P5 实验报告

一、 数据通路

1．顶层视图

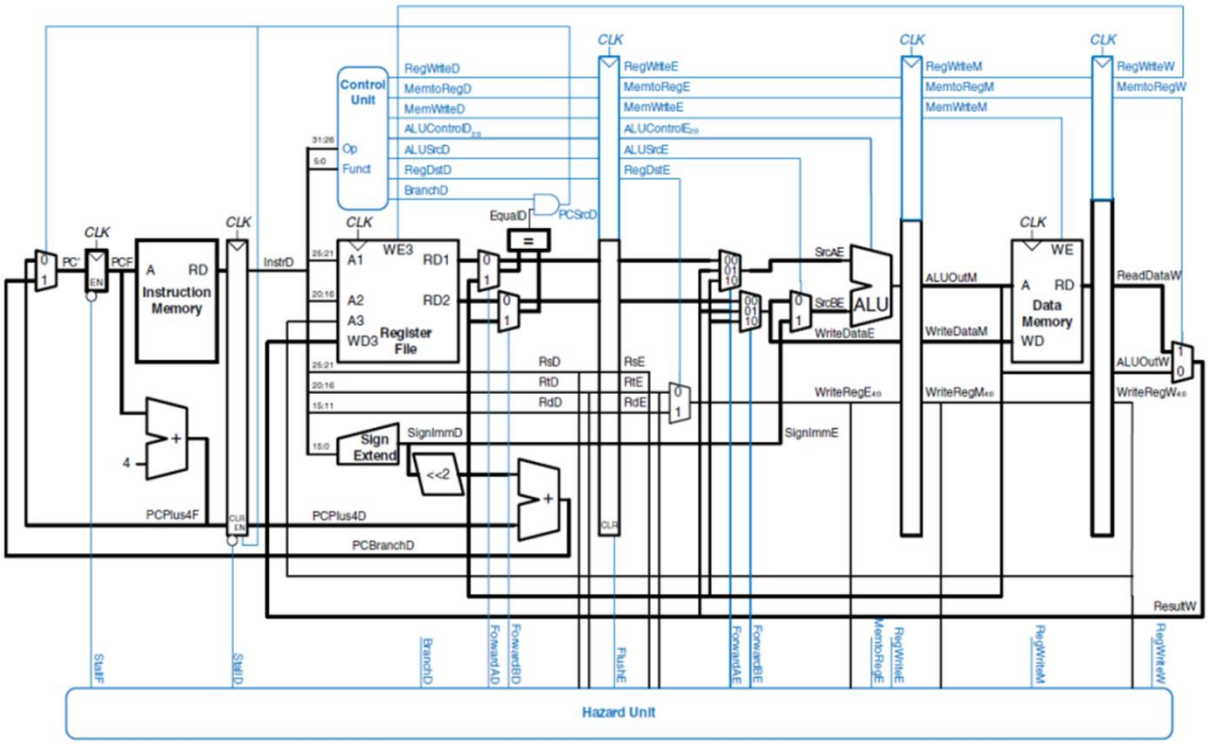


Figure 7.58 Pipelined processor with full hazard handling

2. 构造数据通路：

根据《数字设计和计算机体系结构》中的图 7-58 顶层视图。

级别	部件	输入	输入来源				MUX	控制信号	lw	sw	addu	subu	ori	lui	beq	j	jal	jr	nop
IF级部件	PC		ADD4	D对PC的更新					ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4
	ADD4								PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC
	IM								PC	PC	PC	PC	PC	PC	PC	PC	PC	PC	PC
F级对PC更新	PC								ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4	ADD4
F/D级流水线寄存器	IR_D								IM	IM	IM	IM	IM	IM	IM	IM	IM	IM	IM
	PC4_D																ADD4		
	PC8_D																		
D级部件	RF	Read1							IR_D[rs]	IR_D[rs]	IR_D[rs]	IR_D[rs]	IR_D[rs]		IR_D[rs]			IR_D[rs]	
		Read2								IR_D[rt]	IR_D[rt]	IR_D[rt]			IR_D[rt]				
	CMP	D1	ALUout_M	Read1	WD		MUX_F_CMPD1	F_CMPD1							Read1				
		D2	ALUout_M	Read2	WD		MUX_F_CMPD2	F_CMPD2							Read2				
	EXT								IR_D[i16]	IR_D[i16]			IR_D[i16]	IR_D[i16]	IR_D[i16]				
	NPC	PC4													PC4_D	PC4_D-4	PC4_D-4		
		IMM26													IR_D[i16]	IR_D[i26]	IR_D[i26]		
D级对PC更新	PC		NPC	Read1											NPC	NPC	NPC	Read1	
D/E级流水线寄存器	IR_E								IR_D	IR_D	IR_D	IR_D	IR_D	IR_D	IR_D	IR_D	IR_D	IR_D	IR_D
	PC4_E																PC4_D		
	PC8_E																		
	RS_E								RF_RD1	RF_RD1	RF_RD1	RF_RD1	RF_RD1						
	RT_E									RF_RD2	RF_RD2	RF_RD2							
	EXT_E								EXT	EXT			EXT	EXT					
E级部件	ALU	A	ALUout_M	Read1	WD		MUX_F_ALUA	F_ALUA	RS_E	RS_E	RS_E	RS_E	RS_E						
		B	ALUout_M	Read2	WD		MUX_F_ALUB	F_ALUB	EXT_E	EXT_E	RT_E	RT_E	EXT_E	EXT_E					
E/M级流水线寄存器	IR_M								ID_E	ID_E	ID_E	IR_E	IR_E	IR_E	IR_E	IR_E	IR_E	IR_E	IR_E
	PC4_M																PC4_E		
	PC8_M																		
	ALUout_M								ALUout	ALUout	ALUout	ALUout	ALUout	ALUout					
	RT_M									RF_RD2									
M级部件	DM	RA	ALUout_M						ALUout_M	ALUout_M									
		WD	RT_M	DM_W			MUX_F_WD	F_WD		RT_M									
M/W级流水线寄存器	IR_W								IR_M	IR_M	IR_M	IR_M	IR_M	IR_M	IR_M	IR_M	IR_M	IR_M	IR_M
