



**计算机组成原理 课 程 设 计**

团队实验报告

|  |  |
| --- | --- |
| 学 院 | 计算机学院 |
| 专 业 | 计算机科学与技术 |
| 指导老师 | 王娟 |
| 组 长 | 胡玄哲 |
| 组 员 | 何宇轩、刘懿达、田东琦 |
| 组长联系方式 | QQ：1321992397 |

二O二 三 年 九 月

目 录

[第一章 项目简述 1](#_Toc83632503)

[第二章 组员分工（团队报告，个人报告不需要） 1](#_Toc83632504)

[第三章 设计目的 1](#_Toc83632505)

[第四章 设计环境 1](#_Toc83632506)

[第五章 设计原理及内容 2](#_Toc83632507)

[5.1 CPU整体架构 2](#_Toc83632508)

[5.1.1 数据通路 3](#_Toc83632509)

[5.1.2 控制逻辑 3](#_Toc83632510)

[5.2 流水线冒险问题以及解决方案 4](#_Toc83632511)

[第六章 设计与实现 5](#_Toc83632512)

[第七章 测试 9](#_Toc83632513)

[第八章 问题及解决方法 11](#_Toc83632514)

[第九章 心得体会及总结 12](#_Toc83632515)

[第十章 参考文献有价值的资源推荐 12](#_Toc83632516)

# 项目简述

实现经典五级标量流水线CPU，支持22条MIPS指令，包含冒险冲突处理，结合汇编与接口设计实验，设计测试用例，仿真测试完整，结果正确，并下板验证成功。

# 组员分工（团队报告，个人报告不需要）

小组分工见表2.1 小组分工。

|  |  |
| --- | --- |
| **组员** | **分工任务** |
| 组长：胡玄哲 | * 流水线处理器设计 * 接口程序设计 * 程序仿真测试 * PPT制作 * 报告撰写 |
| 组员：何宇轩 | * 仿真测试 * PPT制作 |
| 组员：刘懿达 | * 仿真测试 |
| 组员：田东琦 | * 仿真测试 * PPT制作 |

表2.1 小组分工

# 设计目的

实现支持MIPS指令子集的流水线CPU，仿真并在精工板上成功验证。

# 设计环境

|  |  |
| --- | --- |
| 操作系统 | Windows11 |
| 编程语言 | Verilog HDL |
| EDA工具 | Vivado2019.2 |
| 汇编语言 | MIPS |
| 汇编程序编辑器 | mars4\_5 |

