**中山大学数据科学与计算机学院本科生实验报告**

**（2019学年秋季学期）**

课程名称：**计算机组成原理实验**  任课教师： 郭雪梅 助教：曹忠平 丁文

|  |  |  |  |
| --- | --- | --- | --- |
| 年级&班级 | **20级** | 专业(方向) | **保密管理** |
| 学号 | **20336006** | 姓名 | **李英伟** |
| 电话 | **13602447660** | Email | **liyw78@mail2.sysu.edu.cn** |
| 开始日期 | **2021/11/26** | 完成日期 | **2021/12/17** |

**一、实验题目**

单时钟周期 CPU 的设计实验

1. **实验目的**

1. 理解MIPS常用的指令系统并掌握单周期CPU的工作原理与逻辑功能实现。

2. 通过对单周期CPU的运行状况进行观察和分析，进一步加深理解。

**三、实验内容**

**1.** 实验原理

1. **单时钟周期CPU**

单周期 CPU 的特点是每条指令的执行只需要一个时钟周期，一条指令执行完再执行下一条指令。再这一个周期中，完成更新地址，取指，解码，执行，内存操作以及寄存器操作。由于每个时钟上升沿时更新地址，因此要在上升沿到来之前完成所有运算，而这所有的运算除可以利用一个下降沿外，只能通过组合逻辑解决。这给寄存器和存储器RAM的制作带来了些许难度。且因为每个时钟周期的时间长短必须统一，因此在确定时钟周期的时间长度时，要依照最长延迟的指令时间来定，这也限制了它的执行效率。