	PC4_W																PC4_M		
	PC8_W																		
	ALUout_W										ALUout	ALUout	ALUout	ALUout					
	DM_W								DM										
W级功能部件	EXT_DM	A																	
		DATA																	
	RF	WRA							IR_W[rt]		IR_W[rd]	IR_W[rd]	IR_W[rt]	IR_W[rt]			31		
		WD							DM_W		ALUout_W	ALUout_W	ALUot_W	ALUot_W			PC4_W+4		

二、 模块定义

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(程序计数器)、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 指令的执行情况决定顺序取指令还是转移取指令。

(2) 模块接口

表 1 模块接口

信号名	方向	描述
PCBranch	I	输入 PC 要跳转几条指令
Branch	I	输入确定 PC 是否要跳转至 PC+4+PCBranch
Reset	I	是否清零
Instr	0	输出指令二进制编码

（3）功能定义

表 2 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 0x00003000
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 PCbranch 有效，那么 PC←PC+4+Branch 如果当前 PCBranch 为 0，那么 PC←PC+4

2.GRF 模块定义:

(1)基本描述

GRF 主要功能是完成寄存器的写入和读出操作。GRF 内部包括 32 个寄存器以及其他相关逻辑。其中 0 号寄存器的值始终保持为 0。GRF 可以根据输入来判断写入哪个寄存器或者读出哪个寄存器。

(2)模块接口

表 3 模块接口

信号名	方向	描述
reset	I	复位信号，将 32 个寄存器中的值全部清零
RegWrite	I	写使能信号
rs	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出到 RD1
rt	I	5 位地址输入信号，指定 32 个寄存器中的一个，将其中存储的数据读出到 RD2
rt/rd	I	5 位地址输入信号，指定 32 个寄存器中的一个作为写入的目标寄存器
WData	I	32 位数据输入信号
RD1	O	输出 A1 指定的寄存器中的

		32 位数据
RD2	O	输出 A1 指定的寄存器中的 32 位数据

（3）功能定义

表 4 功能定义

序号	功能名称	功能描述
1	复位	reset 信号有效时，所有寄存器存储的数值清零
2	读数据	读出 rs，rt 地址对应寄存器中所存储的数据到 RD1，RD2
3	写数据	当 RegWrite 有效且时钟上升沿来临时，将 WData 写入 WAddr 所对应的寄存器中

3.ALU 模块定义:

(1)基本描述

ALU 是算数逻辑单元，读入两个 32 位的数，之后根据 ALUop 来判断是什么运算。其中包括加、减、比较大小、与或等运算。

(2)模块接口

表 5 模块接口

信号名	方向	描述
ALU_A	I	第一个值
ALU_B	I	第二个值
ALUop	I	判断进行什么运算
ALUOut	O	运算结果

（3）功能定义

表 6 功能定义

序号	功能名称	功能描述
1	与	A&B

2	或	A|B
3	加	A+B
4	减	A-B

4. EXT 模块定义:

(1)基本描述

EXT 是位数拓展操作，它可以把一个 16 位的数拓展至 32 位。

(2)模块接口

表 7 模块接口

信号名	方向	描述
EXTIn	I	一个 16 位的数
EXTOp	I	判断拓展的形式
EXTOut	O	输出拓展后的数

（3）功能定义

表 8 功能定义

序号	功能名称	功能描述
1	无符号拓展	把输入的 16 位数拓展至无符号的 32 位数
2	有符号拓展	把输入的 16 位数拓展至有符号的 32 位数
3	拓展至高 16 位	将原有的 16 位数简单赋值到 32 位，新数的高 16 位，低 16 位用 0 填充

