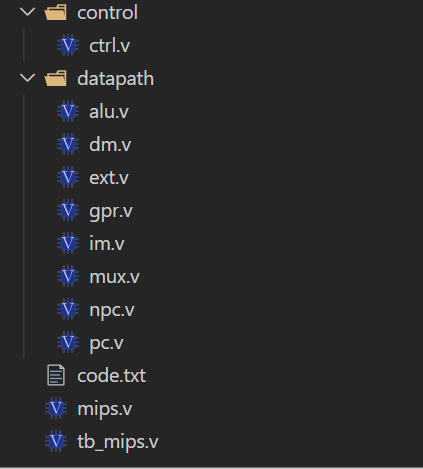
**实验报告**

姓名：孙延浩 学号：21181225

1. 整体概述

处理器支持以下指令{addu,subu,ori,lw,sw,beq,lui,addi,addiu,slt,j,jal,jr},处理器为单周期设计。

目录结构如图所示：



其中mips.v为顶层结构，tb\_mips.v为测试文件,code.txt为IM中要读入的内容。

2.模块定义

2.1 alu

（1）基本描述

alu是该单周期处理器的运算逻辑单元，主要功能是完成需要处理的运算，并将运算结果传输给需要的部件。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| a[31:0] | I | 32位数据1 |
| b[31:0] | I | 32位数据2 |
| aluop[1:0] | I | 00：加  01：减  10：或  11: 判断是否小于 |
| aluout | O | Alu输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 对两个操作数进行加 | aluout<=a+b |
| 2 | 对两个操作数进行减 | aluout<=a-b |
| 3 | 对两个操作数进行或 | aluout<=a|b |
| 4 | 判断a是否小于b | 如果a小于b置1，否则置0 |

2.2 pc

（1）基本描述

PC主要功能是完成输出当前指令地址并保存下一条指令地址。复位后，PC指向0x00003000

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| rst | I | 复位信号 |
| nPC[31:2] | I | nPC作为输入 |
| pcout[31:2] | O | 当前pc输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | rst==1，pc=0x00003000 |
| 2 | 保存nPC并输出 | 在每个clk上升沿保存nPC并输出 |

2.3 Ctrl

（1）基本描述

Ctrl部件用于分析当前指令并产生相应的控制信号

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| instr[31:0] | I | 当前指令 |
| extop[1:0] | O | 对Extender的拓展方式进行规定 |
| aluop[1:0] | O | 控制ALU执行的运算 |
| memwrite | O | 数据存储器写使能 |
| regwrite | O | 寄存器写使能 |
| memtoreg[1:0] | O | 控制寄存器写是来自ALU结果还是DM还是PC |
| alusrc | O | 控制是立即数参与运算还是寄存器rt |
| regdst[1:0] | O | 用于控制写入哪个寄存器（rd/rt/ra） |
| npcop | O | 用于控制PC的下一个值 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 产生信号 | 分析当前指令并产生相应的控制信号 |

2.4 ext

（1）基本描述

ext部件用于对位数不足的数据进行位数拓展

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| offset[15:0] | I | 16位数据输入 |
| extop[1:0] | I | 00：高位不带符号拓展  01：高位带符号拓展  10：低位拓展0 |
| eout[31:0] | O | 32位数据输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 高位不带符号拓展 | 在16位数据前16位填0 |
| 2 | 高位带符号拓展 | 在16位数据前16位填它的符号位 |
| 3 | 低位拓展0 | 在16位数据后16位填充0 |

2.5 gpr

（1）基本描述

gpr放有32个寄存器，用于处理外界读写寄存器的请求，并给出对应的输出结果

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| a1[4:0] | I | 读寄存器地址1 |
| a2[4:0] | I | 读寄存器地址2 |
| a3[4:0] | I | 写寄存器地址 |
| rst | I | 复位 |
| wd[31:0] | I | 写入数据的输入 |
| clk | I | 时钟信号 |
| regWrite | I | 读写控制信号 |
| rd1[31:0] | O | 32位输出1 |
| rd2[31:0] | O | 32位输出2 |
| Ra[31:0] | O | 31号寄存器ra的值 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 写寄存器 | 在有写寄存器信号时往目标寄存器写入 |
| 2 | 读寄存器 | 输出目标寄存器中储存的值 |

