计算机组成原理实验报告

Project3

班级：1616001

学号：161630131

姓名：石一泽

学院：计算机科学与技术

2017.06.12

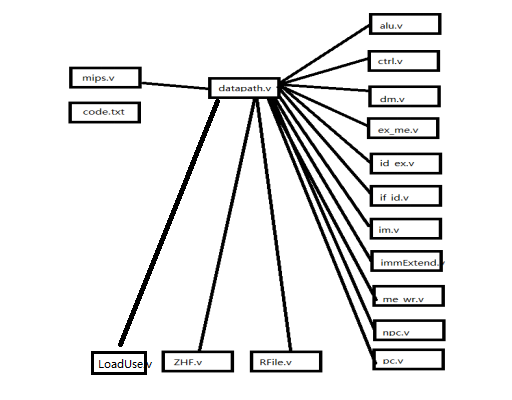
**文档目录：**

1. 功能设计说明
2. 模块化和层次化设计说明
3. 具体模块定义
4. 测试代码及结果
5. 实验完成时间安排
6. 心得体会
7. **功能设计说明**

1.完成以下指令集。

* 1. 书上的9条指令（add，sub，and，or，slt，lw，sw，beq和J指令）以及为方便调试所增加的addi指令。
  2. 不支持溢出。

1. 处理器为流水线设计，支持了转发，load-use冒险和branch前移。
2. 功能模块和数据通路采用课本p198,p208页设计。
3. **模块化和层次化设计说明**



1. **具体模块定义**

Datapath

1）PC模块定义：

**(1) 基本描述**

PC 主要功能是完成输出当前指令地址。复位后，PC指向0x0000\_3000，此处为第一条指令的地址。

**(2) 模块接口**

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| **[31:0]npc** | **I** | **下一条指令** |
| **clk** | **I** | **时钟信号** |
| **rst** | **I** | **复位信号** |
| **[31:0]pc** | **O** | **当前指令** |
| **pcwrite** | **i** | **pc写指令** |

**（3）功能定义**

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **功能描述** |
| **1** | **复位** | **rst=1时，将pc置为0X0000\_3000** |
| **2** | **输出指令地址** | **时钟信号到来时，写信号，将npc赋给pc** |

2）NPC模块定义：

**(1) 基本描述**

NPC 主要功能是根据当前指令是否为beq指令以及j指令，输出下一条指令的地址。该模块需要多路选择器

**(2) 模块接口**

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| **[31:0]pc\_four** | **I** | **Pc+4** |
| **zero** | **I** | **ALU计算结果:**  **1表示当前两寄存器(rs,rt)值相等;**  **0表示不相等。** |
| **branch** | **I** | **是否为beq指令。1是0否** |
| **jump** | **I** | **是否为jump指令。1是0否** |
| **[31:0]jAddr** | **I** | **J指令跳转地址** |
| **[31:0]bAddr** | **I** | **beq指令跳转地址** |
| **[31:0]npc** | **O** | **输出下一条指令地址** |

**（3）功能定义**

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **功能描述** |
| **1** | **输出指令地址** | **根据zero与br和jump的值输出下一条指令的地址。** |

3）ALU模块定义：

**(1) 基本描述**

实现addu，subu，or, sltu四种计算。

**(2) 模块接口**

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| **[2:0]ALUctr** | **I** | **ALU控制信号** |
| **[31:0]A** | **I** | **输入数据A** |
| **[31:0]B** | **I** | **输入数据B** |
| **zero** | **O** | **若A!=B,zero=0,否则，zero=1** |
| **[31:0]result** | **O** | **Alu计算输出的结果。** |

**（3）功能定义**

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **功能描述** |
| **1** | **输出计算结果** | **根据alu控制信号，输出A与B的计算结果** |
| **2** | **输出zero** | **若result=0则输出zero=1，否则输出zero=0。** |