5. DM 模块定义:

(1)基本描述

DM 是数据存储器，它可以存储数据，并在想读出来的时候把他读出来。

(2)模块接口

表 9 模块接口

信号名	方向	描述

MemAddr	I	要写入的地址
MemData	I	要写入的数据
MemWrite	I	是否写入数据
MemOut	O	读出的数据

（3）功能定义

表 10 功能定义

序号	功能名称	功能描述
1	写入数据	确定要写入的数据存储的地址，进而 MemWrite 为 1 时，写入数据；为 0 时不写入
2	读出数据	根据存储器相应地址输出数据

6. Controller 模块定义：

(1)基本描述

Controller 是控制器，我们读入指令的高 6 位和 function 位来判断是什么指令，进而，我们可以确定各个板块的使能端控制端的值。

(2)模块接口

表 9 模块接口

信号名	方向	描述
op[5:0]	I	高 6 位
func[5:0]	I	Function
RegWrite	O	要不要写入寄存器
RegDst	O	写寄存器的目标寄存器号来源
ALUSrc	O	选择是运算扩展后的立即数还是寄存器内的值
Branch	O	与 ALU 零输出结合决定 PC 是否跳转
MemWrite	O	要不要写入存储器

MemtoReg	0	写入寄存器的数据来源：ALU 计算后的或者 DM 输出的
EXTOp	0	判断进行拓展的方式
ALUOp	0	判断 ALU 要进行什么运算
j	0	判断是否为 j 指令
jal	0	判断是否为 jal 指令
jr	0	判断是否为 jr 指令

（3）功能定义

表 10 功能定义

序号	功能名称	功能描述
1	判断是什么指令	我们根据传进来的两组数据，根据 MIPS 对应规则，可以知道他是什么指令。
2	判断是否要写入寄存器	如果 RegWrite 为 1 的话，就可以写入寄存器；否则不写入
3	判断要写入哪个寄存器	如果 RegDst 为 1 的话，就写入 11-15 位的寄存器，否则就写入 16-20 位的寄存器
4	判断要运算立即数还是寄存器的值	如果 ALUSrc 为 1 的话，就会选择立即数进行运算，否则，选择读出的寄存器的值进行运算
5	判断要不要跳转	如果当前指令需要跳转，比如 beq，那么 Branch 就为 1，否则为 0
6	判断要不要写入寄存器	如果 MemWrite 为 1 的话，就写入存储器，否则，不写入

7	判断写入的数据是哪个值	如果要写入寄存器的值是 ALU 算出来的时候，那么 MemtoReg 就为 0，否则为 1
8	判断要进行拓展的方式	比如如果 ExtOp 为 1，就进行有符号拓展
9	判断要进行那个运算	生成 ALUOp 来确定要让 ALU 进行那种运算。

三、控制器设置

1. 真值表:

func	100001	100011							000011	
op	000000	000000	001101	100011	101011	000100	001111	000011	000000	000010
	addu	subu	ori	lw	sw	beq	lui	jal	jr	j
EXTOp	xx	xx	00	01	01	01	10	00	00	00
MemtoReg	0	0	0	1	0	0	0	0	0	0
MemWrite	0	0	0	0	1	0	0	0	0	0
Branch	0	0	0	0	0	1	0	0	0	0
ALUOp	010	011	001	010	010	xxx	100	000	000	000
ALUSrc	0	0	1	1	1	0	1	0	0	0
RegDst	1	1	1	1	0	0	1	0	0	0
RegWrite	1	1	1	1	0	0	1	1	0	0
j	0	0	0	0	0	0	0	1	0	1
jal	0	0	0	0	0	0	0	1	0	0
jr	0	0	0	0	0	0	0	0	1	0

2、表达式:

EXTOp[0] = lw||sw||beq EXTOp[1] = lui

EXTOp[0] =

op[0]&op[1]&!op[2]&!op[3]&!op[4]&op[5]+

op[0]&op[1]&!op[2]&
op[3]&!op[4]&op[5]+ !op[0]&!op[1]&op[2]&!op[3]&!o
p[4]&!op[5]