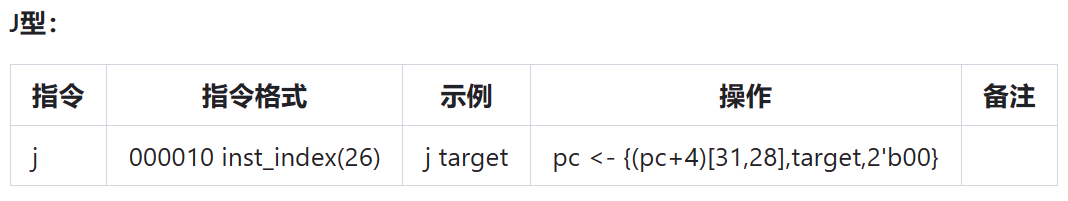
请标注版本号

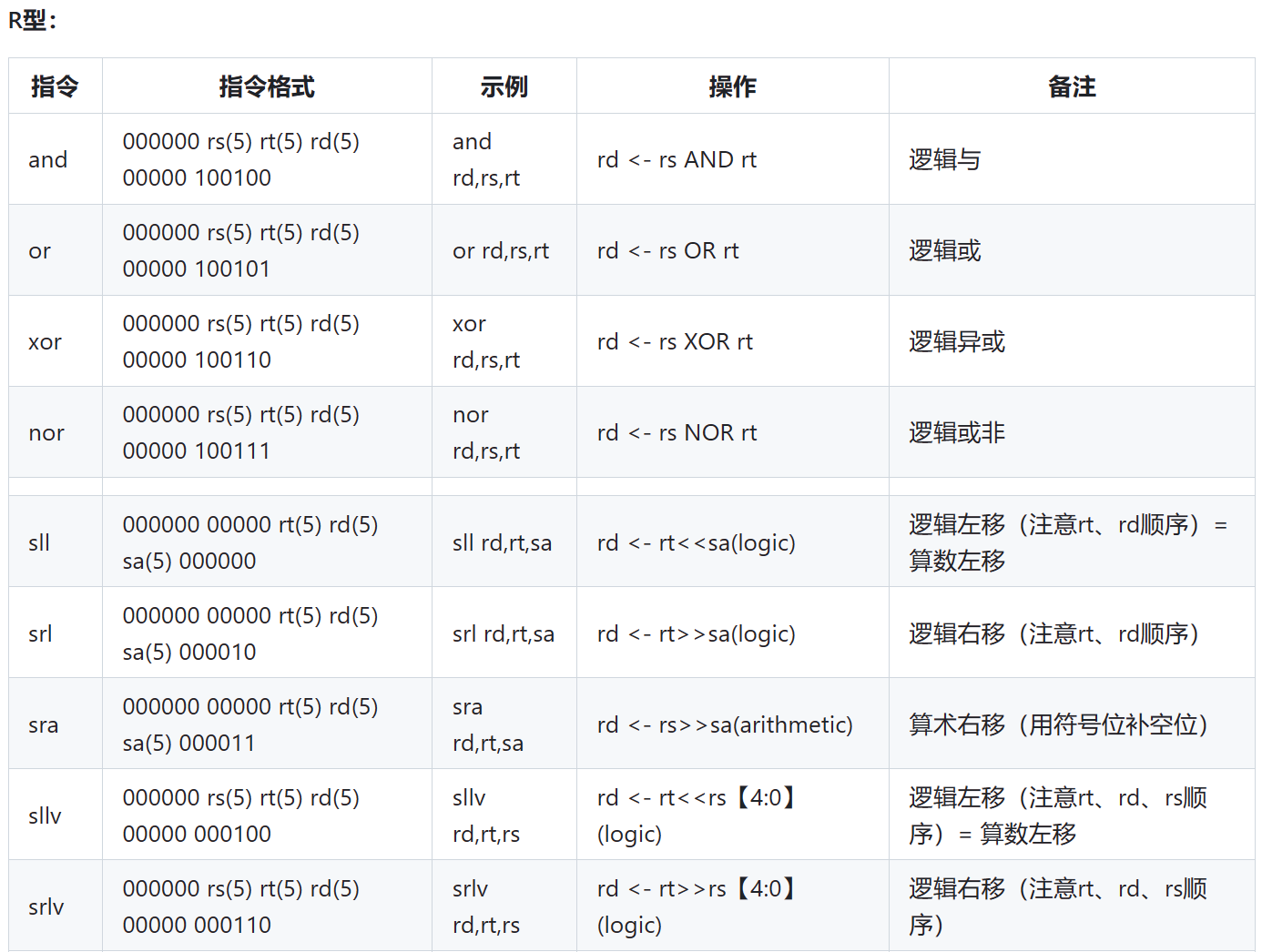
# 设计原理及内容

## CPU整体架构

本流水线CPU覆盖22条指令(见表5.1-1)，下面将从数据通路和控制逻辑阐述设计思路。







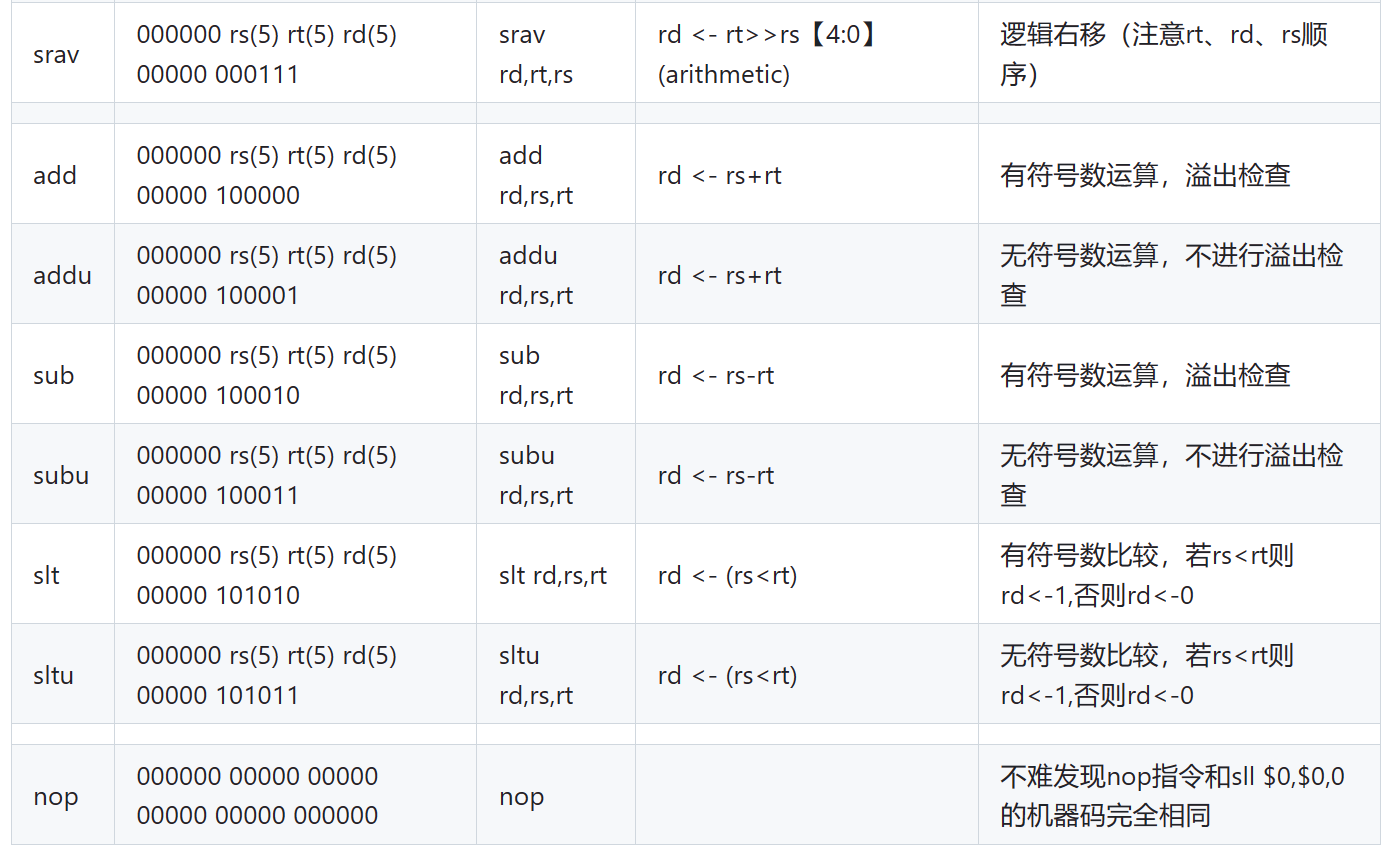


图5.1-1 22条指令

### 数据通路

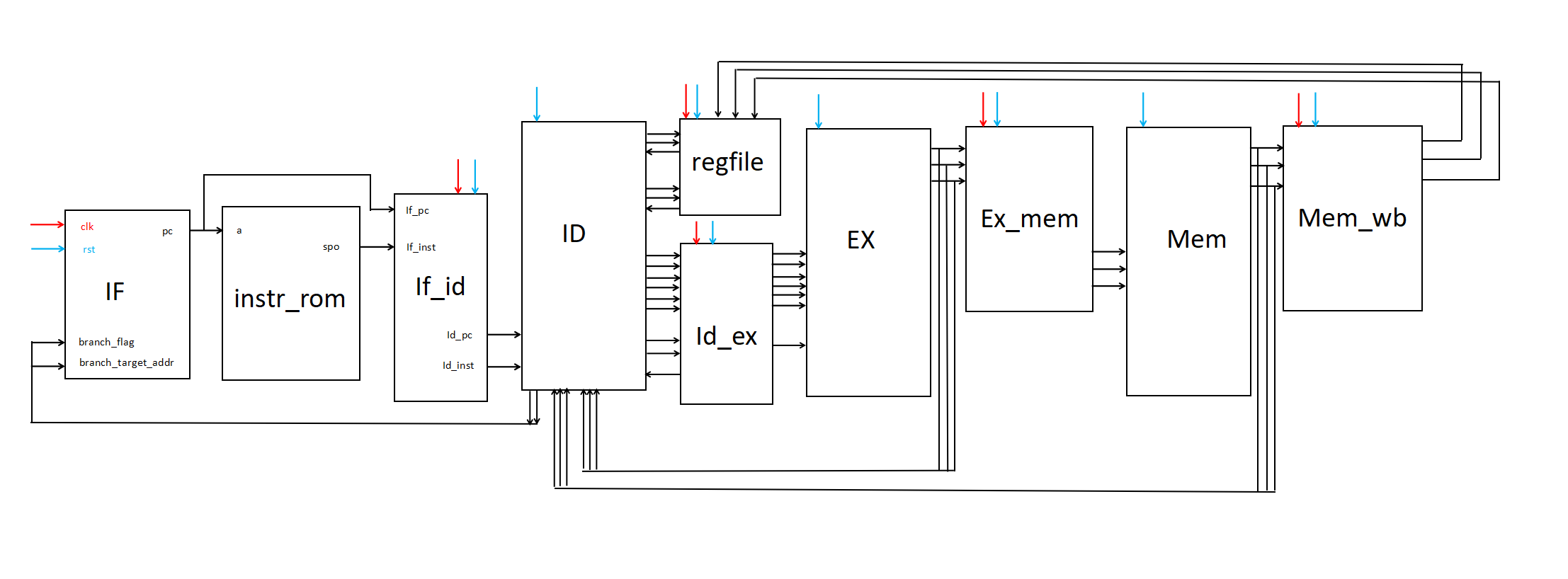


图5.1.1-1 5级流水线CPU数据通路

### 控制逻辑

**实现控制逻辑采用方式：**

id、ex、mem模块使用组合逻辑实现，其余模块使用时序逻辑实现。

**控制信号列表：**

**branch\_flag\_i：**跳转指令标志；

**ex\_wreg\_i：**数据旁路信号；

**mem\_wreg\_i：**数据旁路信号；

**in\_delayslot\_i：**延迟槽指令标志；

**aluop\_i：**指令的运算；

**alusel\_i：**结果的类型；

**we：**寄存器写使能；

**re1：**寄存器1端口读使能

**re2：**寄存器1端口读使能

**各阶段的控制信号：**

**取值阶段：**branch\_flag\_i；

**译码阶段：**ex\_wreg\_i、mem\_wreg\_i、in\_delayslot\_i；

**执行阶段：**aluop\_i、alusel\_i；

**写回阶段：**we、re1、re2；

## 流水线冒险问题以及解决方案

流水线冒险问题包括结构相关，数据相关和控制相关。

1. 结构相关：冯诺依曼结构中取指阶段和访存阶段对内存产生竞争，所以我们使用哈佛结构，分别使用一个ROM存储指令，和一个RAM存储数据。某些指令可能需要在译码阶段同时读两个寄存器，我们设计了两读一写寄存器堆来解决这种情况。