4）immExtend模块定义：

**(1) 基本描述**

将输入的16位立即数按符号位扩展或零扩展为32位。

**(2) 模块接口**

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| **[15:0]din** | **I** | **输入的16位数据** |
| **Extop** | **I** | **扩展方式信号** |
| **[31:0]dout** | **O** | **输出的32位数据** |

**（3）功能定义**

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **功能描述** |
| **1** | **立即数扩展** | **将addrin扩展为32位的addrout.** |

5）RFile模块定义：

**(1) 基本描述**

**根据输入的两个寄存器地址，输出相应寄存器的值，根据寄存器写信号和寄存器地址，将输入的数据选择写入寄存器。**

**(2) 模块接口**

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| **clk** | **I** | **时钟信号** |
| **[4:0]rs** | **I** | **Rs寄存器地址** |
| **[4:0]rt** | **I** | **Rt寄存器地址** |
| **[31:0]result1** | **O** | **Rs寄存器值** |
| **[31:0]result2** | **O** | **Rt寄存器值** |
| **RegWr** | **I** | **写寄存器信号** |
| **[4:0]rw** | **I** | **Rt写寄存器地址** |
| **[31:0]Di** | **I** | **写入寄存器的数据** |

**（3）功能定义**

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **功能描述** |
| **1** | **读寄存器数据** | **读rs、rt寄存器的数据** |
| **2** | **向寄存器写入数据** | **根据写信号向寄存器选择写入数据** |

6）im\_4k模块定义：

**(1) 基本描述**

**指令内存大小为4K，初始化从code.txt载入指令。根据输入的指令地址，输出当前位置存储的指令。**

**(2) 模块接口**

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| **[11:2]addr** | **I** | **指令地址** |
| **[31:0]dout** | **O** | **指令** |

**（3）功能定义**

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **功能描述** |
| **1** | **载入指令** | **初始化载入code.txt中的指令** |
| **2** | **输出指令** | **根据输入的指令地址，输出当前指令** |

7）dm\_4k模块定义：

**(1) 基本描述**

**“数据内存”大小为4K，根据输入的地址读出“数据内存”中的数据，并根据数据写信号，将输入的数据选择写入“数据内存”中。**

**(2) 模块接口**

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| **clk** | **I** | **时钟信号** |
| **[11：2]addr** | **I** | **数据地址** |
| **[31:0]din** | **I** | **写入的数据** |
| **we** | **I** | **数据内存写信号** |
| **[31:0]dout** | **O** | **读出的数据** |

**（3）功能定义**

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **功能描述** |
| **1** | **读数据内存数据** | **根据输入的数据地址，读出数据内存的数据，读出的数据不一定被使用，只有lw指令才会使用。** |
| **2** | **向数据内存写入数据** | **在时钟信号到来时，根据写数据信号，将输入的数据选择写入数据内存中。** |

8) ctrl模块定义:

**(1) 基本描述**

**根据输入的op, func,输出 ExtOp,ALUSrc,ALUctr,RegDst,R\_type,MemWr,Branch,Jump,MemtoReg,RegWr控制信号。利用书上P166表5.4。其中高阻状态设为0。**

**新添加了MemRead信号，判断上一条指令是否是lw,MemRead=1,是load指令，MemRead=0,不是load指令**

**(2) 模块接口**

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| **[5:0]op** | **I** | **操作码** |
| **[5:0]func** | **I** | **指令低六位** |
| **[2:0]ALUctr** | **O** | **ALU控制信号** |
| **RegDst** | **O** | **Rd寄存器控制信号** |
| **ALUSrc** | **O** | **ALU数据来源控制信号** |
| **MemtoReg** | **O** | **数据内存数据写入寄存器控制信号** |
| **RegWr** | **O** | **寄存器写控制信号** |
| **MemWr** | **O** | **数据内存写控制信号** |
| **Branch** | **O** | **Beq指令控制信号** |
| **Jump** | **O** | **J指令控制信号** |
| **Extop** | **0** | **立即数扩展控制信号** |
| **MemRead** | **0** | **判断lw指令信号** |