单周期CPU在每个CLK上升沿时更新PC，并读取新的指令。此指令无论执行时间长短，都必须在下一个上升沿到来之前完成。其时序示意如图Ⅰ。

指令：

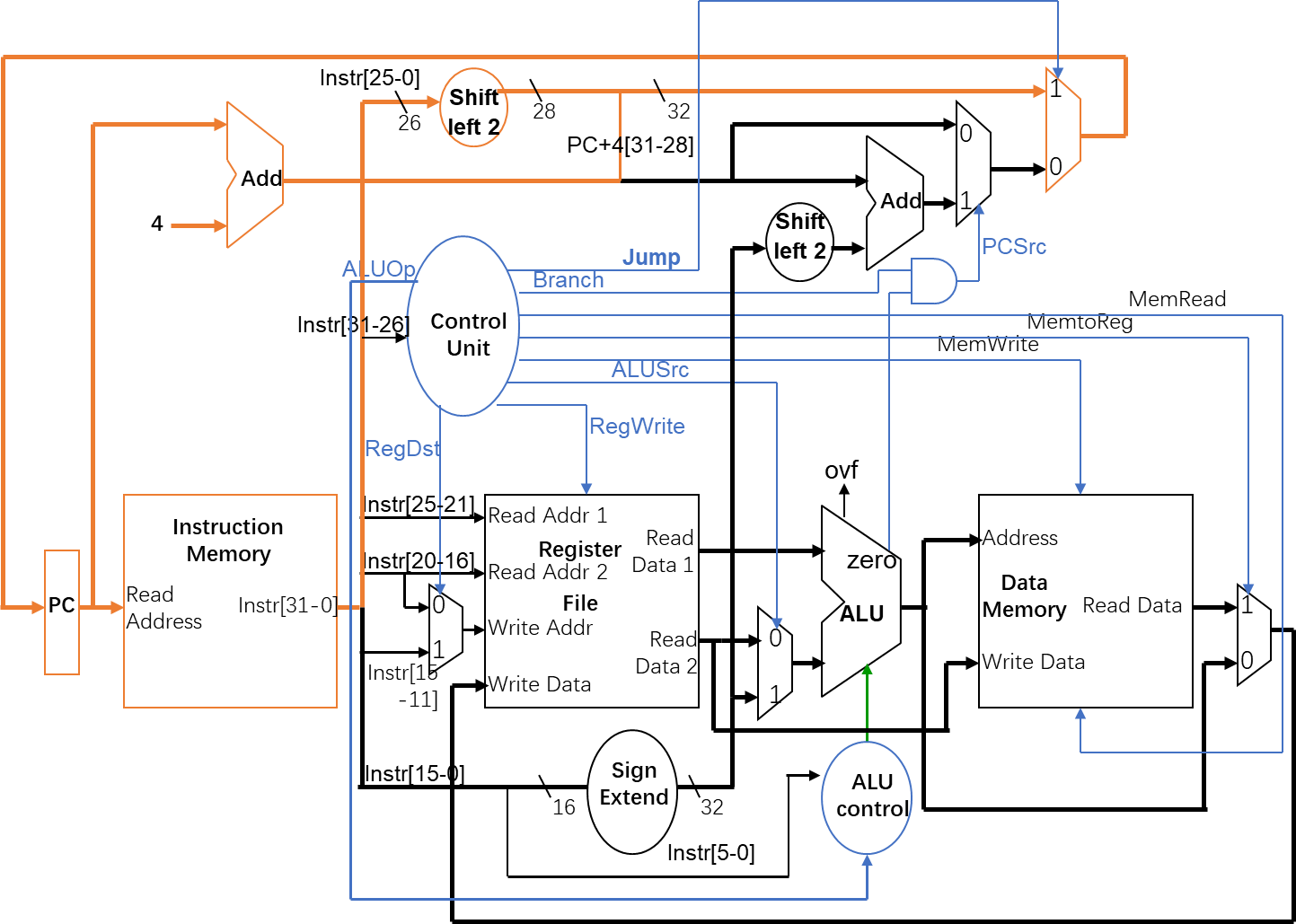
add lw

CLK:

j

图I 单时钟周期CPU时序示意图

1. 下图是一个单周期 CPU 的顶层结构实现。主要器件有程序计数器PC、程序存储器、寄存器堆、ALU、数据存储器和控制部件等。所有的控制信号简单地说明如下：

****

图Ⅱ 单时钟周期CPU详细逻辑设计图参考

其中，控制单元(Ctrl Unit)定义如下：

（1）JUMP：为1时，选择跳转目标地址；为 0时，选择由Branch选出的地址；

（2）MemToReg：为1时，选择存储器数据；为0时，选择ALU 输出的数据；

（3）Branch：为1时，选择转移目标地址；为0时，选择PC +4（图中的 NextPC）；

（4）MemWrite：为1时写入存储器。存储器地址由ALU的输出决定，写入数据为寄存器rt的内容；

（5）ALUOP：ALU控制码；

（6）ALUSrc：ALU操作数B的选择，为1时，选择扩展的立即数；为0时，选择寄存器数据；

（7）RegWrite：为1时写入寄存器堆，目的寄存器号是由RegDst选出的rt或rd，写入数据是由MemToReg选出的存储器数据或ALU的输出结果；

（8）ExtOp：符号扩展。为1时，符号扩展；为0时，0扩展；

（9）RegDst：目的地址，为1时，选择rd；为0时，选择rt。

1. **MIPS指令集**

本次实验共涉及三种类型的MIPS指令，分别为R型、I型和J型，三种类型的MIPS指令格式定义如下：

* R（register）类型的指令从寄存器堆中读取两个源操作数，计算结果写回寄存器堆；
* I（immediate）类型的指令使用一个 16位的立即数作为一个源操作数；
* J（jump）类型的指令使用一个 26位立即数作为跳转的目标地址（target address）；



