计算机组成原理实验报告

57条指令流水级处理器开发

班级：1616001

姓名：周昊冉 161630224

石一泽 161630131

王未来 161630113

学院：计算机科学与技术

2017.07.06

**文档目录：**

**一、设计说明**

**二、模块化和层次化设计说明**

**三、具体模块定义**

**四、测试代码及结果**

**五、波形展示**

**六、完成时间**

**七、心得体会**

1. **设计说明**：

处理器需要实现的指令包括所有非浮点 MIPS I 指令以及 MIPS32 中的 ERET 指令，有 14 条算术运算 指令、8 条逻辑运算指令，6 条移位指令、12 条分支跳转指令、4 条数据移动指令、2 条自陷指令、8 条 访存指令、3 条特权指令，共计 57 条。下面分类给出各部分指令的执行过程。

算术运算指令

1. R指令的执行过程

有ADD,ADDU,SUB,SUBU,SLT,SLTU。

指令经过译码之后，在ID阶段从rs，rt寄存器取数，送至ALU。在EX阶段进行相应的算术运算。最后，在WB阶段将运算结果送回rd寄存器。

还有MULT,MULTU,DIV,DIVU

指令经过译码之后，在ID阶段从rs，rt寄存器取数，送至ALU。在EX阶段进行相应的算术运算。最后，在WB阶段将运算结果送回LO和HI寄存器。

MULT有符号乘法，寄存器 rs 的值乘以寄存器 rt 的值，乘积的低半部分和高半部分分别写入 LO寄存器和 HI 寄存器。

MULTU无符号乘法，寄存器 rs 的值乘以寄存器 rt 的值，乘积的低半部分和高半部分分别写入 LO寄存器和 HI 寄存器。

DIV有符号除法，寄存器 rs 的值除以寄存器 rt 的值，商写入 LO 寄存器中，余数写入 HI 寄存器中。

DIVU无符号除法，寄存器 rs 的值除以寄存器 rt 的值，商写入 LO 寄存器中，余数写入 HI 寄存器中。

1. I指令的执行过程

有ADDI,ADDIU,SLTI。

指令经过译码之后，在ID阶段从rs寄存器取数，另外一个运算数是指令15...0的立即数,送至ALU。在EX阶段进行相应的算术运算。最后，在WB阶段将运算结果送回rt寄存器。

1. 无J指令

## 逻辑运算指令

1. R指令的执行过程

有AND,NOR,OR,XOR

指令经过译码之后，在ID阶段从rs，rt寄存器取数，送至ALU。在EX阶段进行相应的逻辑运算。最后，在WB阶段将运算结果送回rd寄存器。

1. I指令的执行过程

有ANDI,LUI,ORI,XORI

指令经过译码之后，在ID阶段从rs寄存器取数，另外一个运算数是指令15...0的立即数,送至ALU。在EX阶段进行相应的逻辑运算。最后，在WB阶段将运算结果送回rt寄存器。

还有LUI

指令经过译码之后，在ID阶段将 16 位立即数 imm 送入ALU，把立即数左移16位，最后，在WB阶段将运算结果送回rt寄存器

1. 无J指令

## 移位指令

1. R指令的执行过程

有SLLV，SRAV，SRLV

指令经过译码之后，在ID阶段从rs，rt寄存器取数，送至ALU。在EX阶段进行相应的移位运算。最后，在WB阶段将运算结果送回rd寄存器。

还有SLL,SRA,SRL

指令经过译码之后，在ID阶段从rt寄存器取数，还有指令的10...6位立即数。在EX阶段由立即数 sa 指定移位量，对寄存器 rt 的值进行相应的移位运算。最后，在WB阶段将运算结果送回rd寄存器。

1. 无I指令
2. 无J指令

## 分支跳转指令

1. R指令

JR，JALR

在ID阶段无条件跳转。JR指令跳转目标是寄存器rs的值。JALR跳转目标是寄存器 rs 中的值。同时将该分支对应延迟槽指令之后的指令的 PC值保存至寄存器 rd 中。

1. I指令

有BEQ,BNE,BGEZ,BGTZ,BLEZ,BLTZ,BGEZAL.BLTZAL

在ID阶段，对寄存器的值进行判断（如相等，不等，大于等于...）若满足条件，则转移，否则顺序执行。转移目标由立即数 offset（15...0）左移 2 位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到。

1. J指令

有J，JAL

在ID阶段无条件跳转。J指令跳转目标由该分支指令对应的延迟槽指令的 PC 的最高 4 位与立即数左移 2 位后的值拼接得到。JAL还需要将该分支对应延迟槽指令之后的指令的 PC 值保存至第 31 号通用寄存器中。

## 数据移动指令

有MFHI, MFLO,MTLO,MTHI

我们设计的HI和LO寄存器独立于几个阶段之外，随时可读可写。因为其它指令访问寄存器的值必须依靠数据移动指令移动到rd寄存器中，所以不会产生值的错误。

MFHI将 HI 寄存器的值写入到寄存器 rd 中。

MFLO将LO 寄存器的值写入到寄存器 rd 中。

MTHI将寄存器 rs 的值写入到 HI 寄存器中。

MTLO将寄存器 rs 的值写入到 LO 寄存器中。

## 自陷指令

BREAK触发断点例

SYSCALL触发系统调用例外。

包含了cause status EPC三个寄存器。break和syscall指令进行报错，cause寄存器记录下错误原因，status记录下当前是出错的状态，EPC保留PC值，目的是进行返回。

## 访存指令

1. 无R指令
2. I指令  
    有LW,SW,LB,LBU,LH,LHU,SH,SB。

指令经过译码之后，在ID阶段从base寄存器取数，另外一个运算数是指令15...0的立即数offset（符号扩展成32位）,送至ALU。在EX阶段相加得到访存的虚地址。在MEM阶段将虚地址位置的值存入rt寄存器或者将rt寄存器的值存入虚地址位置。