2. 数据相关：通过建立数据旁路解决了数据相关。将执行与访存阶段的待写回的数据直传译码段，译码段在取寄存器数据前先检查数据旁路。注意：ex离id近，先判断；mem离id远，后判断。顺序错误可能读到脏数据！
3. 控制相关：采用延迟槽的方式解决控制冒险；因为目前CPU仅支持 J 指令，因此为了节约时钟周期，将延迟槽判断从传统的ex段提前到了id段，这样只有 J 指令的后一个指令是延迟槽指令。在id读出跳转指令 J 之后，会将他的下一条指令置为延迟槽指令。为了实现这一点，可以先通过id\_ex流水寄存器暂存延迟槽标志，然后流水寄存器会在下一个时钟周期将延迟槽标志传回id段，此时进入id段的指令就会被标记为延迟槽指令；若一个指令被标记为延迟槽指令，它在后续的访存段和写回段的操作都会被无效化。这样就解决了控制冒险。

# 设计与实现

共设计并实现了10个模块，分别为

1. openmips：顶层模块，负责各个模块的连接，信号的选择，PC地址的运算；
2. pc\_reg：存储当前指令地址；
3. if\_id：流水寄存器；流水寄存器模块，位于取指段（instruction fetch）和译码段（instruction decode）间，存储pc和inst以实现id和if段的数据流水；流水寄存器功能：1.不reset时保存并传递信息到下一段；2.reset时将所有存储的信息复位并停止传递信息
4. id：指令译码；id段负责分析指令，依据指令特征字段区分指令，确定要读取的寄存器情况、要执行的运算、要写入的目的寄存器；代码变量命名的suffix，\_i表示输入，\_o表示输出
5. regfile：通用寄存器堆，共32个寄存器；同时是写回段的写回对象。寄存器堆模块，二读一写的特性是MIPS指令集的格式决定的，一个指令最多同时读取两个寄存器，最多同时写一个寄存器，故在模块中。共有三个读写端口。
6. id\_ex：流水寄存器；
7. ex：根据译码得出的指令执行相关操作；
8. ex\_mem：流水寄存器；
9. mem：访存段；
10. mem\_wb：流水寄存器；