EXTOp[1] = op[0]&op[1]&op[2]&op[3]&!op[4]&!op[5]

MemtoReg = lw

MemtoReg =
op[0]&op[1]&!op[2]&!op[3]&!op[4]&!op[5]

Memwrite = sw

MemWrite = op[0]&op[1]&!op[2]&op[3]&!op[4]&op[5]

Branch = beq

Branch = !op[0]&!op[1]&op[2]&!op[3]&!op[4]&!op[5]

ALUOp[0] = subu||ori ALUOp[1] =
addu||subu||lw||sw ALUOp[2] = lui

ALUOp[0]
= !op[0]&!op[1]&!op[2]&!op[3]&!op[4]&!op[5]&fun[0
]&fun[1]&!fun[2]&!fun[3]&!fun[4]&fun[5]+ !op[0]&!
op[1]&op[2]&!op[3]&!op[4]&!op[5]

ALUOp[1]
= !op[0]&!op[1]&!op[2]&!op[3]&!op[4]&!op[5]
+op[0]&op[1]&!op[2]&!op[3]&!op[4]&!op[5]+
op[0]&op[1]&!op[2]&op[3]&!op[4]&op[5]

ALUOp[2] = op[0]&op[1]&op[2]&op[3]&!op[4]&!op[5]

ALUSrc = ori||lw||sw||lui

ALUSrc = op[0]&!op[1]&op[2]&op[3]&!op[4]&!op[5]+
op[0]&op[1]&!op[2]&!op[3]&!op[4]&op[5]+
op[0]&op[1]&!op[2]&op[3]&!op[4]&op[5]+
op[0]&op[1]&op[2]&op[3]&!op[4]&!op[5]

RegDst = addu||subu

RegDst
= !op[0]&!op[1]&!op[2]&!op[3]&!op[4]&!op[5]

RegWrite = addu||subu||ori||lw||lui

RegWrite

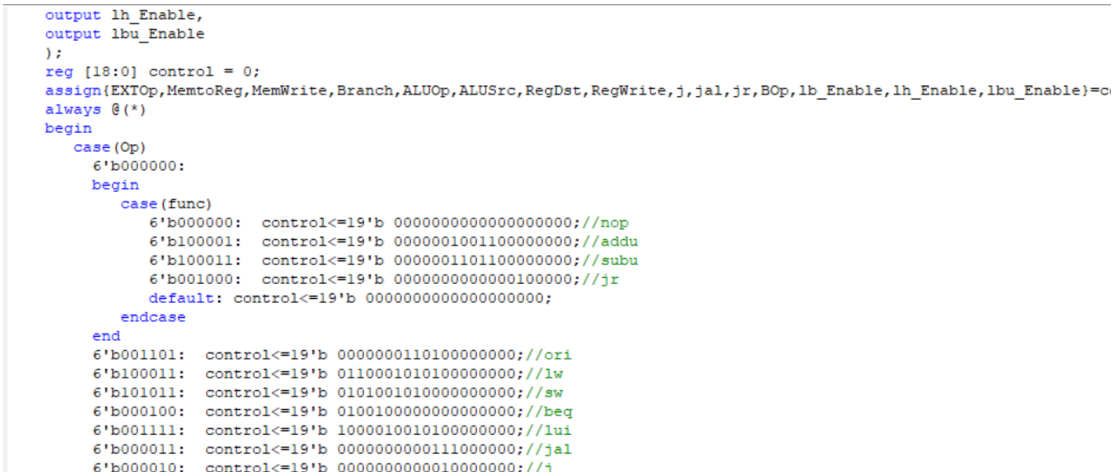
= !op[0]&!op[1]&!op[2]&!op[3]&!op[4]&!op[5]+

op[0]&!op[1]&op[2]&op[3]&!op[4]&!op[5]+

op[0]&op[1]&!op[2]&!op[3]&!op[4]&!op[5]+

op[0]&op[1]&op[2]&op[3]&!op[4]&!op[5]

3、具体实现方式



```
output lh_Enable,
output lbu_Enable
);
reg [18:0] control = 0;
assign{EXTOp,MemtoReg,MemWrite,Branch,ALUOp,ALUSrc,RegDst,RegWrite,j,jal,jr,BOp,lb_Enable,lh_Enable,lbu_Enable}=c
always @(*)
begin
   case(Op)
     6'b000000:
     begin
        case(func)
           6'b000000:  control<=19'b 0000000000000000000;//nop
           6'b100001:  control<=19'b 0000001001100000000;//addu
           6'b100011:  control<=19'b 0000001101100000000;//subu
           6'b001000:  control<=19'b 0000000000000100000;//jr
           default: control<=19'b 0000000000000000000;
        endcase
     end
     6'b001101:  control<=19'b 0000000110100000000;//ori
     6'b100011:  control<=19'b 0110001010100000000;//lw
     6'b101011:  control<=19'b 0101001010000000000;//sw
     6'b000100:  control<=19'b 0100100000000000000;//beq
     6'b001111:  control<=19'b 1000010010100000000;//lui
     6'b000011:  control<=19'b 0000000000111000000;//jal
     6'b000010:  control<=19'b 0000000000010000000;//j
```

