计算机组成原理实验报告

单周期处理器开发

班级：1619402班

学号：161940205 161940206 161940219 161940234

姓名：裴书玥 曹子钰 阴俊晖 牛纪龙

学院：计算机科学与技术学院/人工智能学院

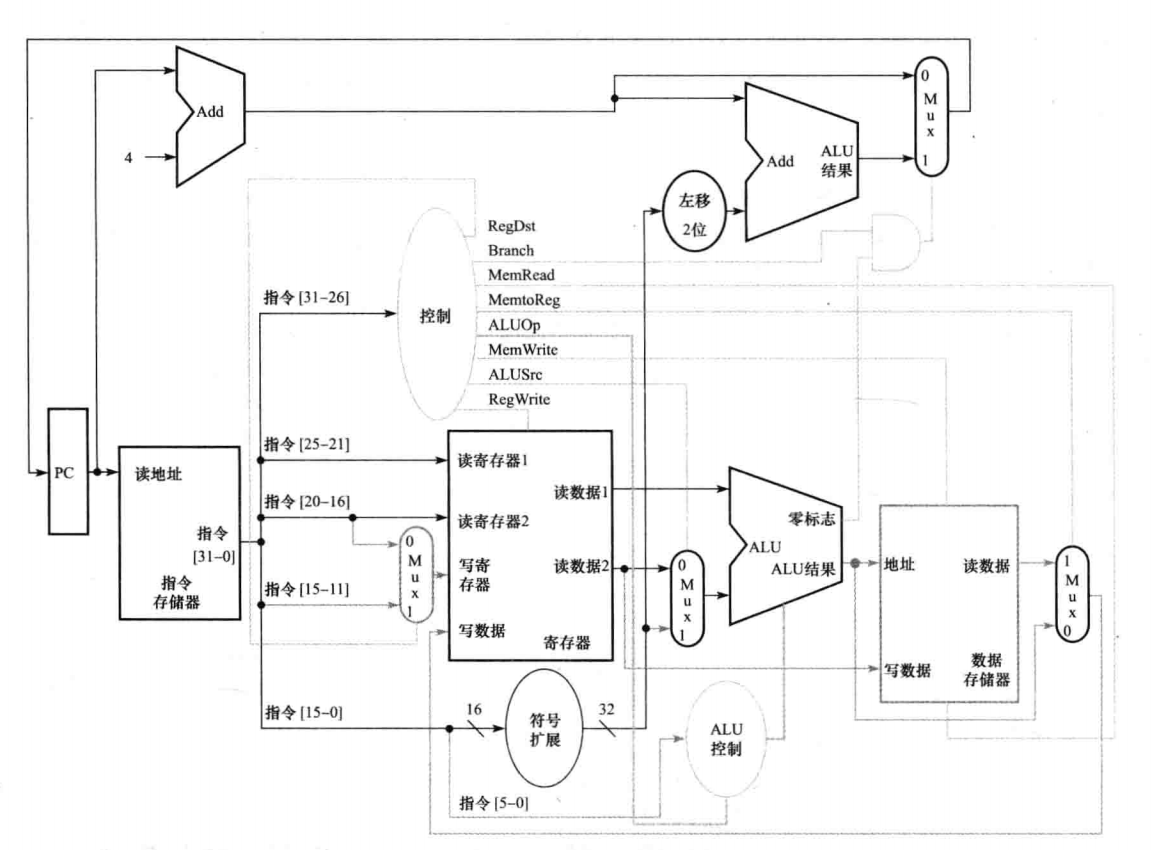
2021.05.28

**文档目录：**

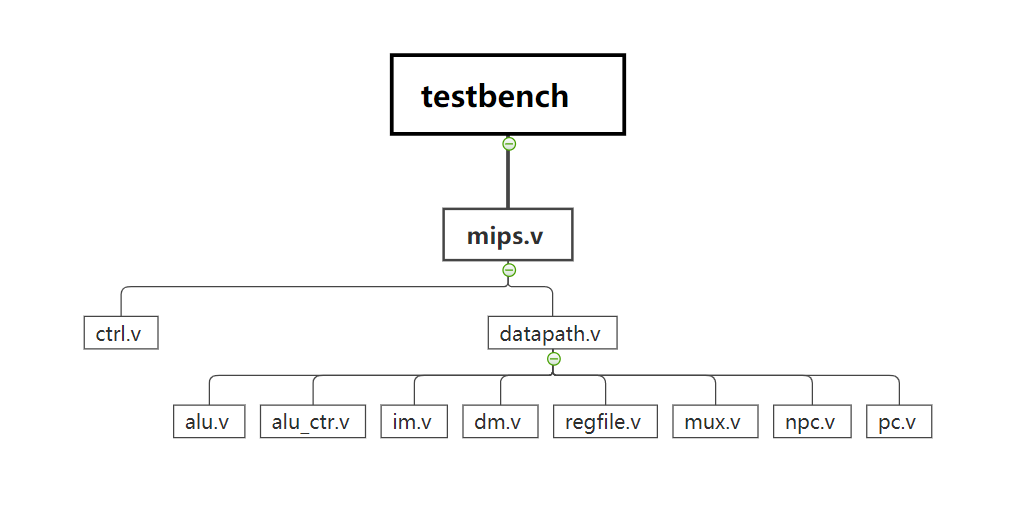
1. 功能设计说明
2. 模块化和层次化设计说明
3. 具体模块定义
4. 测试代码及结果
5. 实验完成时间安排
6. 心得体会

## 功能设计说明

1. 完成的指令集有add, addu, addi, addiu, sub, slt, and, or, xor, beq, j, sw, lw, lui等14条指令。
2. 处理器为单周期设计
3. 参考的指示图如下（教材 p180）此外还添加了j指令的实现，做出了一定的修改



## 模块化和层次设计说明



## 模块定义

1. **alu\_ctr模块**
2. 基本描述

简单的ALU控制单元，根据控制器输入的AluOp信号值和指令的funct字段

对运算器ALU发出AluCtrl控制信号来控制其操作

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| AluOp[1:0] | Input | 由控制器发出的运算器控制信号，只有两个位，能力有限 |
| funct[5:0] | Iuput | 指令的funct字段，用来判断非R指令的类型 |
| AluCtrl[3:0] | Output | 输出的控制信号，来决定运算器做何种运算 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 控制运算器 | 根据指令的funct字段和AluOp信号发出真正的ALU控制信号，相当于采用了一个多级控制器的结构，有利于优化 |

1. **alu模块**
2. 基本描述

处理器中的算术逻辑部件(Arithmetic Logic Unit)，通过该部件可以完成两个操作数

之间的一些基本运算，比如 {add, sub, and, or, slt(set less than), exclusive or}

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| a[31:0] | Input | 第一个操作数(RS寄存器中读取) |
| b[31:0] | Iuput | 第二个操作数(RT寄存器中读取或符号拓展后的立即数) |