**（3）功能定义**

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **功能描述** |
| **1** | **输出各种控制信号** | **根据输入的OP，利用真值表化简，输出各种控制信号。** |

9) ZHF模块定义:

**(1) 基本描述**

**转发单元，用于解决数据冒险**

**与之前的不同之处在于具体的计算过程发生了变化。**

**(2) 模块接口**

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| **RegWr\_4** | **I** | **第4阶段的regwr** |
| **RegWr\_5** | **I** | **第5阶段的regwr** |
| **ALUSrc\_4** | **I** | **第4阶段的alusrc** |
| **[4:0]rd\_4** | **I** | **第4阶段的rd** |
| **[4:0]rd\_5** | **I** | **第5阶段的rd** |
| **[4:0]rs\_3** | **I** | **第3阶段的rs** |
| **[4:0]rt\_3** | **I** | **第3阶段的rt** |
| **[1:0]ALUSrcA\_3** | **O** | **第3阶段的alusA** |
| **[1:0]ALUSrcB\_3** | **O** | **第3阶段的alusA** |

**（3）功能定义**

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **功能描述** |
| **1** | **转发单元** | **转发各个阶段的信号** |

10) LoadUse模块定义:

**(1) 基本描述**

**解决load-use数据冒险。根据上一条指令是否为load指令来决定插入气泡，通过禁止pc和if\_id寄存器接受新的数据。**

**(2) 模块接口**

|  |  |  |
| --- | --- | --- |
| **信号名** | **方向** | **描述** |
| **MemRead\_3** | **I** | **第4阶段的MemRead** |
| **[4:0]rt\_2** | **I** | **第2阶段的rt** |
| **[4:0]rt\_3** | **I** | **第3阶段的rt** |
| **[4:0]rs\_2** | **I** | **第2阶段的rs** |
| **stall** | **I** | **清id\_ex信号** |
| **pcwrite** | **I** | **pc写信号** |
| **ifwrite** | **I** | **If\_id写信号** |

**（3）功能定义**

|  |  |  |
| --- | --- | --- |
| **序号** | **功能名称** | **功能描述** |
| **1** | **LoadUse单元** | **解决load-use冒险** |

1. 四个寄存器模块定义:

四个寄存器分别是if\_id, id\_ex, ex\_me, me\_wr

这四个寄存器功能都是将前一个阶段的数据存储一个周期，当时钟下降沿到来时，发送给下一个阶段。因为输入输出变量过多，这里就不详细解释。

与之前的主要区别是，if\_id多了一个写信号，另外其它阶段增添或减少了一些信号。

1. **测试代码及结果**

**与之前的测试代码不同，为了体现branch前移和load-use冒险，重写了测试代码**

汇编代码：

addiu $s1,$s1,100

addiu $s2,$s2,600

add $s3,$s2,$s1（转发）

lw $s4,0($s2)