4、 转发暂停模块

首先引入供给时间 Tnew 和需求时间 Tuse 两个信号。需求时间 Tuse 是指某条指令位于 D 级的时候，再经过多少个时钟周期就必须要使用相应的数据。它具有两个特点，特点 1：是一个定值，每个指令的 Tuse 是一定的特点。特点 2：一个指令可以有两个 Tuse 值。供给时间 Tnew 是指位于某个流水级的某个指令，它经过多少个时钟周期可以算出结果并且存储到流水级寄存器里。它也具有两个特点，特点 1：是一个动态值，每个指令处于流水线不同阶段有不同的 Tnew 值，特点 2：一个指令在一个时刻只会有一个 Tnew 值

（一个指令只有一个结果）。我们把所有指令的 Tuse 和 Tnew 列出来：

	Tuse	
	rs	rt
lw	1	
sw	1	2
addu	1	1
subu	1	1
ori	1	
lui	1	
beq	0	0
j		
jal		
jr	0	
	{0,1}	{0,1,2}

指令	功能部件	Tnew		
		E	M	W
lw	DM	2	1	0
sw				
addu	ALU	1	0	0
subu	ALU	1	0	0
ori	ALU	1	0	0
lui	ALU	1	0	0
beq				
j				
jal	PC	0	0	0
jr				

然后我们分析转发和暂停的条件，当两条指令发生数据冲突（前面指令的写入寄存器，等于后面指令的读取寄存器），我们就可以根据 Tnew 和 Tuse 值来判断策略。

1. Tnew=0，说明结果已经算出，如果指令处于 WB 级，则可以通过寄存器的内部转发设计解决，不需要任何操作。如果指令不处于 WB 级，则可以通过转发结果来解决。
2. Tnew<=Tuse，说明需要的数据可以及时算出，可以通过转发结果来解决。
3. Tnew>Tuse，说明需要的数据不能及时算出，必须暂停流水线解决。

如下图所示：（F 代表转发，S 代表暂停）

RS									
Tnew	E			M			W		
	ALU	DM	PC+8	ALU	DM	PC+8	ALU	DM	PC+8
Tuse	1	2	0	0	1	0	0	0	0
0	S	S	F	F	S	F	F	F	F
1	F	S	F	F	F	F	F	F	F

RT									
Tnew	E			M			W		
	ALU	DM	PC	ALU	DM	PC	ALU	DM	PC
Tuse	1	2	0	0	1	0	0	0	0
0	S	S	F	F	S	F	F	F	F
1	F	S	F	F	F	F	F	F	F
2	F	F	F	F	F	F	F	F	F

我们将需要转发的情况进行归并与分类，发现共有 5 处地方需要用多选器来选择不同的输入值，分别是 ID 级的 CMP(比较器)的输入 D1 和 D2，EX 级的 ALU 部件的输入 ALU_A 和 ALU_B，MEM 级 DM(数据存储器)的输入 WD(WriteData)。暂停部分与转发相比较容易些，暂停的具体操作内容包括：冻结 PC，不让 PC 的值改变，让 ID/EX 流水级寄存器清零，等于插入了一个 NOP 指令，冻结 IF/ID 流水级寄存器，不让它的值改变。这将引入一个判断是否暂停的信号 PC_en，PC_en 将控制另外的三个信号 IF_en,IF/ID_en,ID/EX_reset，从而实现上面所说的三个操作。

四、测试程序

```
ori $4 $4 4
ori $5 $5 5
ori $6 $6 6
sw $5 -4($4)
sw $4 4($0)
sw $6 8($0)
lw $5 0($0)
addu $5 $5 $5
```

```
lw $5 0($0)
ori $7 $7 1
addu $5 $5 $5

lw $5 0($0)
ori $7 $7 2
ori $8 $8 3
addu $5 $5 $5

lw $5 0($0)
addu $5 $5 $5
addu $5 $5 $5
addu $5 $5 $5

lw $10 4($0)
subu $10 $10 $5

lw $10 4($0)
ori $11 $11 11
subu $10 $10 $10

lw $10 4($0)
ori $11 $11 12
ori $11 $11 13
subu $10 $10 $10

lw $10 4($0)
subu $11 $11 $11
subu $11 $11 $11
subu $11 $11 $1
```