图Ⅲ MIPS指令集

一条指令的执行过程一般有下面的五个阶段，指令的执行过程就是这五个状态的重复过程：



图IV MIPS指令集

在本次实验中，至少要完成16条指令的功能，包括R型、I型和J型，beq型。。

MIPS的31种指令

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **助记符** | **指 令 格 式** | | | | | | **示 例** | **示例含义** | **操作及解释** |
| BIT # | 31..26 | 25..21 | 20..16 | 15..11 | 10..6 | 5..0 |  |  |  |
| R-类型 | op | rs | rt | rd | shamt | func |  |  |  |
| **add** | 000000 | rs | rt | rd | 00000 | 100000 | add $1,$2,$3 | $1=$2+S3 | (rd)←(rs)+(rt); rs=$2,rt=$3,rd=$1 |
| **addu** | 000000 | rs | rt | rd | 00000 | 100001 | addu $1,$2,$3 | $1=$2+S3 | (rd)←(rs)+(rt); rs=$2,rt=$3,rd=$1,无符号数 |
| **sub** | 000000 | rs | rt | rd | 00000 | 100010 | sub $1,$2,$3 | $1=$2-S3 | (rd)←(rs)-(rt); rs=$2,rt=$3,rd=$1 |
| **subu** | 000000 | rs | rt | rd | 00000 | 100011 | subu $1,$2,$3 | $1=$2-S3 | (rd)←(rs)-(rt); rs=$2,rt=$3,rd=$1,无符号数 |
| **and** | **000000** | rs | rt | rd | 00000 | 100100 | and $1,$2,$3 | $1=$2&S3 | (rd)←(rs)&(rt); rs=$2,rt=$3,rd=$1 |
| **or** | 000000 | rs | rt | rd | 00000 | 100101 | or $1,$2,$3 | $1=$2|S3 | (rd)←(rs) | (rt); rs=$2,rt=$3,rd=$1 |
| **xor** | 000000 | rs | rt | rd | 00000 | 100110 | xor $1,$2,$3 | $1=$2^S3 | (rd)←(rs)^(rt); rs=$2,rt=$3,rd=$1 |
| **nor** | 000000 | rs | rt | rd | 00000 | 100111 | nor $1,$2,$3 | $1= ~($2 | S3) | (rd)←~((rs) | (rt)); rs=$2,rt=$3,rd=$1 |
| **slt** | 000000 | rs | rt | rd | 00000 | 101010 | slt $1,$2,$3 | if($2<$3)  $1=1 else  $1=0 | if (rs< rt) rd=1 else rd=0;rs＝$2，rt=$3, rd=$1 |
| sltu | 000000 | rs | rt | rd | 00000 | 101011 | sltu $1,$2,$3 | if($2<$3)  $1=1 else  $1=0 | if (rs< rt) rd=1 else rd=0;rs＝$2，rt=$3, rd=$1, 无符号数 |
| sll | 000000 | 00000 | rt | rd | shamt | 000000 | sll $1,$2,10 | $1=$2<<10 | (rd)←(rt)<<shamt,rt=$2,rd=$1,shamt=10 |
| srl | 000000 | 00000 | rt | rd | shamt | 000010 | srl $1,$2,10 | $1=$2>>10 | (rd)←(rt)>>shamt, rt=$2, rd=$1, shamt=10, (逻辑右移) |
| **sra** | 000000 | 00000 | rt | rd | shamt | 000011 | sra $1,$2,10 | $1=$2>>10 | (rd)←(rt)>>shamt, rt=$2, rd=$1, shamt=10, (算术右移，注意符号位保留) |
| **sllv** | 000000 | rs | rt | rd | 00000 | 000100 | sllv $1,$2,$3 | $1=$2<<$3 | (rd)←(rt)<<(rs), rs=$3,rt=$2,rd=$1 |
| **srlv** | 000000 | rs | rt | rd | 00000 | 000110 | srlv $1,$2,$3 | $1=$2>>$3 | (rd)←(rt)>>(rs), rs=$3,rt=$2,rd=$1, (逻辑右移) |
| **srav** | 000000 | rs | rt | rd | 00000 | 000111 | srav $1,$2,$3 | $1=$2>>$3 | (rd)←(rt)>>(rs), rs=$3,rt=$2,rd=$1, (算术右移，注意符号位保留) |
| **jr** | 000000 | rs | 00000 | 00000 | 00000 | 001000 | jr $31 | goto $31 | (PC)←(rs) |
| I-类型 | op | rs | rt | immediate | | |  | | |
| **addi** | 001000 | rs | rt | immediate | | | addi $1,$2,10 | $1=$2+10 | (rt)←(rs)+(sign-extend)immediate,rt=$1,rs=$2 |
| **addiu** | 001001 | rs | rt | immediate | | | addiu $1,$2,10 | $1=$2+10 | (rt)←(rs)+(sign-extend)immediate,rt=$1,rs=$2 |
| **andi** | 001100 | rs | rt | immediate | | | andi $1,$2,10 | $1=$2&10 | (rt)←(rs)&(zero-extend)immediate,rt=$1,rs=$2 |
| ori | 001101 | rs | rt | immediate | | | ori $1,$2,10 | $1=$2|10 | (rt)←(rs)|(zero-extend)immediate,rt=$1,rs=$2 |
| **xori** | 001110 | rs | rt | immediate | | | xori $1,$2,10 | $1=$2^10 | (rt)←(rs)^(zero-extend)immediate,rt=$1,rs=$2 |
| **lui** | 001111 | 00000 | rt | immediate | | | lui $1,10 | $1=10\*65536 | (rt)←immediate<<16 & 0FFFF0000H，将16位立即数放到目的寄存器高16位，目的寄存器的低16位填0 |
| **lw** | 100011 | rs | rt | offset | | | lw $1,10($2) | $1=Memory[  $2+10] | (rt)←Memory[(rs)+(sign\_extend)offset],  rt=$1,rs=$2 |
| **sw** | 101011 | rs | rt | offset | | | sw $1,10($2) | Memory[  $2+10] =$1 | Memory[(rs)+(sign\_extend)offset]←(rt),  rt=$1,rs=$2 |
| **beq** | 000100 | rs | rt | offset | | | beq $1,$2,40 | if($1=$2)  goto PC+4+40 | if ((rt)=(rs)) then (PC)←(PC)+4+( (Sign-Extend) offset<<2), rs=$1, rt=$2 |
| **bne** | 000101 | rs | rt | offset | | | bne $1,$2,40 | if($1≠$2)  goto PC+4+40 | if ((rt)≠(rs)) then (PC)←(PC)+4+(  (Sign-Extend) offset<<2) , rs=$1, rt=$2 |
| slti | 001010 | rs | rt | immediate | | | slti $1,$2,10 | if($2<10)    $1=1 else    $1=0 | if ((rs)<(Sign-Extend)immediate) then (rt)←1; else (rt)←0, rs=$2, rt=$1 |
| sltiu | 001011 | rs | rt | immediate | | | sltiu $1,$2,10 | if($2<10)    $1=1 else    $1=0 | if ((rs)<(Zero-Extend)immediate) then (rt)←1; else (rt)←0, rs=$2, rt=$1 |
| J-类型 | op | address | | | | |  | | |
| j | 000010 | address | | | | | j 10000 | goto 10000 | (PC)←( (Zero-Extend) address<<2),  address=10000/4 |
| jal | 000011 | address | | | | | jal 10000 | $31=PC+4  goto 10000 | ($31)←(PC)+4;  (PC)←( (Zero-Extend) address<<2),  address=10000/4 |