各模块接口定义如下

1. module openmips(
2. input wire clk,
3. input wire rst,
5. input wire[`RegBus] rom\_data\_i, //从指令rom中取出的指令
6. output wire[`RegBus] rom\_addr\_o, //输出到指令rom的地址
7. output wire rom\_ce\_o, //指令rom的芯片使能信号
9. output wire[`RegBus] data\_o //输出到外设数据
10. );
11. module pc\_reg(
12. input wire clk,
13. input wire rst,
14. output reg[`InstAddrBus] pc, //pc存指令地址，指令地址宽度为32比特，4字节；
15. //pc会传送给指令ROM以取出指令。
16. output reg ce, //ce全程chip\_enable芯片使能，决定了指令ROM能否被访问。
18. //
19. input wire branch\_flag\_i,
20. input wire[`InstAddrBus] branch\_target\_addr\_i
21. );
22. module if\_id(
23. input wire clk,
24. input wire rst,
25. input wire[`InstAddrBus] if\_pc, //取值段取出的指令的地址
26. input wire[`InstBus] if\_inst, //取值段取出的指令
28. output reg[`InstAddrBus] id\_pc, //传递指令的地址
29. output reg[`InstBus] id\_inst //传递指令
30. );
31. module id(
32. input wire rst,
33. input wire[`InstAddrBus] pc\_i, //输入被译码的指令在指令ROM中的地址
34. input wire[`InstBus] inst\_i, //输入被译码的指令
36. //读取的Regfile的值
37. input wire[`RegBus] reg1\_data\_i, //输入寄存器堆读端口1的数据
38. input wire[`RegBus] reg2\_data\_i, //输入寄存器堆读端口2的数据
40. //输出到Regfile的值
41. output reg reg1\_read\_o, //输出寄存器堆读端口1的使能信号
42. output reg reg2\_read\_o, //输出寄存器堆读端口2的使能信号
43. output reg[`RegAddrBus] reg1\_addr\_o, //输出寄存器堆读端口1的读地址
44. output reg[`RegAddrBus] reg2\_addr\_o, //输出寄存器堆读端口2的读地址
46. //送到EX段的信息
47. output reg[`AluOpBus] aluop\_o, //输出运算子类型
48. output reg[`AluSelBus] alusel\_o, //输出运算类型
49. output reg[`RegBus] reg1\_o, //输出源操作数1，ALU的两个输入之一
50. output reg[`RegBus] reg2\_o, //输出源操作数2，ALU的两个输入之一
51. output reg wreg\_o, //输出写寄存器使能信号
52. output reg[`RegAddrBus] wDestRegAddr\_o, //输出写寄存器地址

55. //为了解决数据相关（只会出现写后读数据相关），建立数据旁路，将ex段和mem段的待写回数据直接传到id段
56. //1.ex段传回数据可以解决相邻指令的数据相关；
57. input wire ex\_wreg\_i,
58. input wire[`RegAddrBus] ex\_wDestRegAddr\_i,
59. input wire[`RegBus] ex\_wdata\_i,
60. //2.mem段传回数据可以解决相隔一条指令的数据相关
61. input wire mem\_wreg\_i,
62. input wire[`RegAddrBus] mem\_wDestRegAddr\_i,
63. input wire[`RegBus] mem\_wdata\_i,