| AluCtrl[3:0] | Input | 来自alu\_ctrl的控制信号，来决定运算器做何种运算  AND = 4'b0000, OR = 4'b0001, ADD = 4'b0010,  SUB = 4'b0110, SLT = 4'b0111, OR = 4'b0011, LUI = 4'b0101; |
| zero | Output | beq指令中判断相减后结果是否为0  1: result == 0  0: result != 0 |
| result[31:0] | Output | 运算结果 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 运算器 | 将两数根据对应的控制信号进行运算 |

1. **im模块**
2. 基本描述

处理器中指令存储器，程序初始化时将外部文件中的指令读入到im模块中

一条指令占位32 bits，共可存储2^10条指令，并根据输入的地址进行

指令的读取。

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| addr[11:0] | Input | PC的后12位，用来寻找指令 |
| dout[31:0] | Output | 根据地址输出[31:0] im[1023:0]中存储的指令  dout = im[addr[11:2]] |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 指令存储器 | 存储待运行的指令 |

1. **dm模块**
2. 基本描述

处理器中的数据存储器，用来存储指令运行过程中除寄存器以外的数据

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| addr[11:2] | Input | PC的后12位，用来寻找指令 |
| din[31:0] | Input | 待写入的数据 |
| MemWrite | Input | 控制器发出的控制信号，为1时可以写入  dm |
| Clk | Input | 时钟信号，在dm模块中位于时钟下降沿写入数据 |
| dout[31:0] | Output | 在[31:0] dm[1023:0] 中寻找对应地址的数据  dmout = dm[addr] |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读数据 | lw命令将数据从数据存储器中读出写入到寄存器中 |
| 2 | 写数据 | sw命令将数据从寄存器中独出写入到数据存储器中 |