2.6 dm

（1）基本描述

数据存储器，能读出或写入相关数据。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| din[31:0] | I | 要写入的数据 |
| addr[11:2] | I | 写入DM的地址 |
| we | I | 写使能 |
| clk | I | 时钟信号 |
| dout[31:0] | O | 32位数据输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 存储数据 | 将相关数据在写使能时写入 |
| 2 | 读取数据 | 从存储器中读出数据 |

2.7 im

（1）基本描述

指令存储器，从其中读取预存好的指令。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| addr[11:2] | I | 要读取的指令地址 |
| dout[31:0] | O | 32位指令输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读取指令 | 从指令寄存器读取指令 |

2.8 mux

2.8.1 aluMux

（1）基本描述

作为alu的多路选择器，选择运算的类型并输出相应的结果。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| a[31:0] | I | 32位输入1 |
| b[31:0] | I | 32位输入2 |
| aluop[1:0] | I | 选择器信号：  00:mout<=a+b  01:mout<=a-b  10:mout<=a|b  11:mout的值由a、b大小决定 |
| mout[31:0] | O | 32位多路选择器输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 选择运算类型 | 根据aluop选择mout输出的结果 |

2.8.2 npcMux

（1）基本描述

作为npc的多路选择器，选择npc的值并输出相应结果

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| pc[31:2] | I | 当前PC值 |
| npcop[1:0] | I | 选择器信号：  00:npc<=pc+1  01:npc<=(beq跳转成功)?跳转位置:pc+1  10:npc<={pc[31:28],mem[25:0]}  11:npc<=ra[31:2] //jr时npc为寄存器31内容 |
| ra[31:0] | I | 31号寄存器存的内容 |
| mem[31:0] | I | dm输出的32位数据 |
| aluout[31:0] | I | alu的输出结果 |
| npc[31:2] | O | 计算出的npc作为输出，传给其他模块 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 得出npc的值 | 作为npc的多路选择器，选择npc的值并输出相应结果 |

2.8.3 extNux

（1）基本描述

用于拓展立即数的多路选择器，选择立即数的拓展方式并输出拓展结果

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| offset[15:0] | I | 输入16位立即数 |
| extop[1:0] | I | 选择器信号：  00：高位不带符号拓展  01：高位带符号拓展  10：低位拓展0 |
| eout | O | 32位拓展结果输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 拓展16位立即数 | 实现不同类型的立即数拓展 |

2.8.4 aluScrMux

（1）基本描述

用于选择运算数的多路选择器，根据信号选择运算数的来源并输出

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| ain[31:0] | I | 来自立即数拓展的32位输入 |
| rt[31:0] | I | 来自rt的32位输入 |
| alusrc | I | 选择器信号：  0:rt作为多路选择器结果  1:ain作为多路选择器结果 |
| aout[31:0] | O | 多路选择器输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 选择运算数 | 根据信号选择alu的一个运算数 |

2.8.5 regDstMux

（1）基本描述

用于选择写回的目标寄存器，将目标寄存器的寄存器号输出

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| rt[4:0] | I | rt的寄存器号 |
| rd[4:0] | I | rd的寄存器号 |
| regdst[1:0] | I | 选择器信号：  00:rout<=rt  01:rout<=rd  10:rout<=31 |
| rout[4:0] | O | 输出的寄存器号 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 选择写回的目标寄存器 | 将选择好的目标寄存器的寄存器号输出 |

2.8.6 MemtoRegMux

（1）基本描述

用于选择写入寄存器的值的多路选择器，选出相应的值并输出

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| aluout[31:0] | I | alu的32位输出作为输入 |
| dmout[31:0] | I | dm的32位输出作为输入 |
| pcout[31:2] | I | pc的30位输出作为输入 |
| memtoreg[1:0] | I | 选择信号:  00:mout<=aluout  01:mout<=dmout  10:mout<=aluout  11:mout<={pcout+1,2’b00} |
| mout[31:0] | O | 写入寄存器的值 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 选择写入寄存器的值 | 选择写入寄存器的值的多路选择器，选出相应的值并输出 |

2.9 npc

（1）基本描述

生成下一个pc的值并传给pc，使得pc在每一个时钟周期后指向应在的位置

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| pc[31:2] | I | 当前30位pc输入 |
| npcop[1:0] | I | 选择信号，传入多路选择器 |
| ra[31:0] | I | 31号寄存器中的内容 |
| mem[31:0] | I | 32位内存单元输入 |
| aluout[31:0] | I | 32位alu输出作为输入 |
| nout[31:2] | I | 下一pc值作为输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 判断pc转移状况 | 通过信号判断是否转移 |
| 2 | 得到下一个pc值 | 连接多路选择器后得到下一pc值并输出 |