add $s5,$s4,$s1（load-use冒险）

add $s6,$s2,$s4

sub $s7,$s5,$s1

beq $s7,$zero,else （branch前移）

else:addiu $s1,$s1,200

16进制机器代码：

26310064

26520258

02519820

8e540000

0291a820

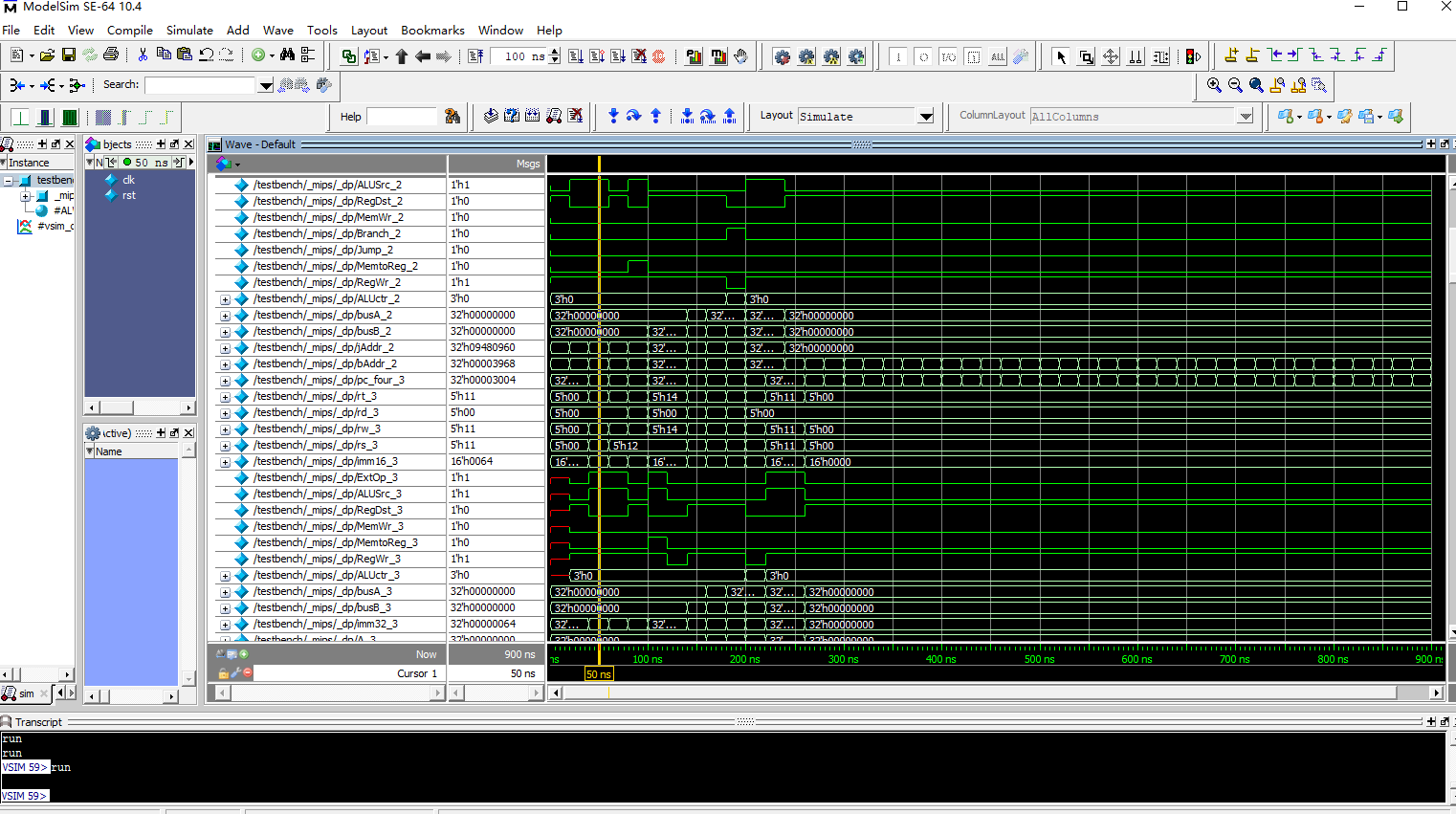
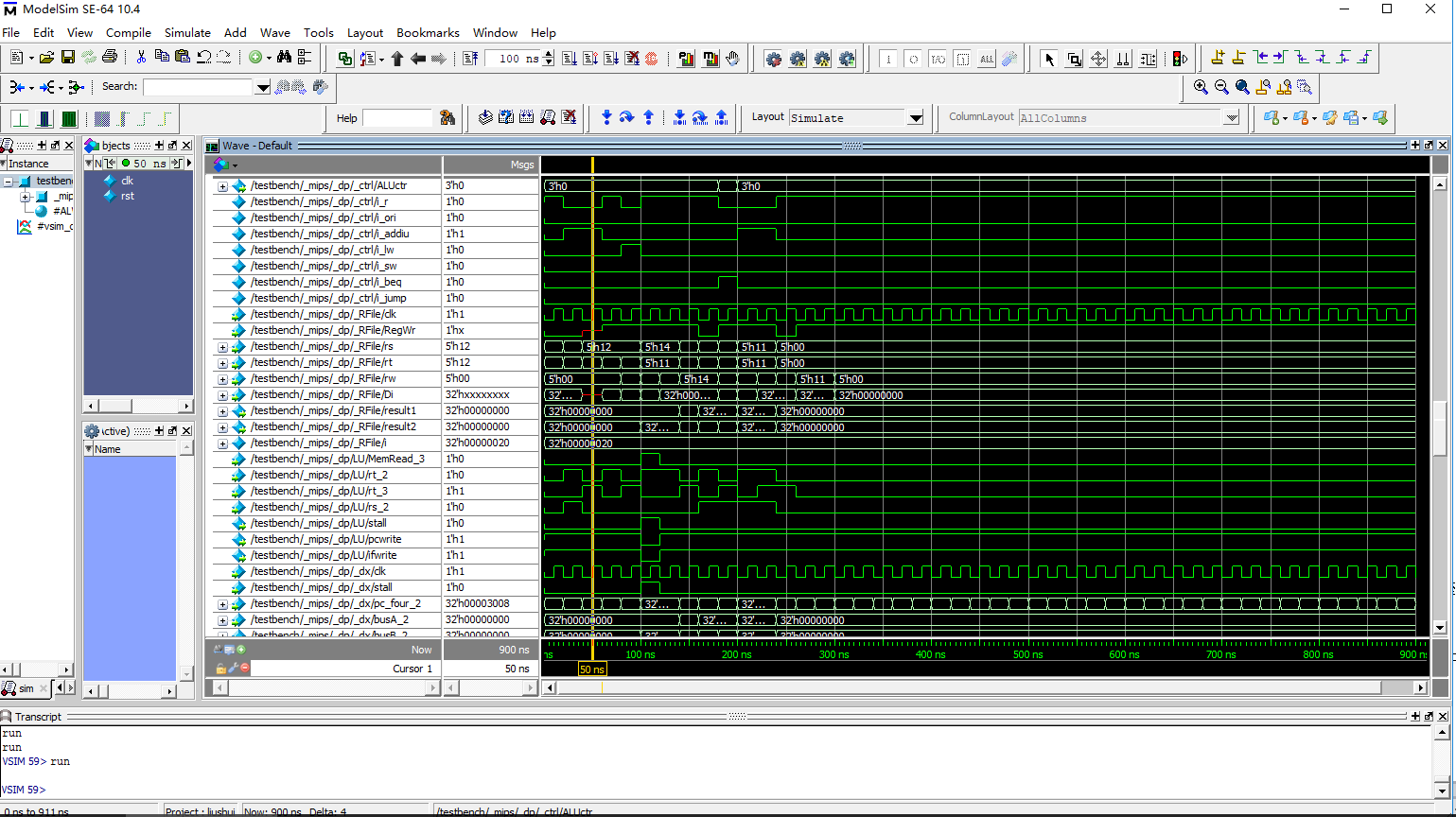
0254b020

02b1b822

12e00000

263100c8

1. **波形展示**



1. **完成时间估计**
2. 总体完成时间在7+小时（在上次的代码上的改动）。

**7、心得体会**

这次的代码比上次的代码主要添加了LoadUse模块解决了load-use冒险，另外把branch前移了。LoadUse解决起来还不算麻烦，只不过要增添一些判断信号，在branch前移的时候，好多之前的信号用不到了，要删除掉。这一增一删，我的信号就有点乱了，搞不太清。。。在改bug时候花了不少时间。如果早知道branch前移并不是太麻烦，在project2的时候就应该做完，就会省了现在的好多时间。