1. **regfile模块**
2. 基本描述

寄存器文件，相当于32个32位的寄存器组

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| rs[4:0] | Input | Instruction[25:21](rs寄存器的地址) |
| rt[4:0] | Input | Instruction[20:16](rt寄存器的地址) |
| desReg[4:0] | Input | 目标寄存器的地址 |
| writeData  [31:0] | Input | 待写入数据，由多路选择器mux3给出 |
| rd1 | Output | rs寄存器中的值 |
| rd2 | Output | rt寄存器中的值 |
| RegWrite | Input | 控制器给出，1表示可以写数据 |
| clk | Input | 时钟信号 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读数据 | 读取指定寄存器中的数据 |
| 2 | 写数据 | 将外来数据写入到指定的目标寄存器中 |

1. **mux模块**
2. 基本描述

定义了3个多路选择器模块，分别用于判断

1. regFile的写数据的目标寄存器
2. 写入ALU部件的第二个操作数来源(rt寄存器或者立即数)
3. 写入regFile的数据来源(dm或者alu运算结果)
4. 模块接口

|  |  |  |  |
| --- | --- | --- | --- |
| 模块 | 信号名 | 方向 | 描述 |
| mux1 | rt[4:0] | Input | rt寄存器的地址 |
|  | rd[4:0] | Input | rd寄存器的地址 |
|  | RegDst | Input | 控制器发出，用于控制目标寄存器地址 |
|  | desReg[4:0] | Output | 目标寄存器的地址  desReg=(RegDst == 1) ? rt : rd |
| mux2 | rd2[31:0] | Input | 从rt寄存器中取出的数值 |
|  | Immediate\_number[31:0] | Input | 将指令的后16位做符号拓展之后的31位立即数 |
|  | AluSrc | Input | 控制器发出，控制alu的第二个操作数的来源 |
|  | b[31:0] | Output | alu部件的第二个操作数，  b = (AluSrc == 1) ? rd2 :  Immediate\_number |
| mux3 | dmout[31:0] | Input | 从数据存储器里面取出的数据 |
|  | aluResult  [31:0] | Input | alu部件的运算结果 |
|  | MemtoReg | Input | 控制器发出，控制写入目标寄存器数据的来源 |
|  | writeData | Output | 代写入的数据  writeData = (MemtoReg == 1) ?  Dmout : aluResult |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 目标寄存器选择器 | 选择待写入数据的目标寄存器 |
| 2 | 操作数来源选择器 | 选择alu的第二个操作数 |
| 3 | 写入数据选择器 | 选择写入目标寄存器的数据来源 |

1. **npc模块**
2. 基本描述

自带多路选择器的功能，用于计算下一条指令的地址

默认状态下为PC+4，当分支指令或者跳转指令的时候则需要额外考虑

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| PC[31:0] | Input | 当前的PC值 |
| instruAddr  [25:0] | Input | 当前指令的后26位，用于处理分支和跳转指令的 |
| branch | Input | 当前指令是否位分支指令 |
| zero | Input | 当前运算器的运算结果是否为0，用于判断是否执行分支指令 |
| jump | Input | 是否为跳转指令 |
| NPC[31:0] | Output | 计算得出的下一条指令的存储地址 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 计算下一条指令的地址 | 根据当前指令计算出下一条指令的存储地址 |

1. **pc模块**
2. 基本描述

Program Counter(程序计数器)，记录了当前运行的指令在im中的地址

并在时钟周期的上升沿根据npc信号进行更新，pc值初始为0x3000

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| npc[31:0] | Input | 由npc模块计算的下一条指令的地址 |
| clk | Input | 时钟信号 |
| rst | Input | 复位信号，为1时pc复位为0x3000 |
| pc[31:0] | Output | 当前指令的存储地址，在下一个时钟周期上升沿到来的时候进行更新 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 程序计数器 | 给出当前执行指令的地址 |