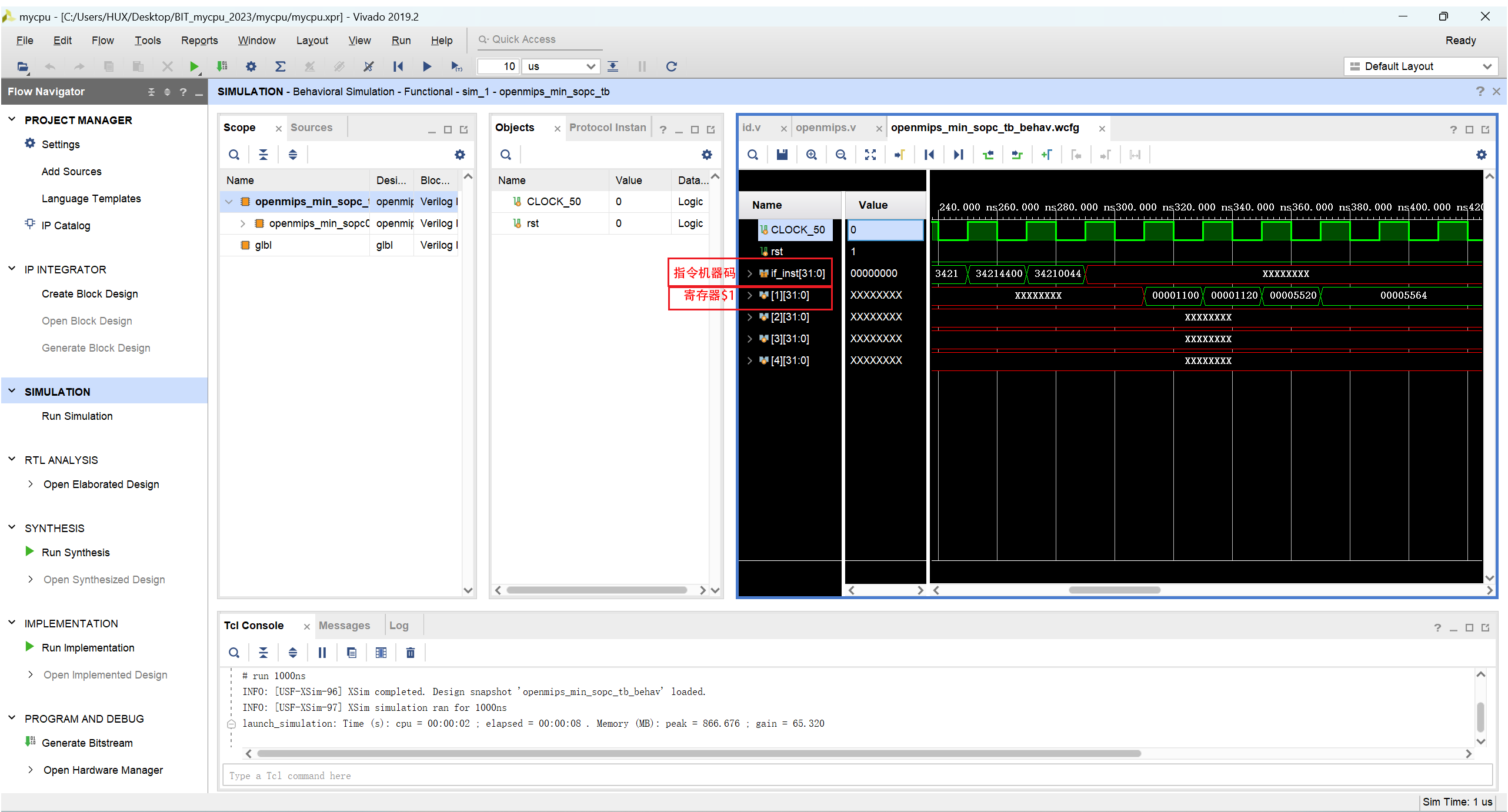

67. //指令跳转
68. output reg branch\_flag\_o,
69. output reg[`InstAddrBus] branch\_target\_addr\_o,
70. output reg in\_delayslot\_o,
71. output reg next\_inst\_in\_delayslot\_o,
72. input wire in\_delayslot\_i
74. );
75. module regfile(
76. input wire clk,
77. input wire rst,
79. //写回段：write port
80. input wire we, //写使能
81. input wire[`RegAddrBus] wRegAddr, //$0~$31
82. input wire[`RegBus] wdata, //要写入的数据
84. //译码段：read port1
85. input wire re1, //读使能1
86. input wire[`RegAddrBus] rRegAddr1, //$0~$31
87. output reg[`RegBus] rdata1, //要读出的数据
89. //译码段：read port2
90. input wire re2, //读使能2
91. input wire[`RegAddrBus] rRegAddr2, //$0~$31
92. output reg[`RegBus] rdata2, //要读出的数据
94. output reg[`RegBus] data\_o //
95. );
96. module id\_ex(
97. input wire clk,
98. input wire rst,
100. input wire[`AluOpBus] id\_aluop, //id传过来的aluop
101. input wire[`AluSelBus] id\_alusel, //id传过来的alusel
102. input wire[`RegBus] id\_reg1, //id传过来的ALU操作数1
103. input wire[`RegBus] id\_reg2, //id传过来的ALU操作数2
104. input wire[`RegAddrBus] id\_wDestRegAddr, //写寄存器地址和写使能信号会一直在流水寄存器中传递到写回段wb
105. input wire id\_wreg,
107. output reg[`AluOpBus] ex\_aluop, //传给ex的aluop
108. output reg[`AluSelBus] ex\_alusel, //传给ex的alusel
109. output reg[`RegBus] ex\_reg1, //传给ex的ALU操作数1
110. output reg[`RegBus] ex\_reg2, //传给ex的ALU操作数2
111. output reg[`RegAddrBus] ex\_wDestRegAddr, //写寄存器地址和写使能信号会一直在流水寄存器中传递到写回段wb
112. output reg ex\_wreg,
114. //
115. input wire id\_in\_delayslot,
116. input wire next\_inst\_in\_delayslot\_i,
117. output reg ex\_in\_delayslot,
118. output reg in\_delayslot\_o
119. );
120. module ex(
121. input wire rst,
122. input wire[`AluSelBus] alusel\_i,
123. input wire[`AluOpBus] aluop\_i,
124. input wire[`RegBus] src1\_i,
125. input wire[`RegBus] src2\_i,
126. input wire wreg\_i,
127. input wire[`RegAddrBus] wDestRegAddr\_i,
129. //执行段的输出结果
130. output reg wreg\_o,
131. output reg[`RegAddrBus] wDestRegAddr\_o,
132. output reg[31:0] wdata\_o, //执行阶段计算出写入寄存器的数据
134. //
135. input wire in\_delayslot\_i
137. );
138. module ex\_mem(
139. input wire clk,
140. input wire rst,
142. input wire[`RegAddrBus] ex\_wDestRegAddr,
143. input wire ex\_wreg,
144. input wire[`RegBus] ex\_wdata,
146. //输出
147. output reg[`RegAddrBus] mem\_wDestRegAddr,
148. output reg mem\_wreg,
149. output reg[`RegBus] mem\_wdata
150. );
151. module mem(
152. input wire rst,
154. input wire[`RegAddrBus] wDestRegAddr\_i,
155. input wire wreg\_i,
156. input wire[`RegBus] wdata\_i,
158. //输出
159. output reg[`RegAddrBus] wDestRegAddr\_o,
160. output reg wreg\_o,
161. output reg[`RegBus] wdata\_o
162. );
163. module mem\_wb(
164. input wire clk,
165. input wire rst,
167. input wire[`RegAddrBus] mem\_wDestRegAddr,
168. input wire mem\_wreg,
169. input wire[`RegBus] mem\_wdata,
171. //输出
172. output reg[`RegAddrBus] wb\_wDestRegAddr,
173. output reg wb\_wreg,
174. output reg[`RegBus] wb\_wdata
175. );