1. 无J指令

## 特权指令

ERET从中断、例外处理返回。

MFCO从协处理器 0 的寄存器取值

MTCO向协处理器 0 的寄存器存值

1. **模块化和层次设计说明：**

Mips.v

Datapath.v

ctrl.v

Aluctrl.v

Pc.v

Alu.v

Im.v

Dm.v

Regfile.v

forward

Hazard detection

Mux.v

Testbench.v

Reg.v

Code.txt

1. **具体模块定义**

### **1、datapath模块定义**

1. 基本描述

Datapath将流水处理器的5个阶段连在一起，IF，ID，Ex，Mem，WB。

1. 具体描述

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| Rst | I | 重置信号 |
| regdst | I | 决定写寄存器  0：写入寄存器rd  1：写入寄存器rt |
| alusrc | I | 决定alu的第二个数据来源  1：alu的第二个数据来自extender扩展后的数  0：alu的第二个数据来自读出的第二个寄存器的值 |
| memtoreg | I | 写入寄存器数据来源  0：ALU计算的数据写入寄存器  1：数据从DM写入寄存器 |
| regwr | I | 寄存器是否可写入数据  1：可写入数据  0：不可写入数据 |
| memwr | I | DM是否可写入数据  1：可写入数据  0：不可写入数据 |
| branch | I | 判断指令是否是分支beq指令  1：指令是beq指令  0：指令非beq指令 |
| jump | I | 判断指令是否是跳转指令  0：不是跳转指令  1：是跳转指令 |
| extop | I | 控制extender进行何种扩展  0：零扩展  1：符号扩展 |
| aluctr[2:0] | I | 控制ALU执行何种运算  具体定义同上 |
| rs[4:0] | I | 指令25...21位为寄存器rs |
| rt[4:0] | I | 指令20...16位为寄存器rt |
| rd[4:0] | I | 指令15...11位为寄存器rd |
| Imm[15:0] | I | 指令15：0位为16位立即数 |
| Branch\_type[2:0] | I | 用于branch指令的几个类型 |
| JumpDst | I | 判断JR和JALR指令 |
| Reg31Wr | I | 是否写入第31号寄存器 |
| Instr\_outIFID | O | 取出的指令 |

|  |  |
| --- | --- |
| 阶段 | 功能 |
| IF | 更新PC的值，取指令部件。  包括寄存器reg\_IFID，pc。 |
| ID | 发出instr，并接受ctrl传来的控制信号，进行通用寄存器的读取。  包括寄存器reg\_IDEx，通用寄存器。 |
| Ex | alu计算，并计算branch跳转的PC的值。  包括寄存器reg\_ExMem，alu。 |
| Mem | 执行dm的存取，判断branch指令。  包括寄存器reg\_MemWB，dm。 |
| WB | 将数据写入通用寄存器。 |

3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 硬件布置 | 将各个阶段的各个部件之间连接起来 |
| 2 | 接受信号 | 将指令转给ctrl，再接受来自ctrl的控制信号 |

**2、pc模块定义**

1. 基本描述

Pc模块的功能为输出当前指令地址并保存下一条指令地址。复位时，PC指向0x0000\_3000，目的是与MARS的Memory Configuration相配合。

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| outpc[31:2] | I | 下条指令的地址（出pc） |
| clk | I | 时钟信号 |
| rst | I | 复位信号  1：复位  0：无效 |
| inpc[31:2] | O | 32位指令存储器地址（入pc） |
| Pc\_Wr | I | 是否进行pc的改写，用于stall |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 复位 | 当复位信号有效时，PC被设置为0x0000\_3000。 |
| 2 | 保存NPC并输出 | 在每个clock的上升沿保存NPC，并输出。 |
| 3 | Stall | 进行阻塞 |

**3**、**IM模块定义**

1）基本描述

IM的主要功能是存储指令，根据地址输出相应指令。用1024个32位的寄存器来保存指令。

2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| addr[11:2] | I | PC[31:2]的后十位 |
| Dout[31:0] | O | 当前32位指令 |

3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 读取指令并输出 | 根据输入地址读取指令并输出 |

**4、Register模块定义**

1）基本描述

实现通用寄存器的模块，完成寄存器的读和取数据及数据回写到寄存器的功能。其中有32个32位寄存器来存储和读取数据。

有两个单独的寄存器HI，LO。主要用于存放乘除法的运算结果。这两个寄存器响应时钟下降沿，如果写信号使能，则可以写入数据。随时可读。

1. 模块接口

Register：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| rs[25:21] | I | 根据指令25...21位读寄存器rs |
| rt[20:16] | I | 根据指令20...16位读寄存器rt |
| rw[4:0] | I | 根据指令选择被写入数据的寄存器 |
| busW[31:0] | I | 回写的数据 |
| clk | I | 时钟信号 |
| Regwr | I | 寄存器写使能信号  1：寄存器可以写入数据  0：寄存器不能写入数据 |
| Reg31Wr | I | 是否写入第31号寄存器 |
| busA[31:0] | O | 读寄存器rs中的数据 |
| busB[31:0] | O | 读寄存器rt中的数据 |

reg\_HI:

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| din[31:0] | I | 写入的数据 |
| clk | I | 时钟信号 |
| wr | I | 寄存器写使能信号  1：寄存器可以写入数据  0：寄存器不能写入数据 |
| dout[31:0] | O | 读寄存器HI中的数据 |

reg\_LO:

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| din[31:0] | I | 写入的数据 |
| clk | I | 时钟信号 |
| wr | I | 寄存器写使能信号  1：寄存器可以写入数据  0：寄存器不能写入数据 |
| dout[31:0] | O | 读寄存器HI中的数据 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 读寄存器 | 根据指令从32个寄存器或HI或LO寄存器中读取指定寄存器 |
| 2 | 读数据 | 从指定寄存器中读取数据 |
| 3 | 回写数据 | 将数据回写到指定寄存器中 |

**5、ALU模块定义**

1）基本描述

ALU的主要功能是完成数据的运算。

ALU可以完成的运算有：

Add, Addu, Sub, Subu, Mul,Mulu,Div, Divu, Slt, Sltu, And, Or, Xor Nor , Lui, Sll, Srl, Sra;