图V 指令详细结构

以上即是MIPS指令说明。上面给出了本次实验的实验原理及设计电路图，下面将给出实验的步骤。

**2. 实验步骤**

**① 各模块设计**

1. **寄存器模块：**

寄存器组是指令操作的主要对象，MIPS 处理器里一共有 32 个 32 位的寄存器，故可以声明一个包含 32 个 32 位的寄存器数组。读寄存器时需要 Rs，Rd 的地址，得到其数据。写寄存器 Rd 时需要所写地址，所写数据，同时需要写使能。以上所有操作需要在时钟和复位信号控制下操作。故寄存器组设计如下：



**2. 控制器模块：**

根据指令中的指令码（op）和功能码（funct）的不同组合输出相应的控制信号。

|  |  |
| --- | --- |
|  | aluop 设计图  ALU 译码单元 |

图 控制器组成原理图

**3. ALU控制译码模块**

**ALU控制译码模块**

ALU 主要执行 5 种操作：与，或，加，减，小于设置(可以拓展)。这五种操作可以使用四位的编码表示：0000，0001，0010，01110，0111。指令不同，则对应的 ALU 运算不同，所以该模块需要根据指令来控制ALU 进行正确的运算。

Lw，sw，addi 指令均要求 ALU 执行加操作，则可分为一类， beq 指令要求 ALU 执行减操作，最后一类是 R 型指令，但不同的R 型指令对应不同的 ALU 运算，故需要再通过指令的功能码进一步确定 ALU 的运算。