具体实现细节可参考源码验证。

# 测试

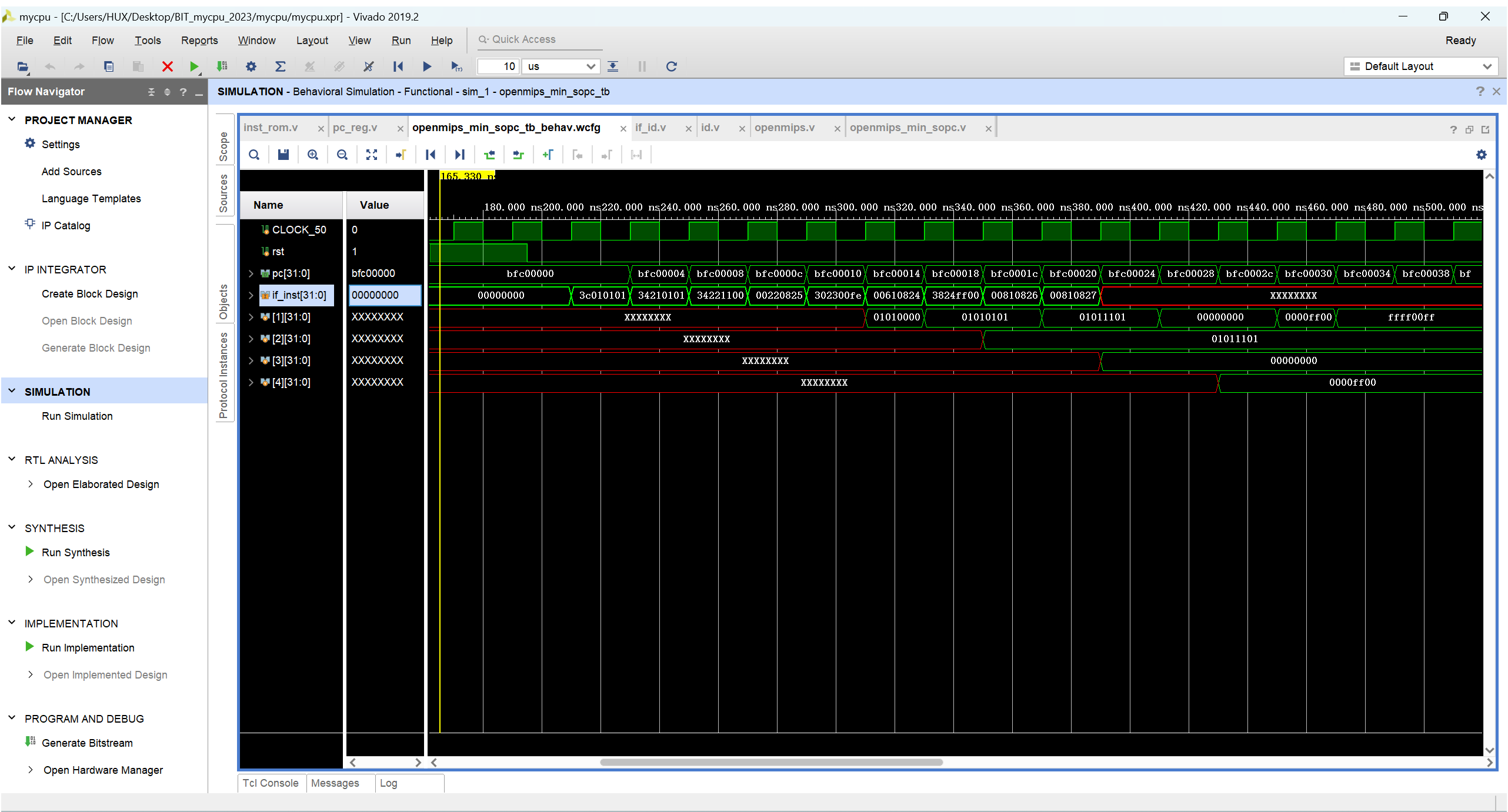
1.数据冒险

1. ori $1,$0,0x1100 # $1 = $0 | 0x1100 = 0x1100
2. ori $1,$1,0x0020 # $1 = $1 | 0x0020 = 0×1120
3. ori $1,$1,0x4400 # $1 = $1 | 0x4400 = 0×5520
4. ori $1,$1,0x0044 # $1 = $1 | 0x0044 = 0x5564



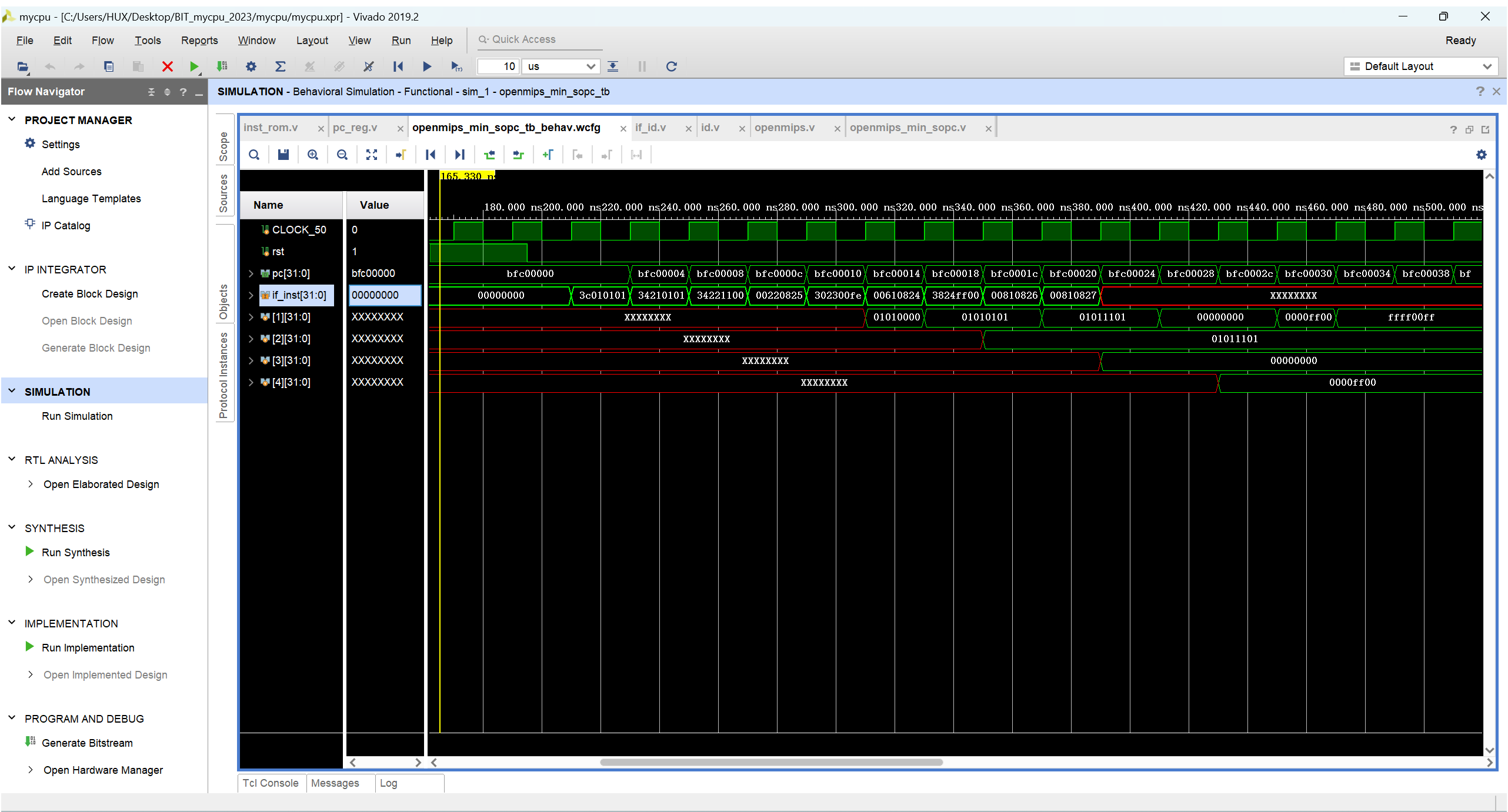
2.逻辑运算指令

1. lui $1,0x0101 #$1=0×01010000
2. ori $1,$1,0x0101 #$1=$1|0x0101=0x01010101
3. ori $2,$1,0x1100 #$2=$1|0x1100=0x01011101
4. or $1,$1,$2 #$1=$1|$2=0x01011101
5. andi $3,$1,0x00fe #$3=$1&0x00fe=0x00000000
6. and $1,$3,$1 #$1=$3&$1=0x00000000
7. xori $4,$1,0xff00 #$4=$1^0xff00=0x0000ff00
8. xor $1,$4,$1 #$1=$4^$1=0x0000ff00
9. nor $1,$4,$1 #$1=$4~^$1=0xffff00ff