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| a[31:0] | I | 数据1 |
| b[31:0] | I | 数据2 |
| aluctr[4:0] | I | 控制ALU执行何种运算  举个例子：Add = 5’b00000，则执行a+b。 |
| result[31:0] | O | 两个数据运算结果 |
| zero | O | 判断beq指令下两数据相减是否为0.  1：两数据相减为0  0：两数据相减非0 |
| HI\_result[31:0] | O | 乘积的高半部分写入HI 寄存器。除法的余数写入 HI 寄存器中。 |
| LO\_result[31:0] | O | 乘积的低半部分写入LO 寄存器。除法的商写入LO寄存器中。 |
| HI\_wr | O | HI寄存器的写信号 |
| LO\_wr | O | LO寄存器的写信号 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 运算 | 将两数据进行相应运算 |

### **6、DM模块定义**

1. 基本描述

DM的主要功能是数据的存储和读出。DM里面有1024个32位寄存器，可以根据读入的地址读出相应数据或存入数据。

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| addr[11:2] | I | 读入的地址、来自alu结果的11...2位 |
| din[31:0] | I | 写入数据 |
| Read | I | 决定数据是否可读取  1：数据可以读取  0：数据不可读取 |
| Wr | I | 决定数据是否可写入  1：数据可以写入  0：数据不可写入 |
| clk | I | 时钟信号 |
| dout[31:0] | O | 读出数据 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 读数据 | 根据读入的地址寻得该地址存的数据并读出，用于lw、lb、lbu、lh、lhu指令 |
| 2 | 写数据 | Sw、sb、sh指令中将读出的第二个寄存器的值存入DM |

### **7、controller模块定义**

1. 基本描述

Controller的主要功能是控制各个部件的执行功能及多路选择器。Controller是根据指令决定各个控制信号的。

ALUcontroller包括在控制器中。

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Ins[31:0] | I | 当前的整条32位指令 |
| opcode[31:26] | O | 指令Instr[31:0]的31...26位 |
| func[5:0] | O | 指令Instr[31:0]的5...0位 |
| regdst | O | 决定写寄存器  0：写入寄存器rd  1：写入寄存器rt |
| alusrc | O | 决定alu的第二个数据来源  1：alu的第二个数据来自extender扩展后的数  0：alu的第二个数据来自读出的第二个寄存器的值 |
| memtoreg | O | 写入寄存器数据来源  0：ALU计算的数据写入寄存器  1：数据从DM写入寄存器 |
| regwr | O | 寄存器是否可写入数据  1：可写入数据  0：不可写入数据 |
| memwr | O | DM是否可写入数据  1：可写入数据  0：不可写入数据 |
| branch | O | 判断指令是否是分支beq指令  1：指令是beq指令  0：指令非beq指令 |
| jump | O | 判断指令是否是跳转指令  0：不是跳转指令  1：是跳转指令 |
| extop | O | 控制extender进行何种扩展  0：零扩展  1：符号扩展 |
| aluctr[2:0] | O | 控制ALU执行何种运算  具体定义同上 |
| rs[4:0] | O | 指令25...21位为寄存器rs |
| rt[4:0] | O | 指令20...16位为寄存器rt |
| rd[4:0] | O | 指令15...11位为寄存器rd |
| Imm[15:0] | O | 指令15：0位为16位立即数 |
| Branch\_type[2:0] | O | 用于branch指令的几个类型 |
| JumpLink | O | 判断JALR和JAL指令 |
| JumpDst | O | 判断JR和JALR指令 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 控制 | 产生控制信号控制各部件执行的功能 |

### **Forwarding Unit模块定义**

1. 基本描述

控制转发。由于流水级处理器可能会导致数据冒险和结构冒险，需要进行数据的转发，通过Forwarding Unit来控制转发。

具体实现见下文。

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| rs\_reg[4:0] | I | 输入regWr信号和寄存器rs，rt，rw的位置。用于判断转发。 |
| rt\_reg[4:0] |
| rs\_IDEx[4:0] |
| rt\_IDEx[4:0] |
| rw\_ExMem[4:0] |
| rw\_MemWB[4:0] |
| ALUSrc\_IDEx |
| RegWr\_ExMem |
| RegWr\_MemWB |
| ForwardA | O | 控制ALU输入口A的转发。 |
| ForwardB | O | 控制ALU输入口B的转发。 |
| ForwardC | O | 控制ALU输入口C的转发。 |
| ForwardD | O | 控制ALU输入口D的转发。 |

### 3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 控制 | 控制转发 |

### **Hazard Detection Unit模块定义**

1）基本描述

控制冒险detection。判断是否需要stall和插入气泡。

具体实现见下文。

2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| rs\_IFID[4:0] | I | 输入regWr信号和寄存器rs，rt，rw的位置。用于判断冒险。 |
| rt\_IFID[4:0] |
| rt\_IDEx[4:0] |
| rw\_Ex[4:0] |
| rw\_ExMem[4:0] |
| MemtoReg\_IDEx |
| RegWr\_ExMem |
| RegWr\_MemWB |
| branch\_ID |
| PC\_Wr | O | 用于控制stall。 |
| IFID\_Wr |
| ctrl\_Wr |

### 3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 控制 | 控制stall |

### **9、mips模块定义**

1. 基本描述

mips是datapath和controller的组合，可以通过clk和rst来控制mips指令的执行。

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| rst | I | 复位信号  1：复位  0：无效 |

3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 最高层架构 | 将datapath 和controller组合起来 |

### **10、 mux模块定义**

1. 基本描述

Mux是多路选择器模块，根据选择信号从多个输入信号中选择一个作为输出信号。

1. 模块接口

|  |  |  |  |
| --- | --- | --- | --- |
| 模块名 | 信号名 | 方向 | 描述 |
| Mux2（32位） | x[31:0] | I | 第一个32位数据 |
| y[31:0] | I | 第二个32位数据 |
| s | I | 选择信号 |
| z[31:0] | O | 输出32位数据 |
| Mux2\_3（3位） | x[2:0] | I | 第一个3位数据 |
| y[2:0] | I | 第二个3位数据 |
| s | I | 选择信号 |
| z[2:0] | O | 输出3位数据 |
| Mux2\_1（1位） | x | I | 第一个1位数据 |
| y | I | 第二个1位数据 |
| s | I | 选择信号 |
| z | O | 输出1位数据 |
| Mux3\_32（32位） | x[31:0] | I | 第一个32位数据 |
| y[31:0] | I | 第二个32位数据 |
| Z[31:0] | I | 第三个32位数据 |
| s[1:0] | I | 选择信号 |
|  | dout[31:0] | O | 输出32位数据 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 选择信号 | 根据选择信号从多个输入中选择一个作为输出信号 |

