**计算机设计与综合实践**

**课程设计报告**

CPU设计

设计题目

学生姓名

学 号

专业班级

指导教师

2018.7.2

完成日期

目录

**一、课程设计任务1**

**二、指令实现原理1**

2.1 Verilog HDL基础1

2.2 MIPS架构简介2

2.2.1 指令基础2

2.2.1 指令格式3

2.3 设计思想3

2.3.1 流水线3

2.3.2 模块化4

**三、指令实现详情及仿真4**

**3.1 逻辑操作类指令**5

**3.2 移位操作类指令**6

**3.3 移动操作类指令**7

**3.4 算数运算类指令**8

**3.5 转移操作类指令**9

**3.6 访存操作类指令**12

**3.7各类指令综合测试**14

**四、课设总结15**

**五、参考文献16**

1. **课程设计任务、要求、目的**

设计CPU系统的总体结构、指令系统和时序信号，实现至少10条并且至少涵盖4种不同类型的指令。利用Modelsim软件进行仿真。

1. **指令实现原理**

**2.1 Verilog HDL基础**

Verilog HDL是在C语言的基础上发展而来的，就语法结构而言，Verilog HDL继承了C语言的很多语法结构，两者有许多相似之处。

Verilog程序的基本设计单元是“模块”（Module），模块结构如图所示，Verilog的模块完全定义在module与endmodule关键字之间，每个模块包括四个主要部分：模块声明、端口定义、数据类型说明和逻辑功能描述。

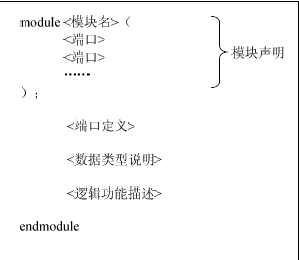


图1.1 Verilog模块组成图

Verilog中的常量（Constant）有三种：整数、实数、字符串。在OpenMIPS的实现过程中只使用到了整数常量，所以，此处也仅介绍整数常量。其格式如图所示。

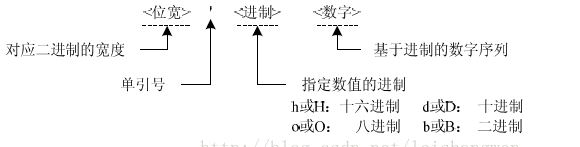


图1.2 Verilog常量结构图

Verilog中变量声明的格式图所示。只有数据类型、变量名是必要的，其他部分都可以省略。如果省略符号和位宽，那么根据数据类型设置为默认值。如果省略元素数，那么默认声明元素数为1。

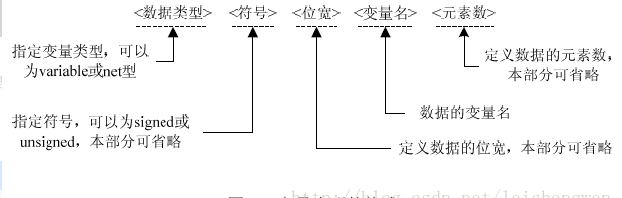


图1.3 Verilog变量结构图

数据类型可以是net型、variable型。net型相当于硬件电路中各种物理连接，其特点是输出的值紧跟输入值的变化而变化。net型相当于硬件电路中各种物理连接，其特点是输出的值紧跟输入值的变化而变化。variable型变量是可以保存上次写入数据的数据类型，一般对应硬件上的一个触发器或锁存器等存储元件，但并不是绝对的，在综合器综合的时候，会根据其被赋值的情况来具体确定是映射成连线还是映射为存储元件。

**2.2 MIPS架构简介**

**2.2.1 指令基础**

指令的主要任务就是对操作数进行运算，操作数有不同的类型和长度，MIPS32提供的基本数据类型有位（b）、字节（Byte）、半字（Half Word）、字（Word）、双字（Double Word）等。

MIPS32的指令中除加载/存储指令外，都是使用寄存器或立即数作为操作数的。MIPS32中的寄存器分为两类：通用寄存器（GPR：General Purpose Register）、特殊寄存器。