最终该模块即实现 4 位ALUop操作码以及 6 位功能码输入，输出 4 位ALU 控制信号码ALUctr。（拓展lui指令和移位指令，增加输入lui和Asel输出,jr指令加一个控制信号jr）

**4. ALU运算器模块**



**5. 符号拓展模块**

**6. 指令存储器模块。**

该模块有4位地址输入和32位数据输出，首先将16种类型的16条指令写入存储单元中，然后根据4位地址输入选择相应的单元指令内容，将数据写入到输出变量中。地址输入数据、存储单元编号与指令的对应规则如下表所示：

该模块由 xilinx 提供的 IP 核进行设计，其中 ROM 模块由地址线、数据线和时钟信号线组成。  
（1） 数据位宽为 32 位， 数据深度为 128， 因此地址线位宽为 7 位。  
（2） 根据自己的汇编代码汇编成的机器码来制作 coe 文件，对 rom 进行初始化。  
（3） 注意：理论上读指令存储器是不需要时钟来触发的，它的地址一改变，输出的数据也立即改变。 但是， xilinx 提供的 ROM 模块无法去掉时钟的作用，一个解决办法是用一个比主时钟（更新 PC 的时钟） 快很多的时钟来驱动 ROM 模块， 使得它近似变得透明。

**7. 数据存储器（LW,SW）**

**8. 数码管显示模块**

七段译码显示的内容是16位的，而ALU的运算结果是32位的，将运算结果分为高十六位和低十六位，分别传进七段译码模块；在顶层模块可用一个开关，来选择是显示高16位还是低16位；

sm\_wei 选择哪一个数码管亮，sm\_duan 选择数码管的哪一段亮，sm\_wei 变换的速度是 1 秒 1000 次，使人眼看起来数码管是同时显示数值的。