### **11、reg模块定义**

1. 基本描述

5级流水两两之间需要有4个寄存器，用于储存当前阶段的控制信号。4个寄存器分别为reg\_IFID，reg\_IDEx，reg\_ExMem，reg\_MemWB

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| rst | I | 复位信号  1：复位  0：无效 |
| 所需控制信号 | I | 输入需要保存的控制信号。 |
| 输出控制信号 | O | 输出已保存的控制信号。 |

3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 描述 |
| 1 | 寄存器 | 存储控制信号。 |

1. **测试代码及结果**

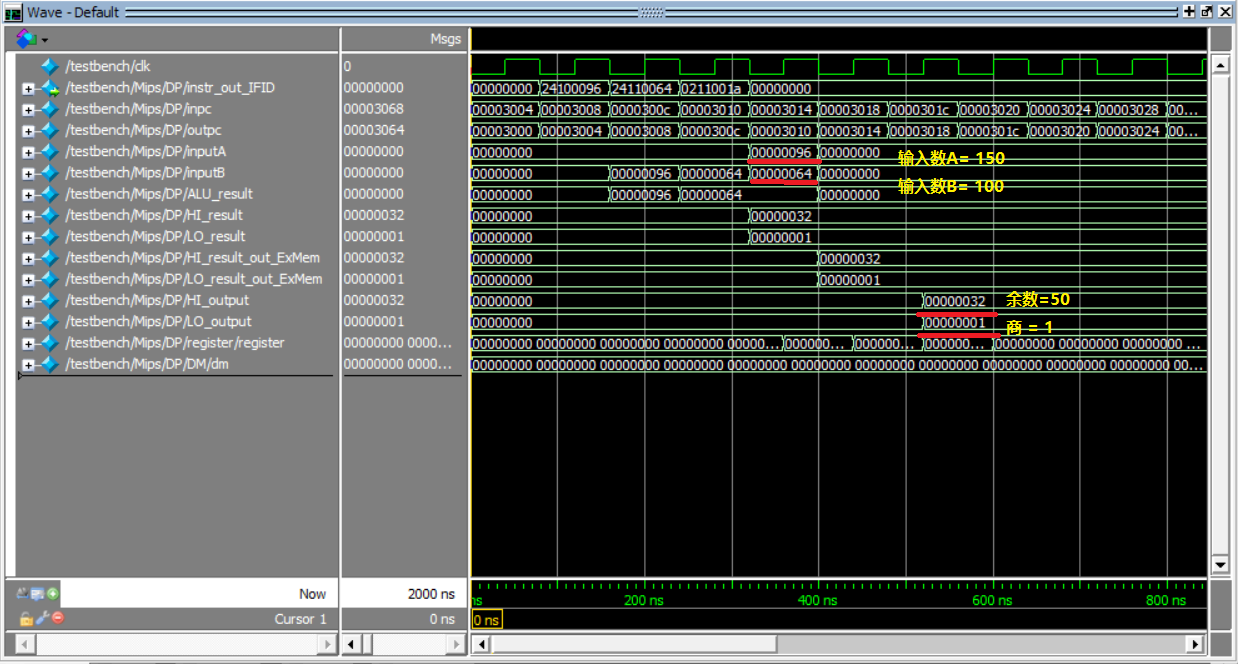
部分指令单独测试：

1. Div

测试代码：

addiu $s0, $zero, 150  
addiu $s1, $zero, 100  
div $s0, $s1

波形图：



1. Mult

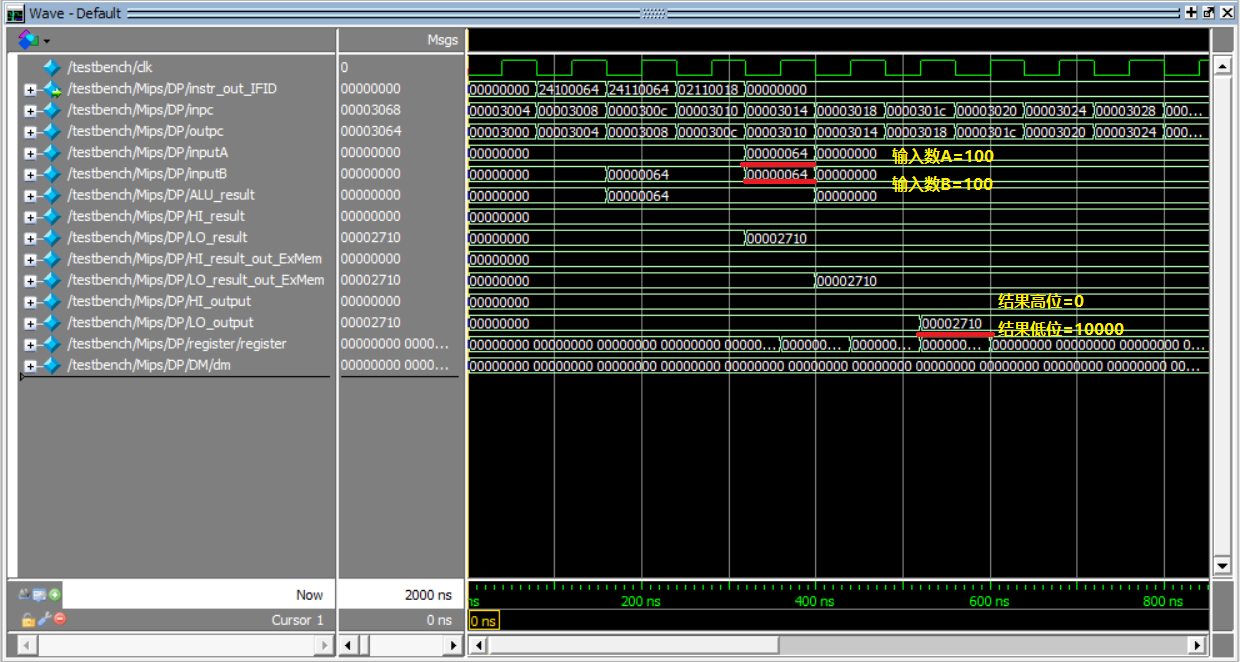
测试代码：

addiu $s0, $zero, 100

addiu $s1, $zero, 100

mult $s0, $s1

波形图：



1. Jal

测试代码：

addiu $s0, $zero, 100

addiu $s1, $zero, 150

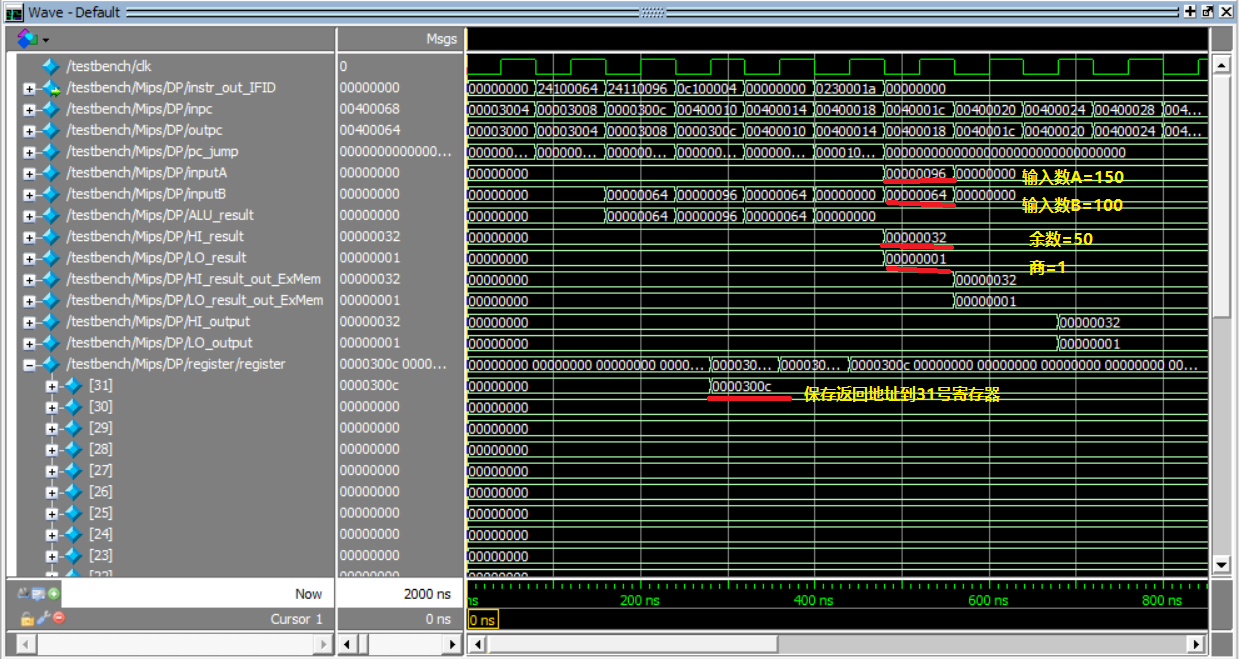
jal start

mult $s0, $s1

start:

div $s1, $s0

波形图：



1. **波形展示**

测试代码：

addiu $s1, $zero, 100

addiu $s2, $zero, 1000

addiu $s5, $zero, 100

ori $s2, $s2, 1

start:

add $s1, $s1, $s1

slt $s4, $s1, $s2

beq $s4, $zero, else

j start

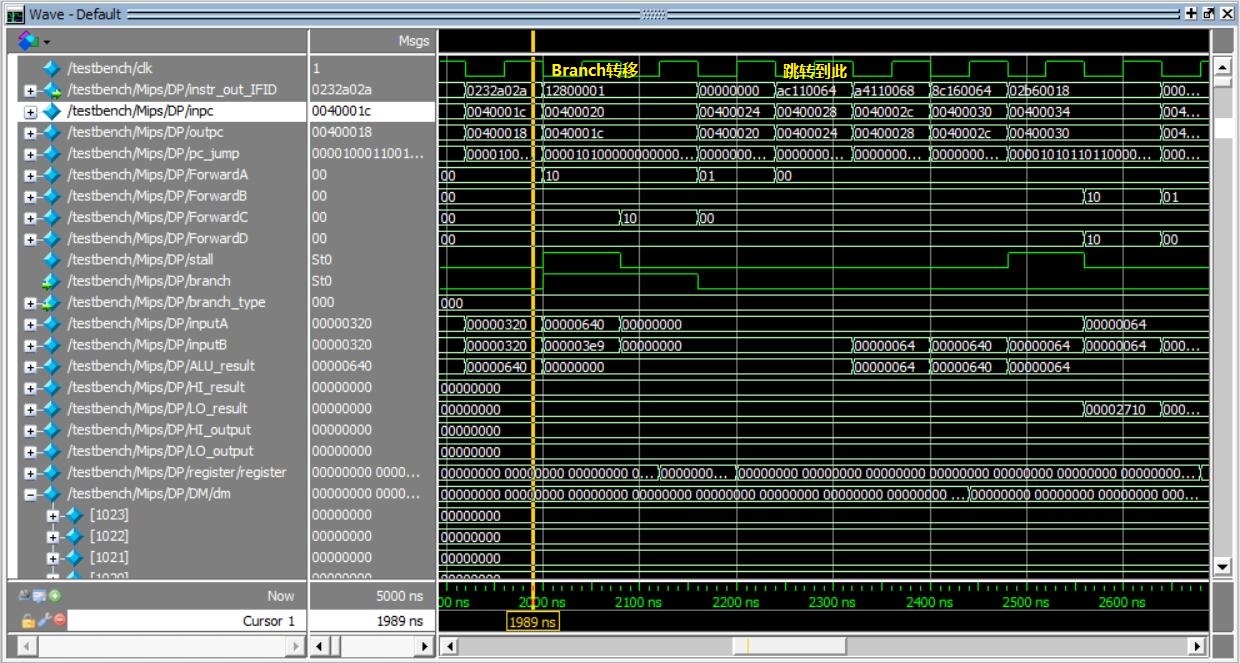