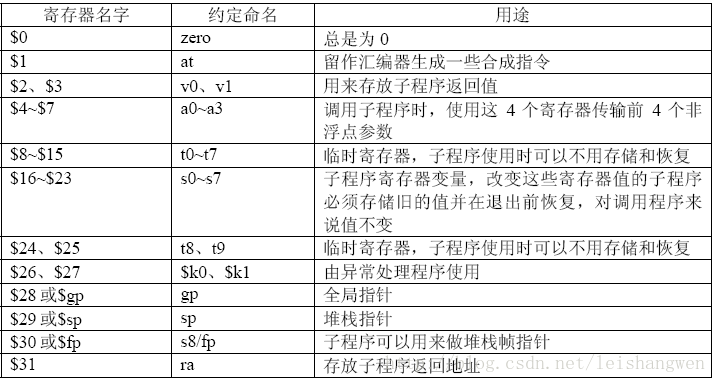


表2.1 MIPS32通用存储器列表

MIPS32架构中定义的特殊寄存器有三个：PC（Program Counter程序计数器）、HI（乘除结果高位寄存器）、LO（乘除结果低位寄存器）。进行乘法运算时，HI和LO保存乘法运算的结果，其中HI存储高32位，LO存储低32位；进行除法运算时，HI和LO保存除法运算的结果，其中HI存储余数，LO存储商。

**2.2.2 指令格式**

MIPS32架构中的所有指令都是32位，也就是32个0、1编码连在一起表示一条指令，有三种指令格式。如图所示。其中op是指令码、func是功能码。

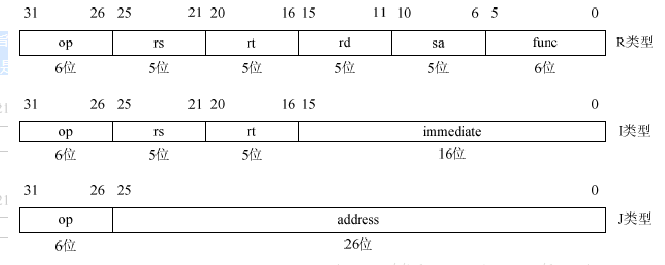


图2.1 MIPS32指令格式

MIPS32架构中定义的指令可以分为以下几类：

1、逻辑操作指令： and、andi、or、ori、xor、xori、nor、lui

2、移位操作指令：sll、sllv、sra、srav、srl、srlv

3、移动操作指令：movn、movz、mfhi、mthi、mflo、mtlo

4、算术操作指令：add、addi、addiu、addu、sub、subu、clo、clz、slt、slti、sltiu、sltu、mul、mult、multu、madd、maddu、msub、msubu、div、divu