**9. PC模块**

主模块

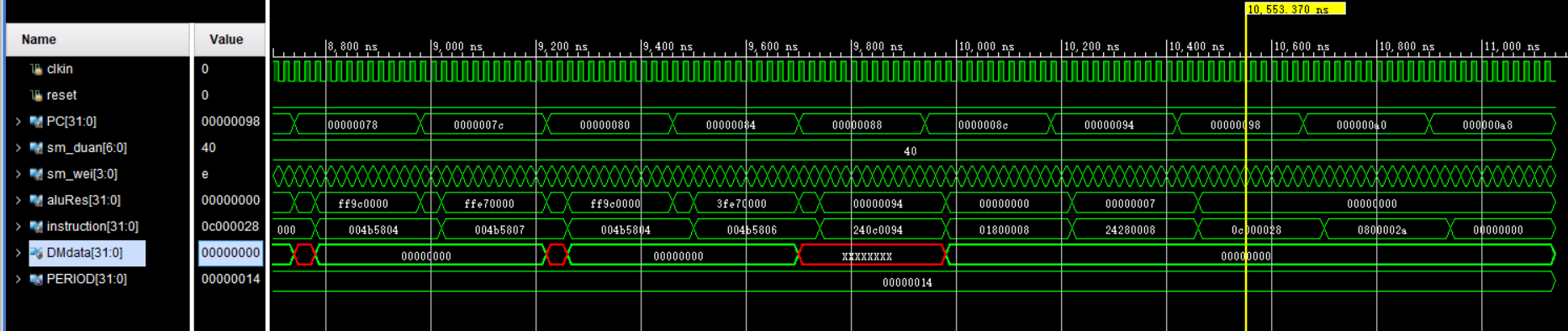
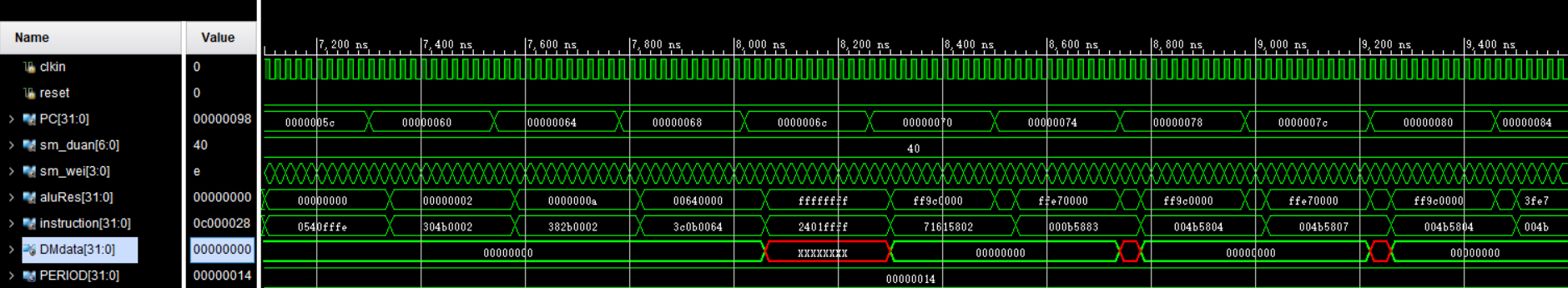