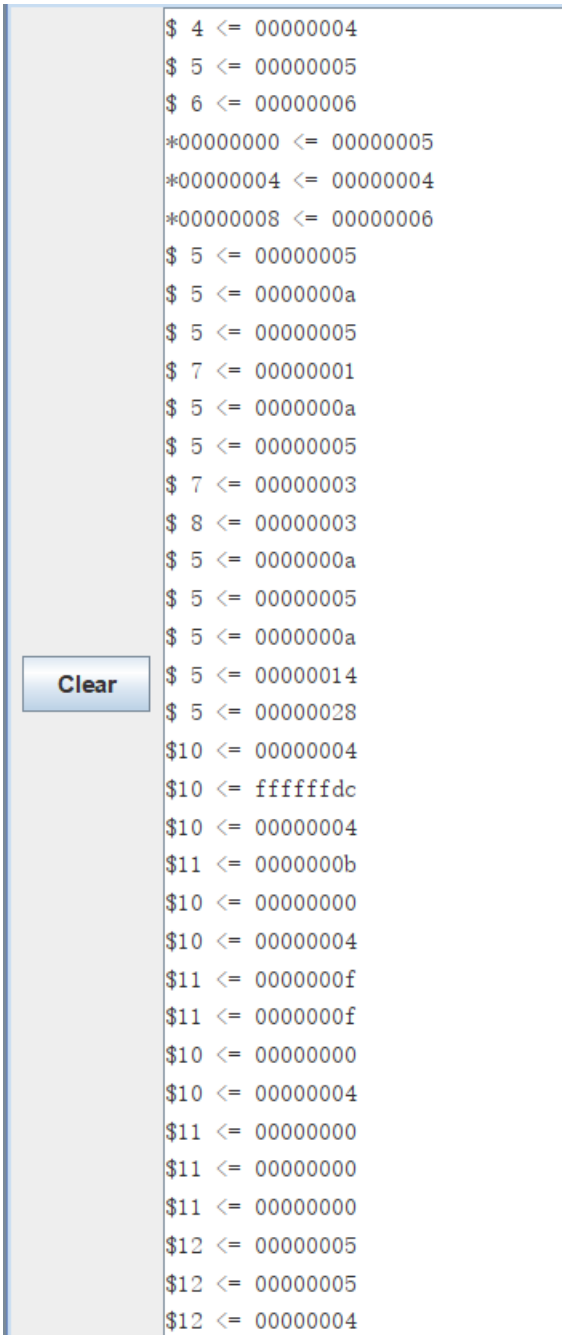
```
lw $12 0($0)
ori $12 $12 0

lw $12 4($0)
ori $13 $13 13
ori $12 $12 0

lw $12 8($0)
ori $13 $13 13
ori $13 $13 14
ori $12 $12 0

lw $12 8($0)
ori $12 $12 0
ori $12 $12 0
ori $12 $12 0

lw $14 0($0)
sw $14 0($0)

lw $14 4($0)
ori $15 $15 5
sw $14 4($0)

lw $14 8($0)
ori $15 $15 6
ori $15 $15 7
sw $14 8($0)
```