5、转移指令：jr 、jalr 、j 、jal、b、bal、beq、bgez、bgezal、bgtz、blez、bltz、bltzal、bne

6、加载存储指令：lb、lbu、lh、lhu、ll、lw、lwl、lwr、sb、sc、sh、sw、swl、swr

还有协处理器访问指令、异常相关指令等。但是我们只要实现其中的一小部分就可以了。

**2.3 设计思想**

2.3.1 流水线

将指令执行过程划分为5个模块，即取指阶段、译码阶段、执行阶段、访存阶段、回写阶段。并将其对应的文件描述如下：

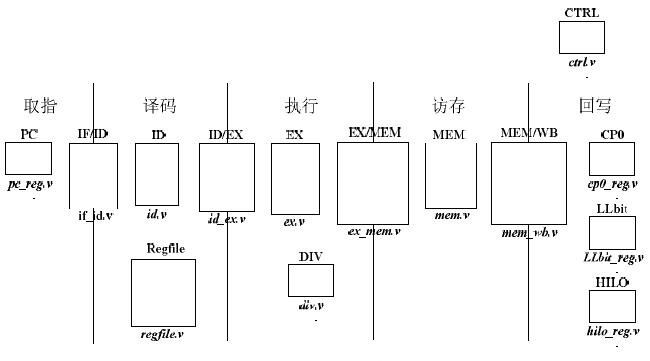


图2.2 流水线对应各个阶段的模块、对应的文件

具体说明如下。

（1）取指阶段

PC模块：给出指令地址，其中实现指令指针寄存器PC，该寄存器的值就是指令地址。对应pc\_reg.v文件

IF/ID模块：实现取指与译码阶段之间的寄存器，将取指阶段的结果（取得的指令、指令地址等信息）在下一个时钟传递到译码阶段。对应if\_id.v文件

（2）译码阶段

ID模块：对指令进行译码，译码结果包括运算类型、运算所需的源操作数、要写入的目的寄存器地址等。对应id.v文件。

Regfile模块：实现了32个32位通用整数寄存器，可以同时进行两个寄存器的读操作和一个寄存器的写操作。对应regfile.v文件。

ID/EX模块：实现译码与执行阶段之间的寄存器，将译码阶段的结果在下一个时钟周期传递到执行阶段。对应id\_ex.v文件。

（3）执行阶段

EX模块：依据译码阶段的结果，进行指定的运算，给出运算结果。对应ex.v文件。

DIV模块：进行除法运算的模块。对应div.v文件。

EX/MEM模块：实现执行与访存阶段之间的寄存器，将执行阶段的结果在下一个时钟周期传递到访存阶段。对应ex\_mem.v文件。

（4）访存阶段

MEM模块：如果是加载、存储指令，那么会对数据存储器进行访问。此外，还会在该模块进行异常判断。对应mem.v文件。

MEM/WB模块：实现访存与回写阶段之间的寄存器，将访存阶段的结果在下一个时钟周期传递到回写阶段。对应mem\_wb.v文件。

（5）回写阶段

CP0模块：对应MIPS架构中的协处理器CP0。

LLbit模块：实现寄存器LLbit，在链接加载指令ll、条件存储指令sc的处理过程中会使用到该寄存器。

HILO模块：实现寄存器HI、LO，在乘法、除法指令的处理过程中会使用到这两个寄存器。