2.10 mips

（1）基本描述

将所有的部件通过线网连接起来，作为顶层模块完成电路构建，在rst为1时初始化线路，在时钟脉冲来临时运转整个模块。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 作为时钟脉冲控制电路的运转 |
| rst | I | 作为置位信号 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 连接所有器件 | 通过线网让电路运作起来 |
| 2 | 初始化 | 在rst为1时初始化某些值 |
| 3 | 给予时钟脉冲 | 接收到外界的时钟脉冲后传递给线路中的器件 |

2.11 tb\_mips(testbench)

（1）基本描述

测试模块，用于合理地给出clk和rst信号

（2）模块接口

无接口

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 初始化 | 最初时给出rst上升沿，对具体部件进行置位 |
| 2 | 给出脉冲 | 每隔固定时间给出时钟脉冲 |

3.测试程序原理及结果

3.1测试源代码

测试程序test.asm:

.data

.text

main:

lui $t0 0 #赋值0

lui $t1 0 #赋值0

ori $t1,$t1,10 #给$t1寄存器赋值10

lui $t4,0 #给$t4赋值0

lui $t5,0 #赋值0

ori $t5,$t5,4 #给$t5赋值4，作为DM地址递增量

lui $t6,0 #赋值0，作为累加结果

labelFor: #for循环的跳转点

jal sum #跳转到累加函数

beq $t1,$t0,labelJump #如果$t1内容减到0了就跳出for循环

j labelFor #用来建立循环，在上一步没跳出就继续

labelJump: #用来跳出循环的标记点

subu $t4,$t4,$t5 #将$t4指向最后写入的内容

lw $t1,0($t4) #加载最后写入的内容到$t1

lui $t6,0 #赋值0

addu $t4,$t4,$t5 #将指向DM的指针移到下一块空内存单元

ori $t0,$t0,55 #给$t0赋值55

beq $t1,$t0,labelTest #如果运算结果等于55就跳转

addi $t6,$t6,1 #加1用来判断是否成功跳转，从而判断结果是否正确

labelTest: #跳转点

addi $t6,$t6,1 #$t6值+1

sw $t6,0($t4) #如果最后写入的值是1证明累加1到10运算结果正确，否则不正确

addu $t4,$t4,$t5 #将指向DM的指针移到下一块空内存单元

lui $t6,0

ori $t6,255

lui $t2,0

ori $t2,200

slt $t3,$t6,$t2 # ($t3)=(($t6)<($t2))? 1 : 0;

sw $t3,0($t4) #若为0 则指令正确执行

addu $t4,$t4,$t5 #将指向DM的指针移到下一块空内存单元

lui $t6,0

lui $t2,0

addi $t6,$t6,-10

addi $t2,$t2,-2

slt $t3,$t6,$t2 # ($t3)=(($t6)<($t2))? 1 : 0;

sw $t3,0($t4) #若为1 则指令正确执行

addu $t4,$t4,$t5 #将指向DM的指针移到下一块空内存单元

LabelWhile: #开始死循环

addi $t6,$t6,-1

j LabelWhile

syscall

sum:

addu $t6,$t6,$t1 #第i次循环把10-i+1累加

sw $t6,0($t4) #将每次的临时结果存入到DM

addu $t4,$t4,$t5 #将指向DM的指针移到下一块空内存单元

addiu $t1,$t1,-1 #$t1减一

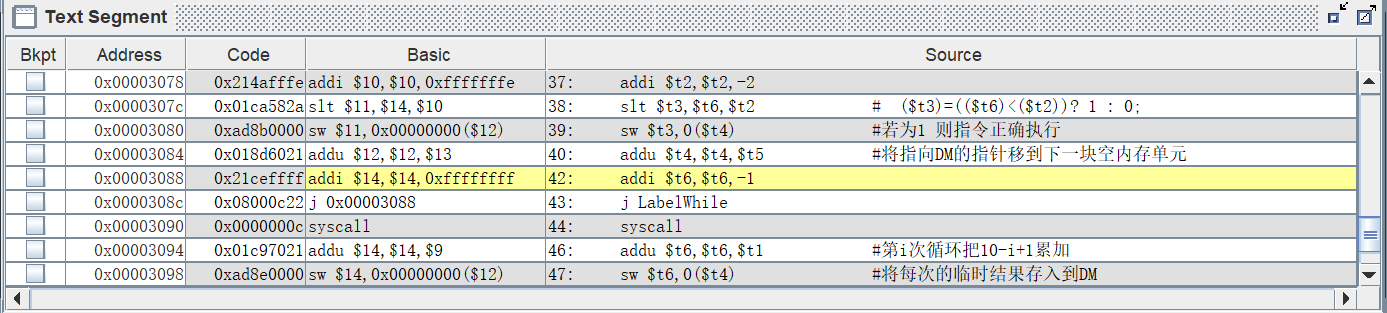
jr $ra #跳转出函数

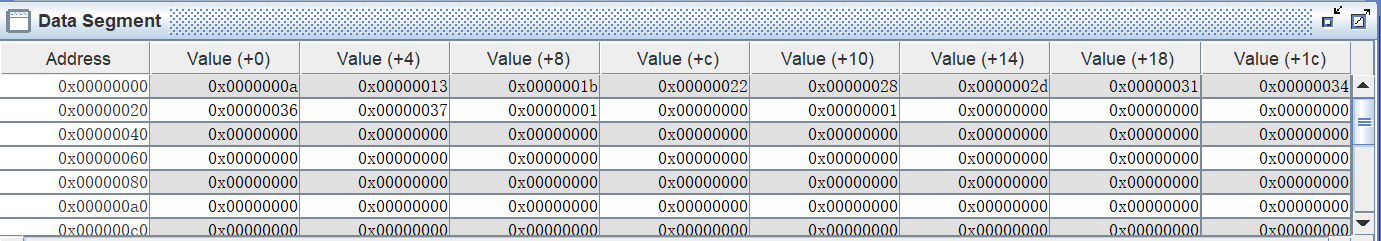
3.2测试原理

程序中所有MIPS-Lite3中的指令都参与测试，其中为了验证jal和jr的正确性，代码中包含累加函数sum，并对其进行了循环调用共10次，每次调用完会将结果保存在dm中，计算的结果若和从1到10累加的结果55一致将会在0x00000028存入1，否则存入0，若结果一致，则证明这部分程序涉及的一系列指令无误，并在后面分别测试slt的结果是否正确，若0x0000002c为0，0x00000030是1，则指令无误，最终用一个死循环保证程序持续运行。

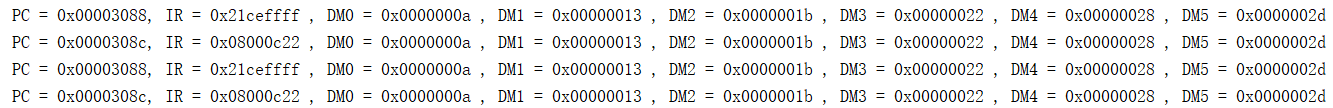
3.3测试结果（若看不清放大word比例即可）

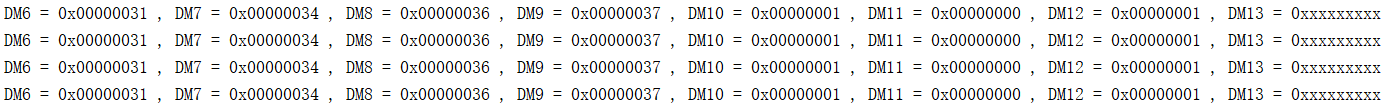
3.3.1 test.asm在mars中的运行结果：





3.3.2 verilog程序模拟后的结果：





由上图所看，dm中数据完全一致，并且与测试方案中理想结果一致，指令执行正确。

4.问答

addi是支持溢出检测的加立即数指令，遇到溢出时溢出错误标志置1，控制器将导致寄存器写信号无效，导致这次产生了溢出的结果不写入目标寄存器，而addiu是不受溢出限制的立即数加指令，遇到溢出时取模再将结果写入目标寄存器。当不考虑溢出时两者都是计算取模后的结果，并将结果写入寄存器，两者等价。

add是符号加，若不产生溢出，直接将结果存入rd寄存器，如果产生，抛出溢出异常，此次不进行rd的写入，addu是无符号加，不检查溢出，产生溢出也不会抛出异常。当不考虑溢出的时候，对于cpu来说不管有没有符号位，都是将寄存器的数值各位相加并进位得出结果后存入rd中。