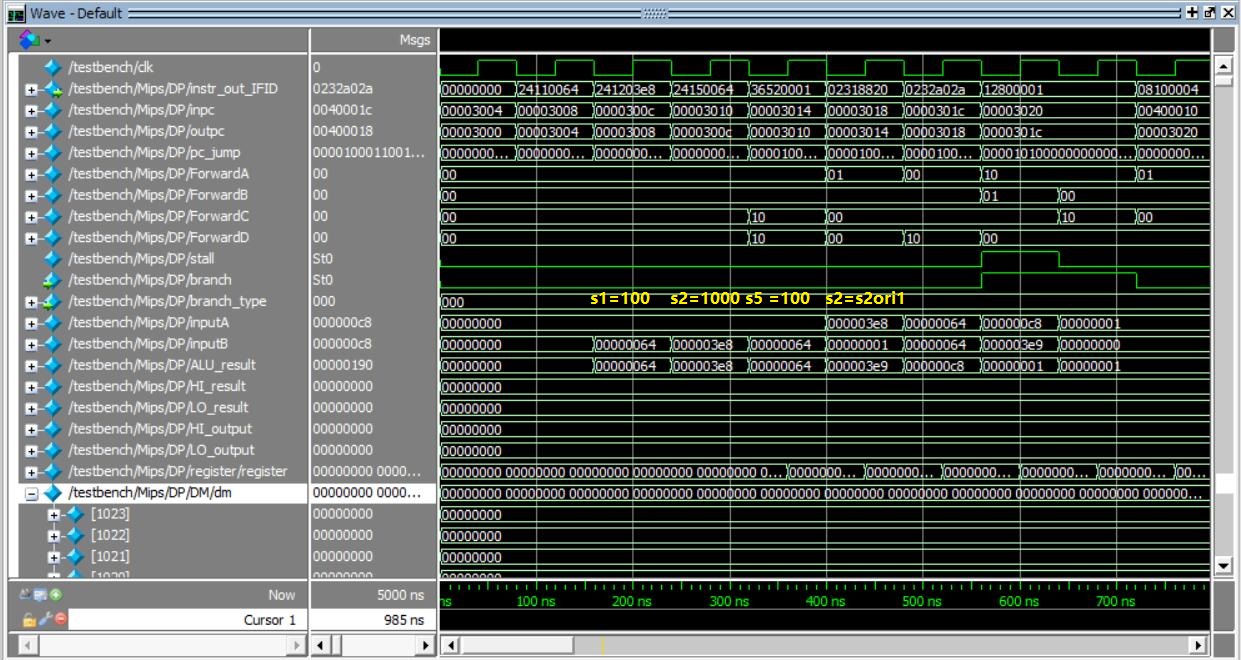
else:

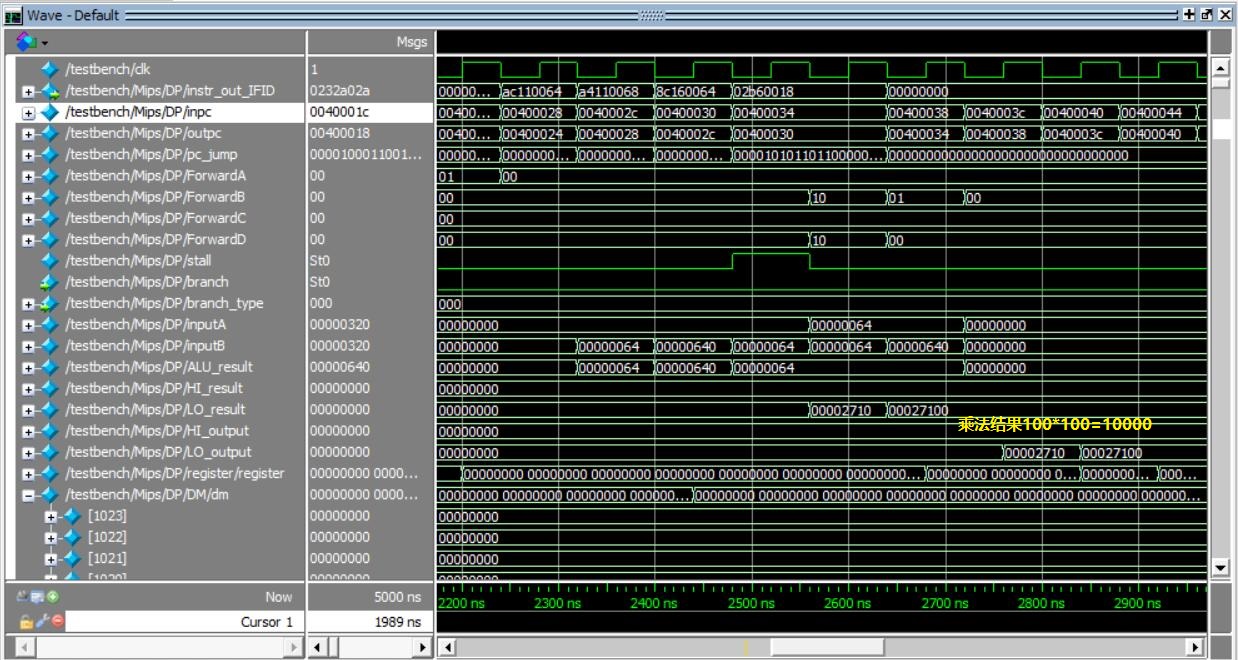
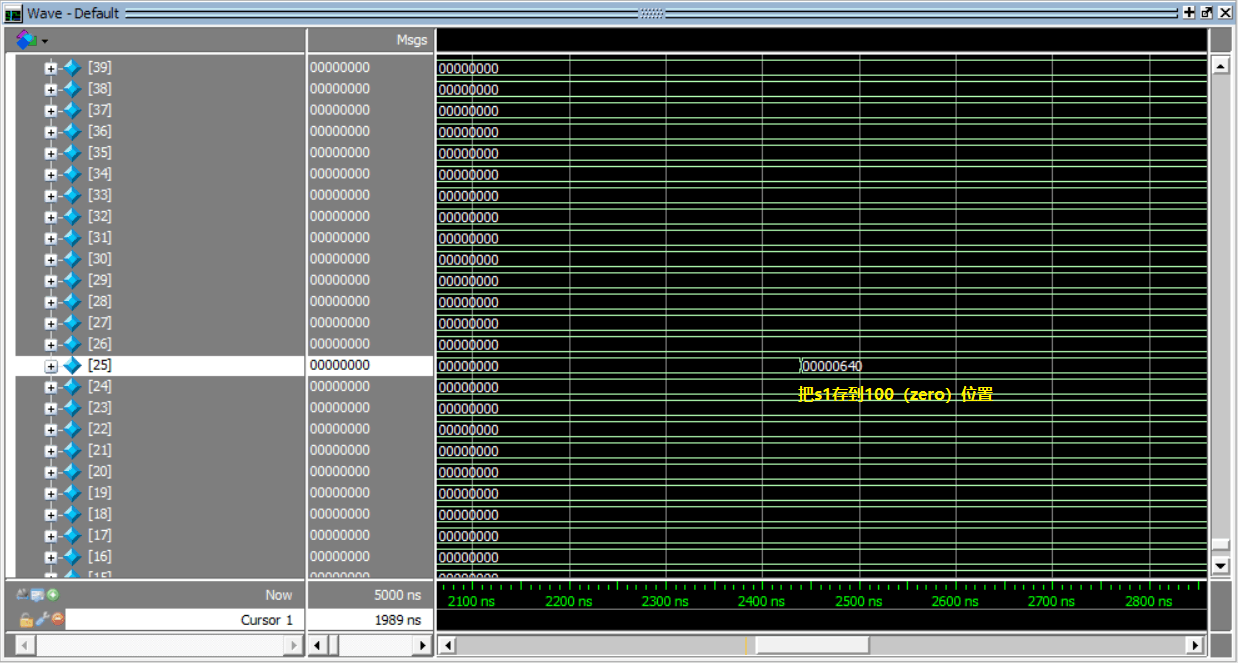
sw $s1, 100($zero)