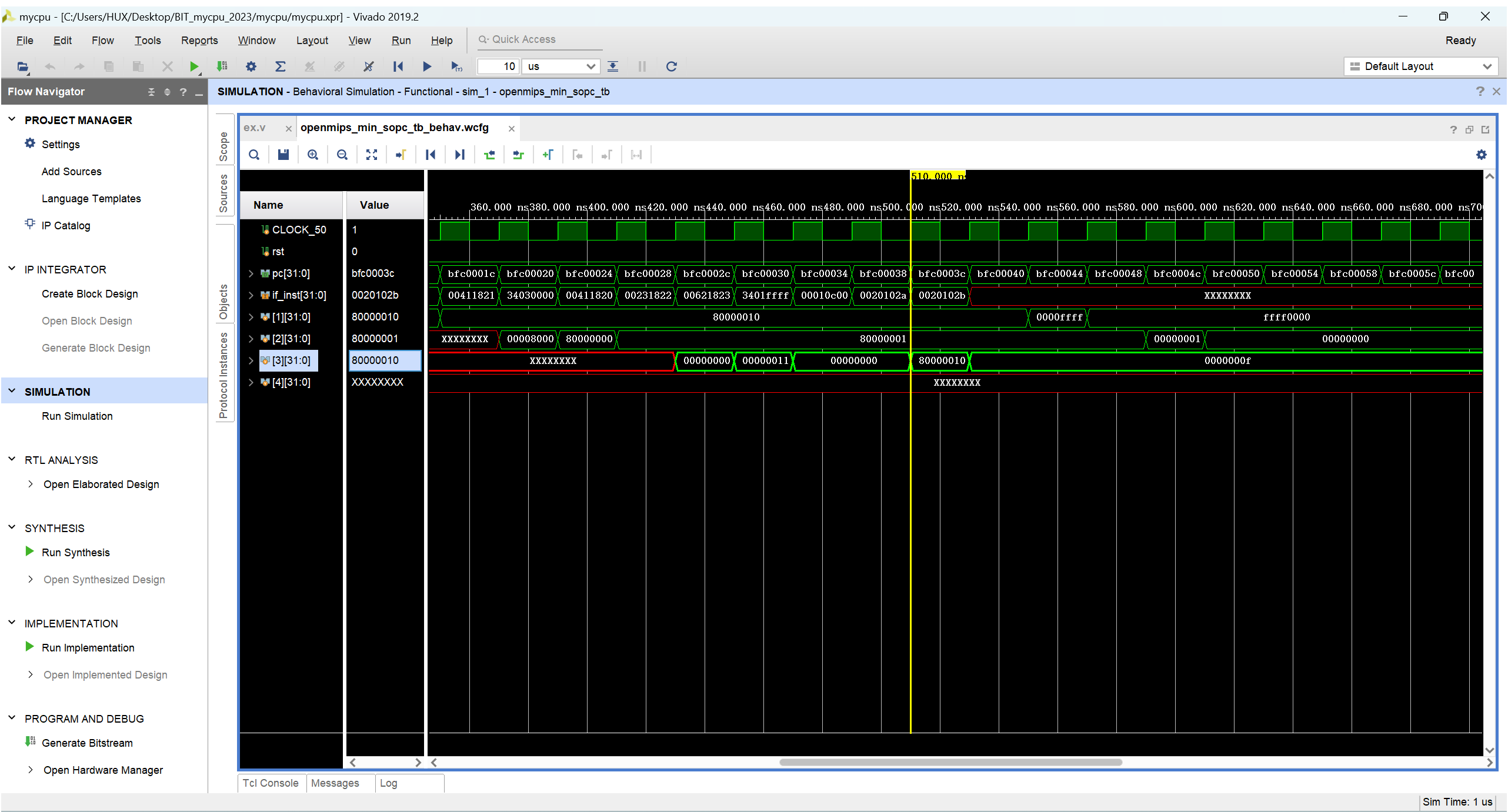
3.移位运算指令

1. lui $2,0x0404 #$2=0x04040000
2. ori $2,$2,0x0404 #$2=0x04040000|0x0404= 0x04040404
3. ori $7,$0,0x7
4. ori $5,$0,0x5
5. ori $8,$0,0x8
6. nop
7. sll $2,$2,8 #$2=0x40404040 sll 8= 0x04040400
8. sllv $2,$2,$7 #$2=0x04040400 s11 7= 0x02020000
9. srl $2,$2,8 #$2=0x02020000 sr1 8= 0x00020200
10. srlv $2,$2,$5 #$2=0x00020200 srl 5= 0x00001010
11. nop
12. nop
13. sll $2,$2,19 #$2=0×00001010 sll 19= 0x80800000
14. nop
15. sra $2,$2,16 #$2=0x80800000 sra 16= 0xffff8080
16. srav $2,$2,$8 #$2 0xffff8080 sra 8= 0xffffff80