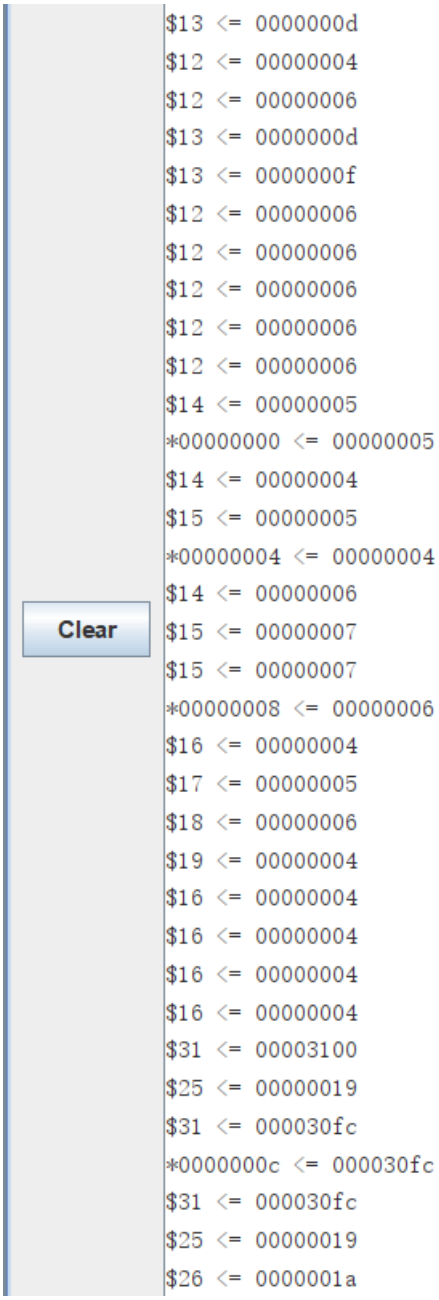
```
lw $16 4($0)
lw $17 -4($16)
lw $18 4($16)
lw $19 0($16)

 lw $16 4($0)
 lw $16 0($16)
 lw $16 0($16)
 lw $16 0($16)

  jal bb
  ori $25 $25 25
  ori $26 $26 26
  ori $27 $27 27
  j l
  nop
  bb:
  nop
  nop
  subu $31 $31 $4
  sw $31 12($0)
  lw $31 12($0)
  jr $ra
    nop
 l:
 jal cc
  ori $25 $25 25
  ori $26 $26 26
  ori $27 $27 27
 j ll
```

```
 cc:
 nop
 nop
 subu $31 $31 $4
 sw $31 12($0)
 lw $31 12($0)
 nop
 jr $ra
   nop
ll:
jal dd
 ori $25 $25 25
 ori $26 $26 26
 ori $27 $27 27
j lll
 dd:
 nop
 nop
 subu $31 $31 $4
 sw $31 12($0)
 lw $31 12($0)
 nop
 nop
 jr $ra
   nop
lll:
lw $20 0($0)
lw $21 4($0)
beq $20 $21 ee
    ori $23 23
```

```
    ori $24 24
    ori $25 25
ee:

lw $20 0($0)
lw $21 4($0)
nop
beq $20 $21 ww
    ori $23 23
    ori $24 24
    ori $25 25
ww:
```

期望运行结果：


$ 4 <= 00000004
$ 5 <= 00000005
$ 6 <= 00000006
*00000000 <= 00000005
*00000004 <= 00000004
*00000008 <= 00000006
$ 5 <= 00000005
$ 5 <= 0000000a
$ 5 <= 00000005
$ 7 <= 00000001
$ 5 <= 0000000a
$ 5 <= 00000005
$ 7 <= 00000003
$ 8 <= 00000003
$ 5 <= 0000000a
$ 5 <= 00000005
$ 5 <= 0000000a
$ 5 <= 00000014
$ 5 <= 00000028
$10 <= 00000004
$10 <= fffffdc
$10 <= 00000004
$11 <= 0000000b
$10 <= 00000000
$10 <= 00000004
$11 <= 0000000f
$11 <= 0000000f
$10 <= 00000000
$10 <= 00000004
$11 <= 00000000
$11 <= 00000000
$11 <= 00000000
$12 <= 00000005
$12 <= 00000005
$12 <= 00000004
Clear


$13 <= 0000000d
$12 <= 00000004
$12 <= 00000006
$13 <= 0000000d
$13 <= 0000000f
$12 <= 00000006
$12 <= 00000006
$12 <= 00000006
$12 <= 00000006
$12 <= 00000006
$14 <= 00000005
*00000000 <= 00000005
$14 <= 00000004
$15 <= 00000005
*00000004 <= 00000004
$14 <= 00000006
$15 <= 00000007
$15 <= 00000007
*00000008 <= 00000006
$16 <= 00000004
$17 <= 00000005
$18 <= 00000006
$19 <= 00000004
$16 <= 00000004
$16 <= 00000004
$16 <= 00000004
$16 <= 00000004
$31 <= 00003100
$25 <= 00000019
$31 <= 000030fc
*0000000c <= 000030fc
$31 <= 000030fc
$25 <= 00000019
$26 <= 0000001a
Clear

Clear

```
$16 <= 00000004
$17 <= 00000005
$18 <= 00000006
$19 <= 00000004
$16 <= 00000004
$16 <= 00000004
$16 <= 00000004
$16 <= 00000004
$31 <= 00003100
$25 <= 00000019
$31 <= 000030fc
*0000000c <= 000030fc
$31 <= 000030fc
$25 <= 00000019
$26 <= 0000001a
$27 <= 0000001b
$31 <= 00003134
$25 <= 00000019
$31 <= 00003130
*0000000c <= 00003130
$31 <= 00003130
$25 <= 00000019
$26 <= 0000001a
$27 <= 0000001b
$31 <= 00003168
$25 <= 00000019
$31 <= 00003164
*0000000c <= 00003164
$31 <= 00003164
$25 <= 00000019
$26 <= 0000001a
$27 <= 0000001b
$20 <= 00000005
$21 <= 00000004
$23 <= 00000017
$24 <= 00000018
$25 <= 00000019
$20 <= 00000005
$21 <= 00000004
$23 <= 00000017
$24 <= 00000018
$25 <= 00000019
```