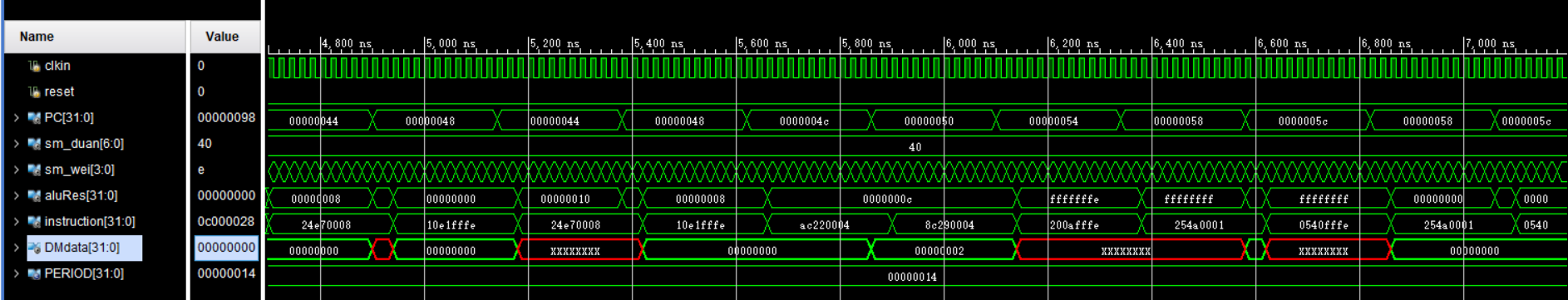
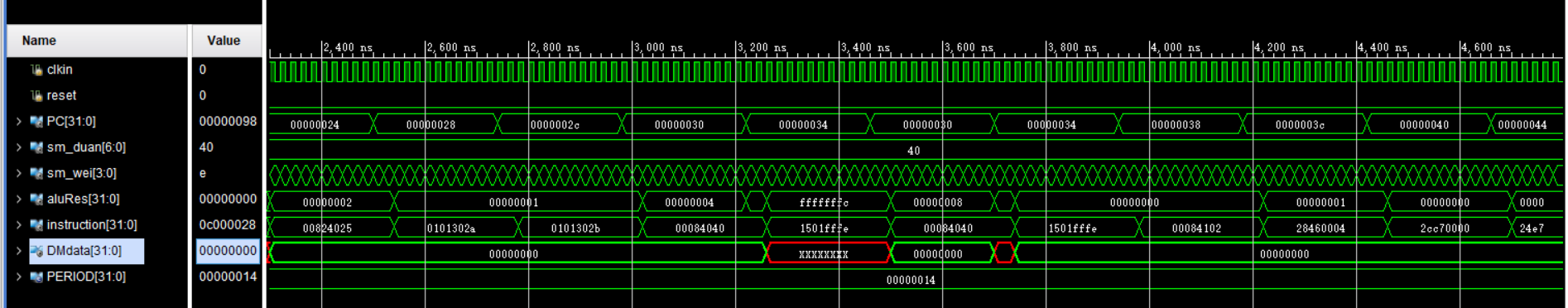
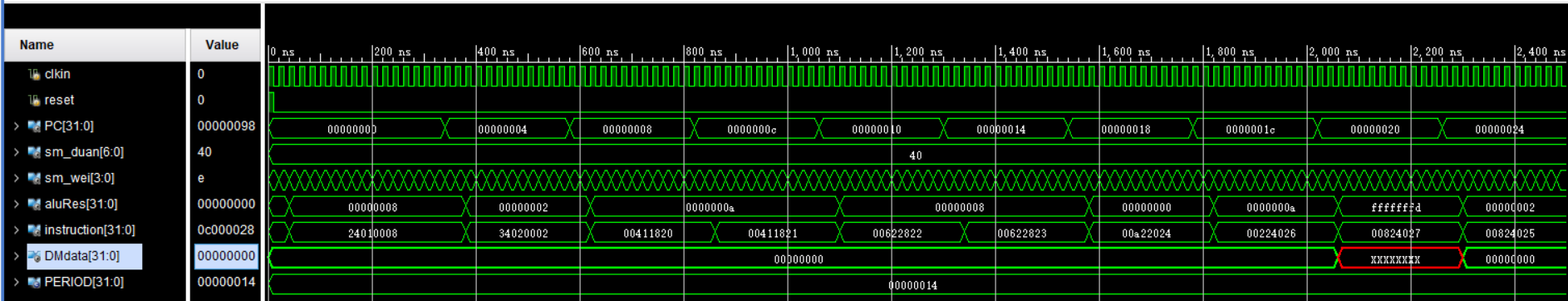
TOP文件，组装各个模块

