffff9
$v0	2	0x00000007
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000006
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000009
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x00003008
pc		0x00003034
hi		0x00000000
lo		0x00000000

5.2.2 Logisim 中结果





六、总结与心得体会

这次通过写 verilog 代码的方式，完成了单周期处理器的开发，支持 MIPS-Lite1 指令集，同时新添加了 bgezal 指令。

首先，通过这次开发，我对于开发一个设备的流程更加的清晰了。以这一次开发单周期处理器为例，在开发前，应该构思需要用到的功能，并根据需要的功能，将其模块化。在模块化的基础上，进行数据通路的设计和控制器的设计。在完成上述的设计以后，再进行 verilog 语言的编写，就会变得容易许多。在编写 verilog 时，应该注意每一个模块的输入输出的信号位宽，以及需要注意在顶层模块将各个模块连接到一起的时候，需要注意信号之间的对应问题。在本次编写中，我就曾因为模块之间的两个信号连反了，导致 debug 了一晚上都没有发现错误，十分的致命。这加强了我对这类错误的注意，以后在顶层模块中进行模块连接时，会更加的注意这一点。

其次，在开发过程中，我遇到的几个问题，我认为是值得总结的：

一，在完成对应指令功能的开发时，需要注意是有符号数的运算还是无符号数的运算。

二，在考虑 addi 的溢出问题时，是两个有符号数的加法，需要使用双符号位进行判断是否在加法过程中产生了溢出。

三，在更新 pc 值时，需要注意有一些指令是在 pc+4 的基础上更新的，这一点我在做新指令 bgezal 的时候忽略了，导致了 pc 跳转的错误。同时，bgezal 指令的功能是判断当前操作数（符号扩展）是否大于等于 0，也就是需要进行有符号数的判断和 0 之间的大小，在编写判断语句时注意需要进行的是有符号数之间的判断。

四，我发现相同类型的指令之间，其产生新 pc 值的逻辑有时会有些相似，如本次的 bgezal 指令实际上与 beq 指令的新 pc 值的产生方式是基本一致的。也就是说，其实可以进行一些变量的复用，使得程序代码更加简洁、高效。

最后，是我对此次大作业的体会。我认为通过这一次大作业，我更加深刻的理解了单周期 cpu 的数据通路的设计，同时对 mips 指令集中的代码更加的熟悉了，对于寄存器以及数据存储器中的操作变得不再害怕了。其次，我还明白了编程能力不仅仅体现在是否能使用代码完成自己设计思想的实现，更加重要的是编程的速度以及 debug 的能力。本次新增加指令时，我就由于改程序速度比较慢，再加上没有在规定时间内完成 bug 的修改，导致最后没有在规定时间内完成新增指令。这在一定程度上让我知道了需要改进的地方，我也一定会按照这个方向继续努力的。