CPU 运行结果：

```
 5@00003000: $ 4 <= 00000004
 6@00003004: $ 5 <= 00000005
 7@00003008: $ 6 <= 00000006
 7@0000300c: *00000000 <= 00000005
 8@00003010: *00000004 <= 00000004
 9@00003014: *00000008 <= 00000006
11@00003018: $ 5 <= 00000005
13@0000301c: $ 5 <= 0000000a
14@00003020: $ 5 <= 00000005
15@00003024: $ 7 <= 00000001
16@00003028: $ 5 <= 0000000a
17@0000302c: $ 5 <= 00000005
18@00003030: $ 7 <= 00000003
19@00003034: $ 8 <= 00000003
20@00003038: $ 5 <= 0000000a
21@0000303c: $ 5 <= 00000005
23@00003040: $ 5 <= 0000000a
24@00003044: $ 5 <= 00000014
25@00003048: $ 5 <= 00000028
26@0000304c: $10 <= 00000004
28@00003050: $10 <= ffffffdc
29@00003054: $10 <= 00000004
30@00003058: $11 <= 0000000b
31@0000305c: $10 <= 00000000
32@00003060: $10 <= 00000004
33@00003064: $11 <= 0000000f
34@00003068: $11 <= 0000000f
35@0000306c: $10 <= 00000000
36@00003070: $10 <= 00000004
37@00003074: $11 <= 00000000
38@00003078: $11 <= 00000000
39@0000307c: $11 <= 00000000
40@00003080: $12 <= 00000005
42@00003084: $12 <= 00000005
43@00003088: $12 <= 00000004
44@0000308c: $13 <= 0000000d
45@00003090: $12 <= 00000004
46@00003094: $12 <= 00000006
47@00003098: $13 <= 0000000d
48@0000309c: $13 <= 0000000f
49@000030a0: $12 <= 00000006
50@000030a4: $12 <= 00000006
52@000030a8: $12 <= 00000006
53@000030ac: $12 <= 00000006
54@000030b0: $12 <= 00000006
55@000030b4: $14 <= 00000005
55@000030b8: *00000000 <= 00000005
57@000030bc: $14 <= 00000004
58@000030c0: $15 <= 00000005
58@000030c4: *00000004 <= 00000004
60@000030c8: $14 <= 00000006
61@000030cc: $15 <= 00000007
62@000030d0: $15 <= 00000007
62@000030d4: *00000008 <= 00000006
64@000030d8: $16 <= 00000004
66@000030dc: $17 <= 00000005
67@000030e0: $18 <= 00000006
68@000030e4: $19 <= 00000004
69@000030e8: $16 <= 00000004
71@000030ec: $16 <= 00000004
73@000030f0: $16 <= 00000004
75@000030f4: $16 <= 00000004
76@000030f8: $31 <= 00003100
77@000030fc: $25 <= 00000019
80@00003118: $31 <= 000030fc
80@0000311c: *0000000c <= 000030fc
82@00003120: $31 <= 000030fc
87@000030fc: $25 <= 00000019
88@00003100: $26 <= 0000001a
89@00003104: $27 <= 0000001b
92@0000312c: $31 <= 00003134
93@00003130: $25 <= 00000019
96@00003148: $31 <= 00003130
96@0000314c: *0000000c <= 00003130
98@00003150: $31 <= 00003130
103@00003130: $25 <= 00000019
104@00003134: $26 <= 0000001a
105@00003138: $27 <= 0000001b
108@00003160: $31 <= 00003168
109@00003164: $25 <= 00000019
112@0000317c: $31 <= 00003164
112@00003180: *0000000c <= 00003164
114@00003184: $31 <= 00003164
119@00003164: $25 <= 00000019
120@00003168: $26 <= 0000001a
121@0000316c: $27 <= 0000001b
124@00003198: $20 <= 00000005
125@0000319c: $21 <= 00000004
129@000031a4: $23 <= 00000017
130@000031a8: $24 <= 00000018
131@000031ac: $25 <= 00000019
132@000031b0: $20 <= 00000005
133@000031b4: $21 <= 00000004
137@000031c0: $23 <= 00000017
138@000031c4: $24 <= 00000018
139@000031c8: $25 <= 00000019
ISim>
```

上面显示验证为输出直接与 Mars 输出对比，过程中 testbench 中的值也进行过对比。

所以运行结果与 MARS 相同，所以对这 11 条指令没有问题。