2.3.2 模块化

另外，模块化的接口以及连接关系如下所示：

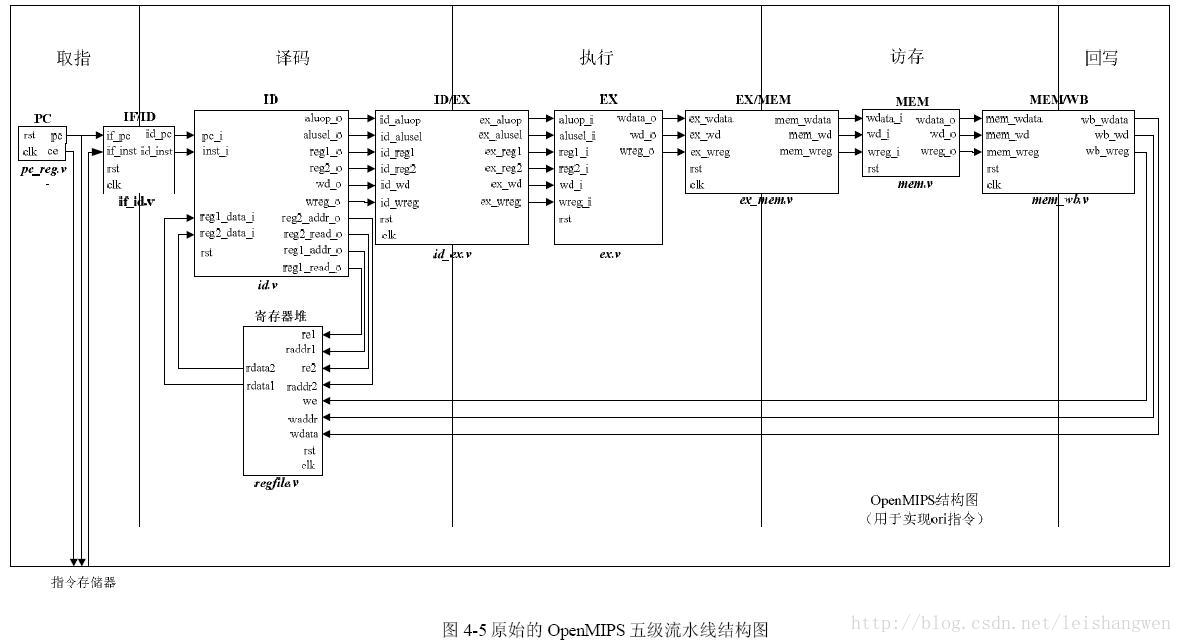


图2.3 五级流水线结构及模块对应关系图

1. **指令实现详情及仿真**

鉴于程序实现代码太多，就不贴出来了。

**3.1 逻辑操作类指令：ORI、ANDI、AND、XORI、XOR、NOR、OR**

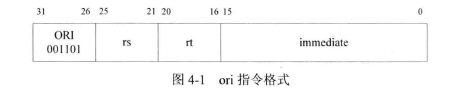


图3.1

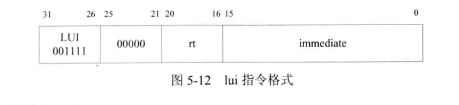


图3.2

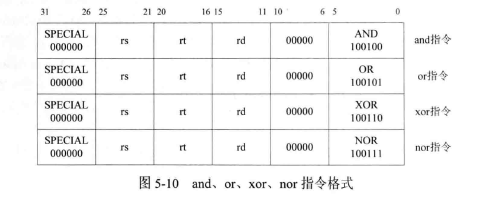


图3.3

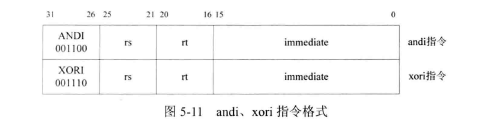


图3.4

测试程序：

.org 0x0

.global \_start

.set noat

\_start:

lui $1,0x0101

ori $1,$1,0x0101

ori $2,$1,0x1100 # $2 = $1 | 0x1100 = 0x01011101

or $1,$1,$2 # $1 = $1 | $2 = 0x01011101

andi $3,$1,0x00fe # $3 = $1 & 0x00fe = 0x00000000

#（$1 = 0x0101 1101 andi 0x00fe0000 –>$3）

and $1,$3,$1 # $1 = $3 & $1 = 0x00000000

xori $4,$1,0xff00 # $4 = $1 ^ 0xff00 = 0x0000ff00

xor $1,$4,$1 # $1 = $4 ^ $1 = 0x0000ff00

nor $1,$4,$1 # $1 = $4 ~^ $1 = 0xffff00ff nor is "not or"

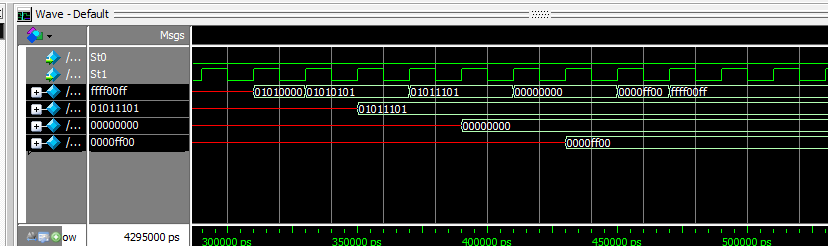


图3.5

**3.2 移位操作类指令：SLL、SLLV、SRL、SRLV、SRA、SRAV**

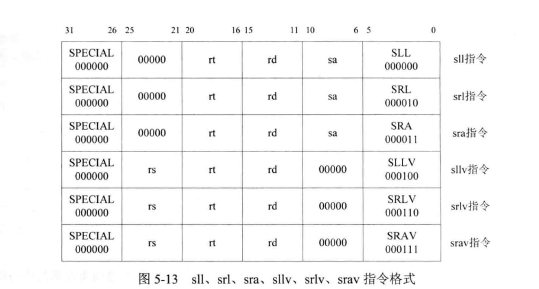


图3.6

测试程序：

.org 0x0

.set noat

.global \_start

\_start:

lui $2,0x0404

ori $2,$2,0x0404

ori $7,$0,0x7

ori $5,$0,0x5

ori $8,$0,0x8

sync

sll $2,$2,8 # $2 = 0x40404040 sll 8 = 0x04040400//算术左移

sllv $2,$2,$7 # $2 = 0x04040400 sll 7 = 0x02020000//逻辑左移，移位位数是通过寄存器的值

srl $2,$2,8 # $2 = 0x02020000 srl 8 = 0x00020200//逻辑右移

srlv $2,$2,$5 # $2 = 0x00020200 srl 5 = 0x00001010//逻辑右移，移位位数是通过寄存器的值

nop

sll $2,$2,19 # $2 = 0x00001010 sll 19 = 0x80800000

ssnop

sra $2,$2,16 # $2 = 0x80800000 sra 16 = 0xffff8080//算术右移

srav $2,$2,$8 # $2 = 0xffff8080 sra 8 = 0xffffff80 //算术右移，移位位数是通过寄存器的值

对应波形如下：

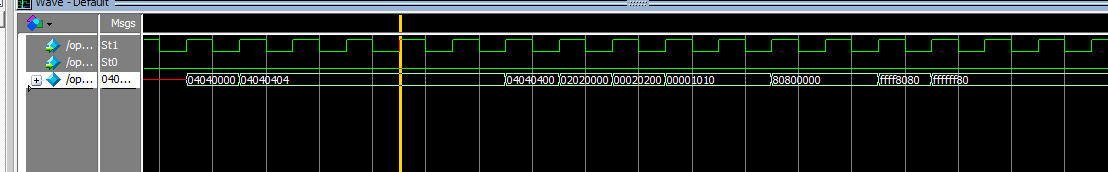


图3.7

**3.3 移动操作类指令：MOVZ、MOVN**

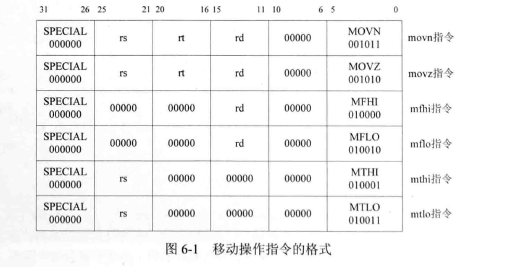


图3.8

测试程序：

.org 0x0

.set noat

.global \_start

\_start:

#//给寄存器$1、$2、$3、/4赋初值

lui $1,0x0000 # $1 = 0x00000000

lui $2,0xffff # $2 = 0xffff0000

lui $3,0x0505 # $3 = 0x05050000

lui $4,0x0000 # $4 = 0x00000000

#//movz指令，寄存器$1为0，所以$2的值赋给$4

movz $4,$2,$1 # $4 = 0xffff0000

#//movn指令，寄存器$1为0，所以$2不值赋，$4不变

movn $4,$3,$1 # $4 = 0xffff0000

#//movn指令，寄存器$1不为0，所以$2的值赋给$4

movn $4,$3,$2 # $4 = 0x05050000

#//movz指令，寄存器$3不为0，所以不值赋，$4保持不变

movz $4,$2,$3 # $4 = 0x05050000

对应波形如下：

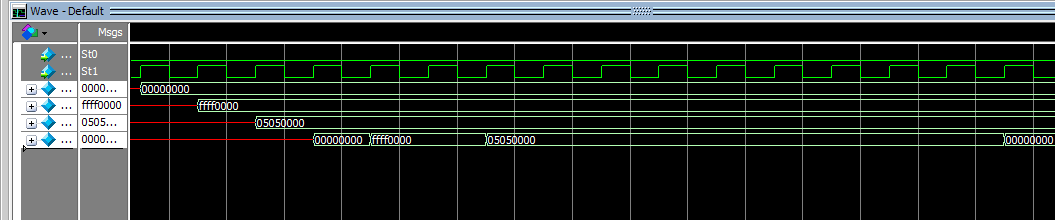


图3.9

**3.4 算数运算类指令：ADD、ADDI、ADDIU、ADDU、SUBU、SUB、SUBI**

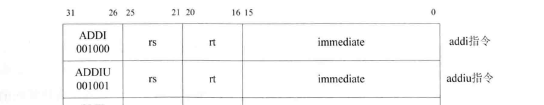


图3.10

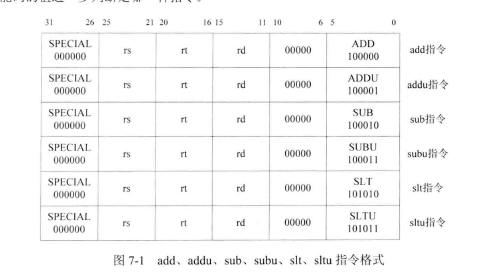


图3.11

测试程序：

.org 0x0

.set noat

.global \_start

\_start:

######### add\addi\addiu\addu\sub\subu ##########

ori $1,$0,0x8000 # $1 = 0x8000

sll $1,$1,16 # $1 = 0x80000000

ori $1,$1,0x0010 # $1 = 0x80000010//$1赋值

ori $2,$0,0x8000 # $2 = 0x8000

sll $2,$2,16 # $2 = 0x80000000

ori $2,$2,0x0001 # $2 = 0x80000001//$2赋值

ori $3,$0,0x0000 # $3 = 0x00000000

addu $3,$2,$1 # $3 = 0x00000011//$1加$2，无符号加

ori $3,$0,0x0000 # $3 = 0x00000000

#//$1加$2，有符号加，溢出

add $3,$2,$1 # overflow,$3 keep 0x00000000

sub $3,$1,$3 # $3 = 0x80000010 //$1减去$3，有符号减

subu $3,$3,$2 # $3 = 0xF //$3减去$2，(0x80000010 -0x80000001 )无符号减

addi $3,$3,2 # $3 = 0x11 //(0xF +0x2)$3加2，有符号加

ori $3,$0,0x0000 # $3 = 0x00000000

addiu $3,$3,0x8000 # $3 = 0xffff8000 // $3 (0x0)加0xffff8000，无符号加

对应波形如下：

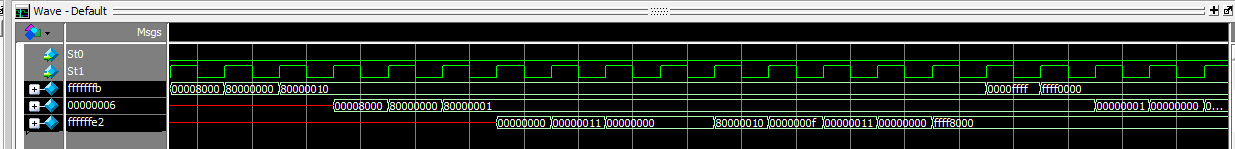


图3.12

**3.5 转移操作类指令：J、JAL、JR、BEQ**

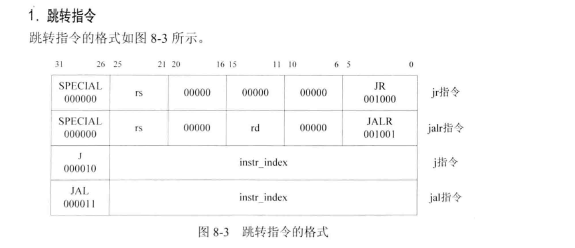


图3.13

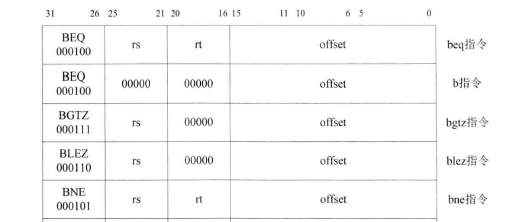


图3.14

测试程序：

.org 0x0

.set noat

.set noreorder

.set nomacro

.global \_start

\_start:

ori $1,$0,0x0001 # $1 = 0x1

j 0x20 ##//直接跳转

ori $1,$0,0x0002 # $1 = 0x2，这是延迟槽指令 //乱序执行

ori $1,$0,0x1111

ori $1,$0,0x1100

.org 0x20

ori $1,$0,0x0003 # $1 = 0x3

jal 0x40# 转移到0x40处，同时设置$31位0x2c

#//jal指令的延迟槽指令是除法指令，$1需要多个时钟周期保持为0x3

div $zero,$31,$1 # $31 = 0x2c, $1 = 0x3

# HI = 0x2, LO = 0xe //除法计算结果，延迟槽指令

ori $1,$0,0x0005 # r1 = 0x5

ori $1,$0,0x0006 # r1 = 0x6

j 0x60 #//跳到0x60

nop

.org 0x40

jalr $2,$31 # $31为0x2c，转移到0x2c，同时设置$2为0x48

or $1,$2,$0 # $1 = 0x48,延迟槽指令

#//乱序执行，除法之后$1保留站数据到位，该指令就执行了

ori $1,$0,0x0009 # $1 = 0x9

ori $1,$0,0x000a # $1 = 0xa

j 0x80

nop

.org 0x60

ori $1,$0,0x0007 # $1 = 0x7

jr $2 #此时$2为0x48，所以转移到0x48处

ori $1,$0,0x0008 # $1 = 0x8，延迟槽指令

ori $1,$0,0x1111

ori $1,$0,0x1100

.org 0x80

nop

\_loop:

j \_loop

nop ori $1,$0,0x0005 # r1 = 0x5

ori $1,$0,0x0006 # r1 = 0x6

j 0x60#//跳到0x60

nop

.org 0x40

jalr $2,$31

or $1,$2,$0 # $1 = 0x48//乱序执行，除法之后$1保留站数据到位，该指令就执行了

ori $1,$0,0x0009 # $1 = 0x9

ori $1,$0,0x000a # $1 = 0xa

j 0x80

nop

.org 0x60

ori $1,$0,0x0007 # $1 = 0x7

jr $2

ori $1,$0,0x0008 # $1 = 0x8

ori $1,$0,0x1111

ori $1,$0,0x1100

.org 0x80

nop

\_loop:

j \_loop

nop

对应波形如下：

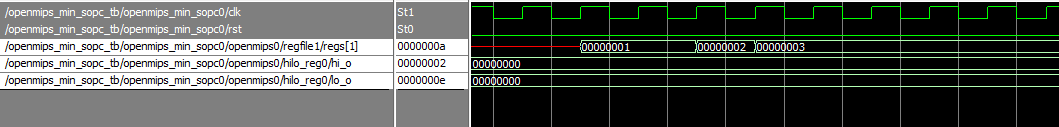


图3.15

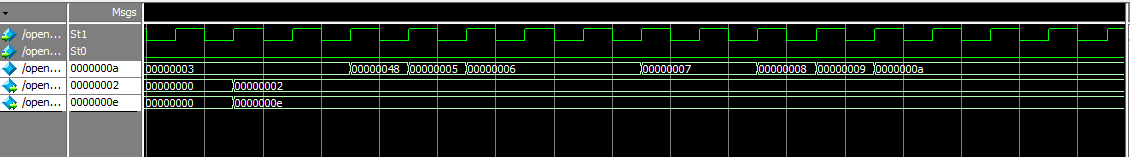


图3.16

**3.6 访存操作类指令：SW、LW**



图3.17



图3.18

测试程序：

.org 0x0

.set noat

.set noreorder

.set nomacro

.global \_start

\_start:

ori $1,$0,0x1234 # $1 = 0x00001234

sw $1,0x0($0) # [0x0] = 0x00001234

#向数据存储器的地址0x0处存储0x00001234

ori $2,$0,0x1234 # $2 = 0x00001234

ori $1,$0,0x0 # $1 = 0x0

lw $1,0x0($0) # 从数据存储器的地址0x0将数据处加到寄存器$1，

# $1 = 0x00001234

beq $1,$2,Label #比较寄存器$1与$2，若相等，则则转移到Label处

nop

ori $1,$0,0x4567

nop

Label:

ori $1,$0,0x89ab # $1 = 0x000089ab

nop

\_loop:

j \_loop

nop

对应波形如下：

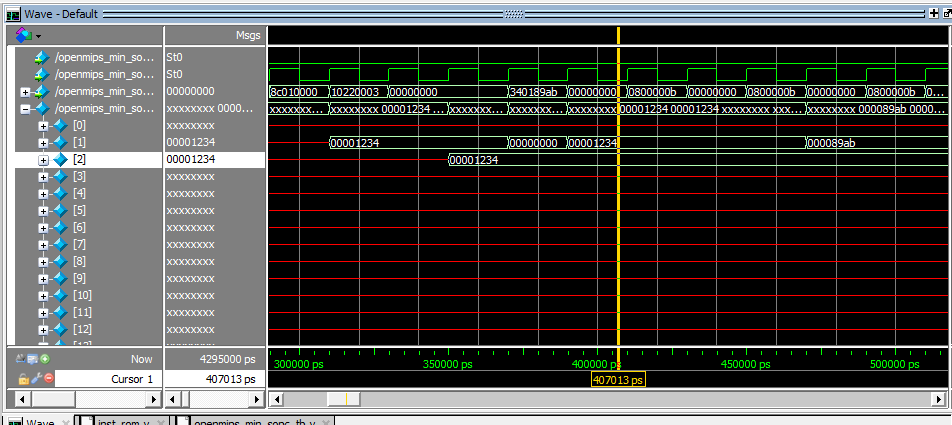


图3.19

34011234 ac010000 34021234 34010000 8c010000 10220003 00000000 34014567 00000000 340189ab 00000000 0800000b 00000000

**3.7 各类指令综合测试：**

//addi $1, $0, 2 $1=0000 0002

001000 00000 00001 0000000000000010

//addi $2, $1, 4 $2=0000 0006

001000 00001 00010 0000000000000100

//add $3, $1, $2 $3=0000 0008

000000 00001 00010 00011 00000 100000

//sub $4, $3, $2 $4=0000 0002

000000 00011 00010 00100 00000 100010

//ori $5, $2, 1 $5=0000 0007

001101 00010 00101 0000000000000001

//or $6, $2, $1 $6=0000 0006

000000 00010 00001 00110 00000 100101

//and $7, $5, $6 $7=0000 0006

000000 00101 00110 00111 00000 100100

//sw $7, 0($7) mem[$7]=00000006

101011 00111 00111 0000000000000000

//lw $8, 0($2) $8=0000 0006

100011 00010 01000 0000000000000000

//beq $6, $7, -3 PC-=8;

20010002//addi $1, $0, 2 $1=0000 0002

20220004//addi $2, $1, 4 $2=0000 0006

00221820//add $3, $1, $2 $3=0000 0008

00622022//sub $4, $3, $2 $4=0000 0002

34450001//ori $5, $2, 1 $5=0000 0007

00413025//or $6, $2, $1 $6=0000 0006

00a63824//and $7, $5, $6 $7=0000 0006

ace70000//sw $7, 0($7) $7=00000006

8c480000//lw $8, 0($2) $8=0000 0006

10c7fffd//beq $6, $7, -3 PC-=8;

各个寄存器的值：

对应波形如下：

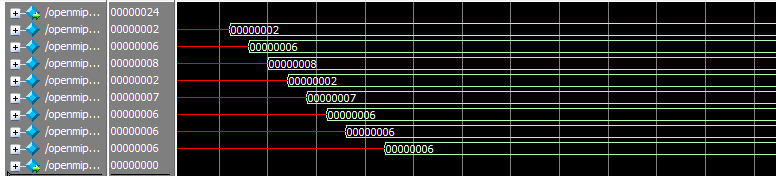


图3.20

PC值减8：0x28到0x1c



图3.21

1. **课设总结**

本次计算机设计与综合实践课程设计令我获益匪浅。经过这次CPU的设计，是我对于计算机指令的执行有了更深的了解，同时体会到计算机各个功能结构协同运行实现相关指令的美妙。

学习Verilog语言时候我才意识到语言的学习都是想通的，有了C语言的基础后，也就等于掌握了一类高级语言的语法。

1. **参考文献**

[1] 雷思磊《自己动手写CPU》及其博客

<https://blog.csdn.net/leishangwen/article/category/5723475/1>

[2] MIPS单周期CPU设计（Verilog）：

<https://blog.csdn.net/quinze_lee/article/details/51174019>

[3] 张晨曦 王志英《计算机体系结构》

[4] 唐朔飞《计算机组成原理》