lw $s6, 100($zero)

mult $s5, $s6

波形图：



****

1. **完成时间**

整理信号、理解指令实现代码：5~6h

完成代码：10+h

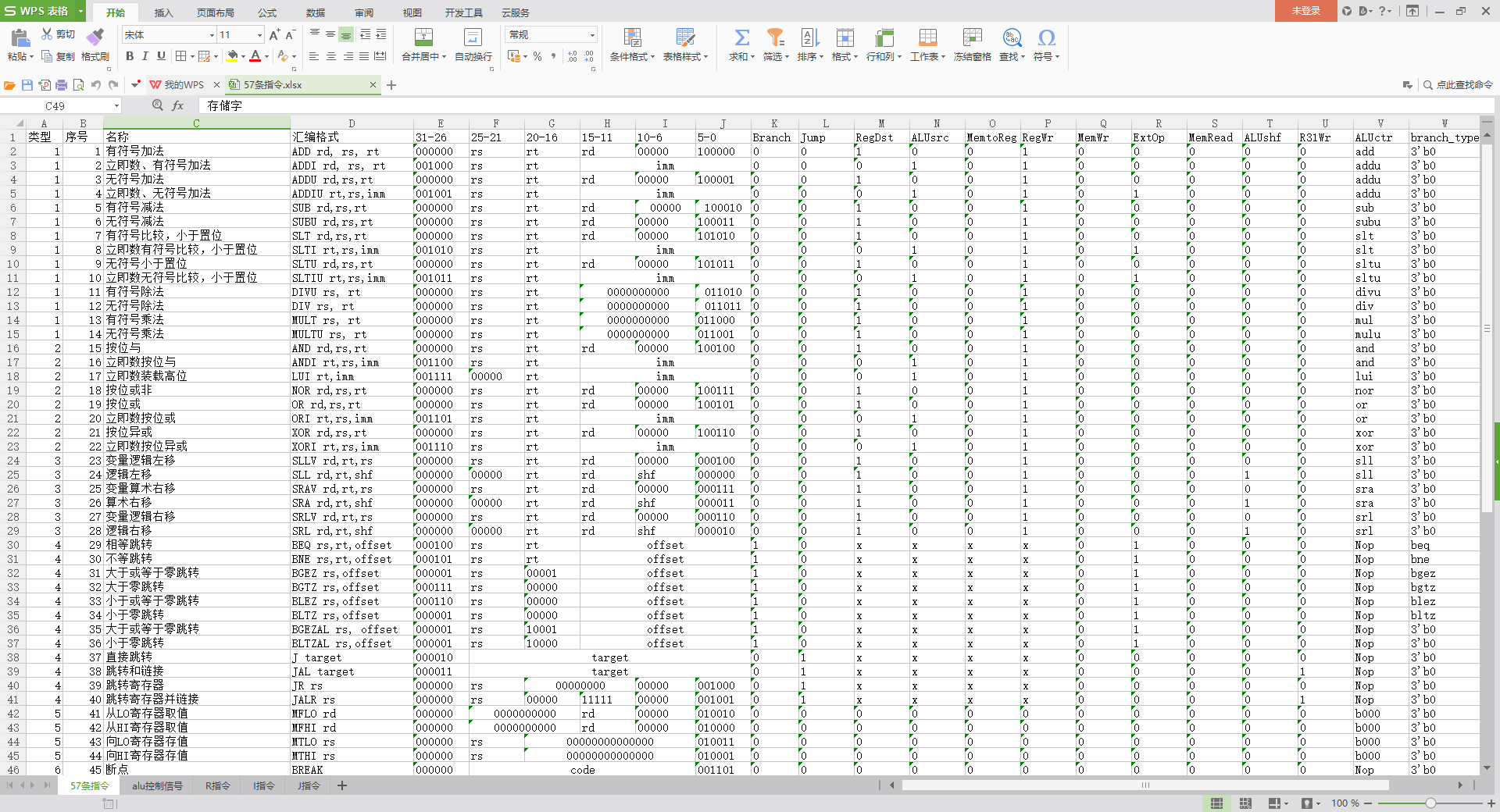
测试代码：4~5h

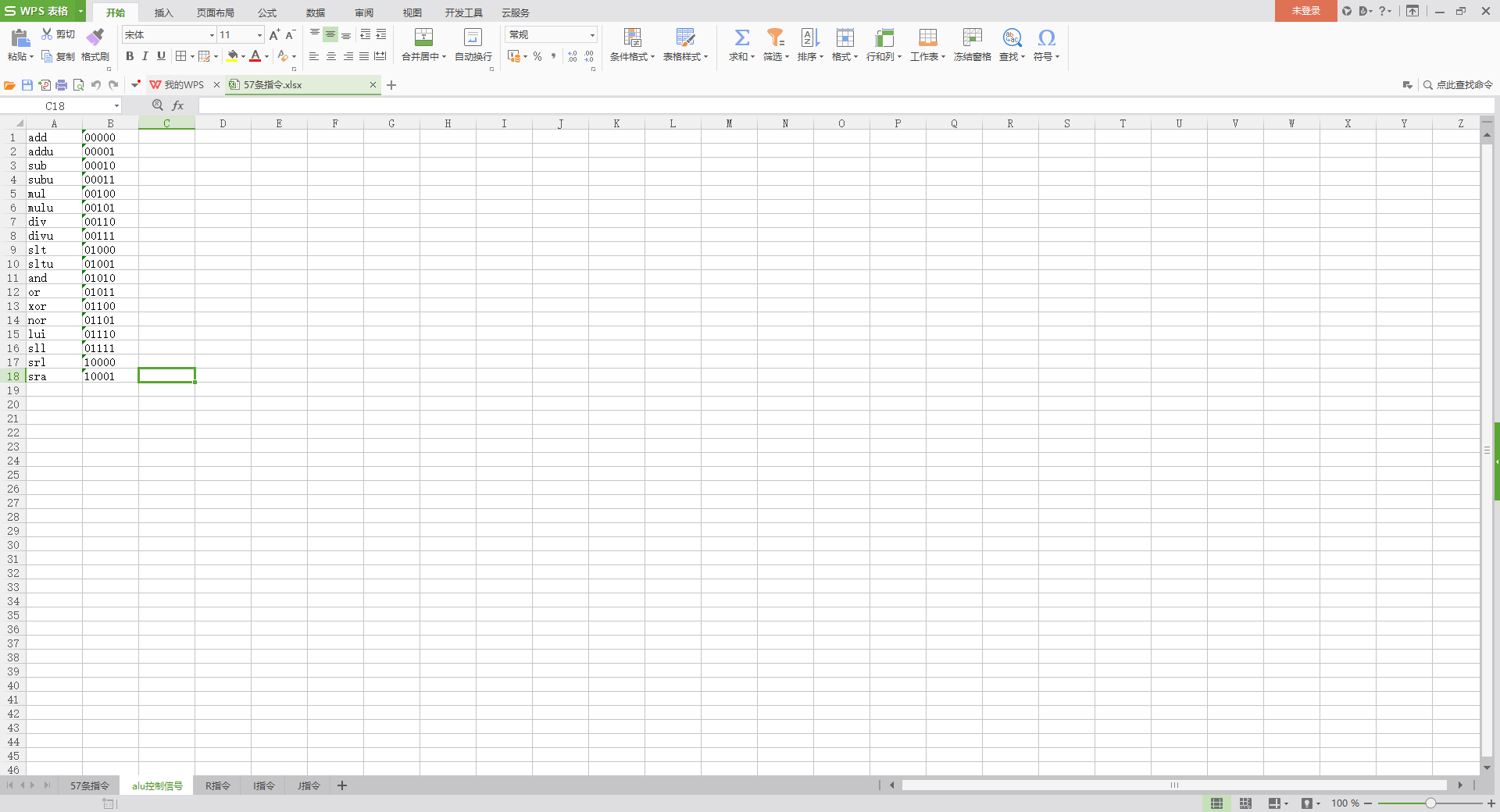
总共完成时间：20h左右

1. **心得体会**

57条的流水cpu中，大部分的指令的实现方式与我们之前做的project2和project3的类似，再alu中进行不同的操作就可以实现不同的算术和逻辑类指令。但是因为指令的数量大大提高了，因此对控制信号的选取有了更高的要求，故我们将控制信号和aluctr等信号做成表格，方便实现。

如图。





在完成信号整理之后，开始正式写代码了。

代码主要对Controller、ALUCtr 和 ALU 做了修改。将之前整理的指令转换为常量编码，再使用case和if指令，实现控制信号和指令之间的对应关系。因为指令编码和对应的控制变量特别的多，我们不停的在写代码和表格界面切换，一条条的对照，仔细的检查每一条指令，才能保证其正确性，为此花费了大量的时间和精力。

本次实验，使我对流水处理器的是实现有了更深刻的理解。对于移位和数据转移指令，最开始我们是将这两类指令单独处理，这样的好处是在第二或者第三阶段就可以将结果存入寄存器。但是这不符合流水的思想，并且可能会导致移位和数据转移指令在实现的过程中，与其它指令发生冲突，因此我们将这两类指令的实现延长到5个周期将除了branch和jump的指令都做成5级流水。

我们花了大量的时间进行代码的测试，但是在代码中可能还是存在许多错误仍未发现，对于单独的指令可以正确的执行，但是当指令放到整个程序中实现时，就可能会导致很多无法预料的错误。57条指令的流水cpu是我们首次接触到的较大的项目实验，对我们未来做更大的项目或软件有着重要的意义。我们会继续完善流水cpu，好的软件需要不断的改错和纠正。