**② 仿真实现**

**③ 烧板**

**四、实验结果**

仿真波形图：



**验收结果见附件视频**

**五、实验感想**

**这是我目前为止做的规模最大的计组实验，在编写的过程中也确实遇到了不少困难，花了很长时间才大概弄清楚各个模块的关系，找到进行的方向。代码块完成后的测试过程是最困难的，一开始的波形图总是不对。后来经过老师和同学的指点，才知道是时钟分频的问题。在完成这次实验的过程中，我查阅了许多资料，反复比对，修改，终于得到一个比较满意的结果。最后成功在Basys3板上看到自己实现的结果，还是很有成就感的。**

**附录（流程图，注释过的代码）：**

**测试程序段**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 地址 | 汇编程序 | 指令 | | | 运行结果 | | |
| op（6） | 16进制 | Next\_PC | | ALU运算结果 | DM输出结果（LW） |
| 0x00000000 | addiu $1,$0,8 | 001001 | 24010008 | 0x00000004 | | 00000008 |  |
| 0x00000004 | ori $2,$0,2 | 001101 | 34020002 | 0x00000008 | | 00000002 |  |
| 0x00000008 | add $3,$2,$1 | 000000 | 00411820 | 0x0000000C | | 0000000a |  |
| 0x0000000C | addu $3, $2, $1 | 000000 | 00411821 | 0x00000010 | | 0000000a |  |
| 0x00000010 | sub $5,$3,$2 | 000000 | 00622822 | 0x00000014 | | 00000008 |  |
| 0x00000014 | subu $5, $3, $2 | 000000 | 00622823 | 0x00000018 | | 00000008 |  |
| 0x00000018 | and $4,$5,$2 | 000000 | 00a22024 | 0x0000001C | | 00000000 |  |
| 0x0000001C | xor $8, $1, $2 | 000000 | 00224026 | 0x00000020 | | 0000000a |  |
| 0x00000020 | nor $8, $4, $2 | 000000 | 00824027 | 0x00000024 | | fffffffd |  |
| 0x00000024 | or $8,$4,$2 | 000000 | 00824025 | 0x00000028 | | 00000002 |  |
| 0x00000028 | slt $6, $8, $1 | 000000 | 0101302a | 0x0000002C | | 00000001 |  |
| 0x0000002C | sltu $6, $8, $1 | 000000 | 0101302b | 0x00000030 | | 00000001 |  |
| 0x00000030 | sll $8,$8,1 | 000000 | 00084040 | 0x00000034  0x00000034 | | 00000004  00000008 |  |
| 0x00000034 | bne $8,$1,-2 (≠,转30) | 000101 | 1501fffe | 0x00000030  0x00000038 | | fffffffc  00000000 |  |
| 0x00000038 | srl $8, $8, 4 | 000000 | 00084102 | 0x0000003C | | 00000000 |  |
| 0x0000003C | slti $6,$2,4 | 001010 | 28460004 | 0x00000040 | | 00000001 |  |
| 0x00000040 | sltiu $7,$6,0 | 001010 | 2cc70000 | 0x00000044 | | 00000000 |  |
| 0x00000044 | addiu $7,$7,8 | 001001 | 24e70008 | 0x00000048  0x00000048 | | 00000008  00000010 |  |
| 0x00000048 | beq $7,$1,-2 (=,转44) | 000100 | 10e1fffe | 0x000000440x0000004C | | 00000000  00000008 |  |
| 0x0000004C | sw $2,4($1) | 101011 | ac220004 | 0x00000050 | | 0000000c |  |
| 0x00000050 | lw $9,4($1) | 100011 | 8c290004 | 0x00000054 | | 0000000c | 00000002 |
| 0x00000054 | addi $10,$0,-2 | 001001 | 200afffe | 0x00000058 | | fffffffe |  |
| 0x00000058 | addiu $10,$10,1 | 001001 | 254a0001 | 0x0000005C  0x0000005C | | ffffffff  00000000 |  |
| 0x0000005C | bltz $10,-2(<0,转58) | 000001 | 0540fffe | 0x00000058  0x00000060 | | ffffffff  00000000 |  |
| 0x00000060 | andi $11,$2,2 | 001100 | 304b0002 | 0x00000064 | | 00000002 |  |
| 0x00000064 | xori $11, $1, 2 | 001110 | 382b0002 | 0x00000068 | | 0000000a |  |
| 0x00000068 | lui $11, 100 | 001111 | 3c0b0064 | 0x0000006C | | 00640000 |  |
| 0x0000006C | addiu $1, $0, -1 | 001001 | 2401ffff | 0x00000070 | | ffffffff |  |
| 0x00000070 | mul $11, $11, $1 | 011100 | 71615802 | 0x00000074 | | ff9c0000 |  |
| 0x00000074 | sra $11, $11, 2 | 000000 | 000b5883 | 0x00000078 | | ffe70000 |  |
| 0x00000078 | sllv $11, $11, $2 | 000000 | 004b5804 | 0x0000007C | | ff9c0000 |  |
| 0x0000007C | srav $11, $11, $2 | 000000 | 004b5807 | 0x00000080 | | ffe70000 |  |
| 0x00000080 | sllv $11, $11, $2 | 000000 | 004b5804 | 0x00000084 | | ff9c0000 |  |
| 0x00000084 | srlv $11, $11, $2 | 000000 | 004b5806 | 0x00000088 | | 3fe70000 |  |
| 0x00000088 | addiu $12, $0, 0x94 | 001001 | 240c0094 | 0x0000008C | | 00000094 |  |
| 0x0000008C | jr $12 | 000000 | 01800008 | 0x00000094 | | 00000000 |  |
| 0x00000090 | or $8,$4,$2 | 000000 | 00824025 | 0x00000094 | | 00000002 |  |
| 0x00000094 | addiu $8, $1, 8 | 000101 | 24280008 | 0x00000098 | | 00000007 |  |
| 0x00000098 | jal 0x000000a0 | 000011 | 0c000028 | 0x000000a0 | | 00000000 |  |
| 0x0000009C | or $8,$4,$2 | 000000 | 00824025 | 0x000000a0 | | 00000002 |  |
| 0x000000a0 | j 0x000000a8 | 000010 | 0800002a | 0x000000a8 | | 00000000 |  |
| 0x000000a4 | or $8,$4,$2 | 000000 | 00824025 | 0x000000a8 | | 00000002 |  |
| 0x000000a8 | halt | 111111 | FC000000 |  | |  |  |
| 0x000000aC |  |  |  |  | |  |  |