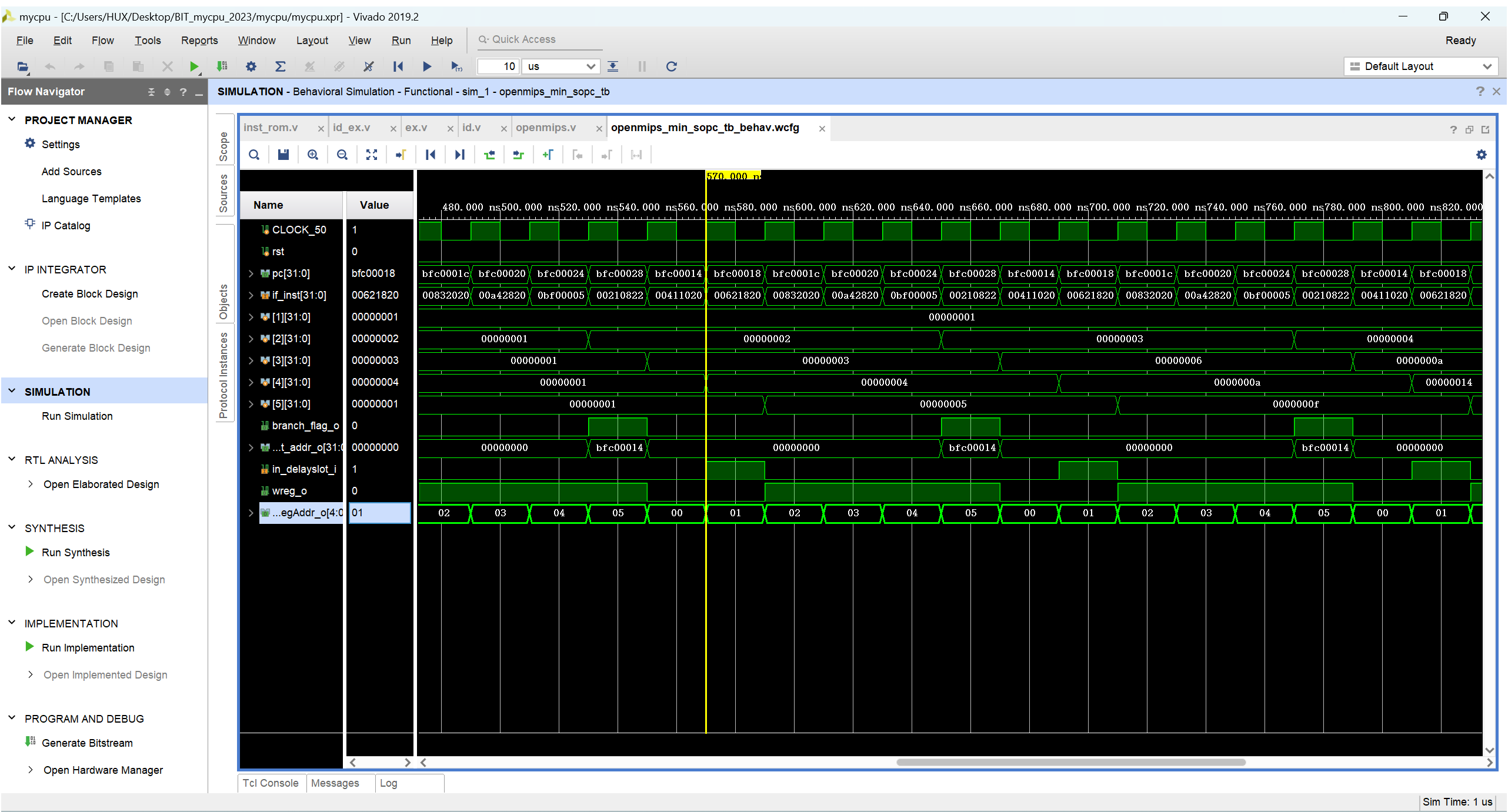
4.算数运算指令

1. ori $1,$0,0x8000 #$1=0x00008000
2. sll $1,$1,16 #$1=0x80000000
3. ori $1,$1,0x0010 #$1=0×80000010 给$1赋初值
4. ori $2,$0,0x8000 #$2=0x00008000
5. sll $2,$2,16 #$2=0x80000000
6. ori $2,$2,0x0001 #$2=0x80000001 给$2赋初值
7. ori $3,$0,0x0000 #$3=0x00000000
8. addu $3,$2,$1 #$3=0x00000011 $1加$2，无符号加法
9. ori $3,$0,0x0000 #$3=0x00000000
10. add $3,$2,$1 #$2加$1，有符号加法，结果溢出，所以$3应保持不变,$3保持为0x00000000
11. sub $3,$1,$3 #$3=0x80000010 $1减去$3，有符号减法
12. subu $3,$3,$2 #$3=0xF $3减去$2，无符号减法
13. or $1,$0,0xffff #$1=0x0000ffff
14. sll $1,$1,16 #$1=0xffff0000 给$1赋初值
15. slt $2,$1,$0 #$2=1 比较$1与0x0,有符号比较
16. sltu $2,$1,$0 #$2=0 比较$1与0x0,无符号比较



5.跳转指令

1. ori $1,$0,1 #$1 = 1
2. ori $2,$0,0 #$2 = 0
3. ori $3,$0,0 #$3 = 0
4. ori $4,$0,0 #$4 = 0
5. ori $5,$0,0 #$5 = 0
6. loop:
7. add $2, $2, $1
8. add $3, $3, $2
9. add $4, $4, $3
10. add $5, $5, $4
11. j loop
12. sub $1, $1, $1



# 问题及解决方法

1. 问题：数据旁路读出脏数据。

解决：ex离id近，先判断；mem离id远，后判断。顺序错误可能读到脏数据！

1. 问题：溢出判断。

解决：先将所有运算变为加法运算，然后根据 负加负得正 或 正加正得负 的规则来判断是否发生溢出。如果溢出标志overflow为1，那么该运算的结果无效。

1. 问题：算术右移。

解决：例：1100算术右移一位：

1.先将原数逻辑右移一位（0110）

2.再将全为符号位的数左移三位（1000）

3.两数相或即所得结果

# 心得体会及总结

1.在MIPS架构中，$0寄存器不让写，是因为很多指令的机器码在寄存器字段都包含00000，但是其并不指代$0寄存器，为了避免混淆而规定$0寄存器不让写。  
2.有符号运算较为复杂，溢出判断的实现参考了网络资料。  
3.测试跳转指令时需要手动计算一下跳转地址，如测试文件inst\_rom4，需要跳转到 pc = bfc00014（1011 1111 1100 0000 0000 0000 0001 0100）处，则机器码为 obf00005（000010 + 1111 1100 0000 0000 0000 0001 01） 。

# 参考文献有价值的资源推荐

无