1. **ctrl模块**
2. 基本描述

整个单周期处理器的控制器模块，根据指令的op字段和funct字段

来为其他模块提供控制信号。

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| opcode[5:0] | Input | 指令的前6位操作码 |
| funct[5:0] | Input | 指令的后6位功能字段(R型指令) |
| RegDst | Output | 控制写入数据的目标寄存器 |
| Branch | Output | 是否为分支指令 |
| MemtoReg | Output | 控制写入寄存的数据来源 |
| Alusrc | Output | 控制进行运算的第二个操作数的数据来源 |
| AluOp[1:0] | Output | ALU控制器的顶层信号，用于控制ALU的运算方式 |
| MemWrite | Output | 能否写入数据到dm中 |
| RegWrite | Output | 能否写入数据到寄存器组中 |
| Jump | Output | a是否为分支指令 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 控制器 | 发出控制信号来控制各个处理器各个部件的功能 |

1. **datapath模块**
2. 基本描述

整个单周期处理器的数据通路，用来连接各个模块

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| RegDst | Input | 控制写入数据的目标寄存器 |
| Branch | Input | 是否为分支指令 |
| MemtoReg | Input | 控制写入寄存的数据来源 |
| Alusrc | Input | 控制进行运算的第二个操作数的数据来源 |
| AluOp[1:0] | Input | ALU控制器的顶层信号，用于控制ALU的运算方式 |
| MemWrite | Input | 能否写入数据到dm中 |
| RegWrite | Input | 能否写入数据到寄存器组中 |
| Jump | Input | a是否为分支指令 |
| clk | Input | 时钟周期 |
| rst | Input | 复位信号 |
| instr[31:0] | Output | 根据pc信号从im中取出的指令 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 数据通路 | 连接处理器的各个模块 |

1. **mips模块**
2. 基本描述

控制器模块和数据通路模块的组合，通过时钟信号和复位信号完成对

Mips指令的执行

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | Input | 时钟信号 |
| rst | Input | PC复位信号 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 执行指令 | 执行指令 |

## 测试代码及结果

1. asm代码(冒泡排序)

while:

addiu $t1,$zero,10

addiu $t2,$zero,0x0

addiu $t3,$zero,10

addiu $t4,$zero,1

addiu $t7,$zero,4

addiu $v0,$zero,0

for:

slt $v1,$v0,$t3

beq $v1,0,after1

addi $v0,$v0,1

sw $t1,($t2)

sub $t1,$t1,$t4

addi $t2,$t2,4

j for

after1:

addiu $t1,$zero,10

addiu $v0,$zero,0

addiu $a0,$zero,0

sub $t3,$t3,$t4

bubble1:

addiu $t2,$zero,0x0

slt $v1,$v0,$t3

beq $v1,0,after2

addi $v0,$v0,1

addiu $a0,$zero,0

bubble2:

slt $v1,$a0,$t3

beq $v1,0,bubble1

addi $a0,$a0,1

lw $t1,($t2)

lw $t5,4($t2)

addi $t2,$t2,4

slt $v1,$t1,$t5

beq $v1,1,bubble2

sub $t2,$t2,$t7

or $t6,$t1,$zero

and $t1,$t1,$zero

or $t1,$t1,$t5

and $t5,$t5,$zero

or $t5,$t5,$t6

sw $t1,($t2)

sw $t5,4($t2)

add $t2,$t2,$t7

j bubble2

after2:

addiu $v0,$zero,0

addiu $t2,$zero,0x0

result:

slt $v1,$v0,$s3

beq $v1,0,afterall

addi $v0,$v0,4

lw $t1 ($t2)

addi $t2,$t2,4

j result

afterall:

lui $t1,0x0100

xor $t1,$t1,$t1

addiu $t1,$zero,10

j end

end:

j end

1. 生成的16进制文件(code.txt)

2409000a

240a0000

240b000a

240c0001

240f0004

24020000

004b182a

20010000

10230005

20420001

ad490000

012c4822

214a0004

08000c06

2409000a

24020000

24040000

016c5822

240a0000

004b182a

20010000

10230016

20420001

24040000

008b182a

20010000

1023fff7

20840001

8d490000

8d4d0004

214a0004

012d182a

20010001

1023fff6

014f5022

01207025

01204824

012d4825

01a06824

01ae6825

ad490000

ad4d0004

014f5020

08000c18

24020000

240a0000

0053182a

20010000

10230004

20420004

8d490000

214a0004

08000c2e

3c090100

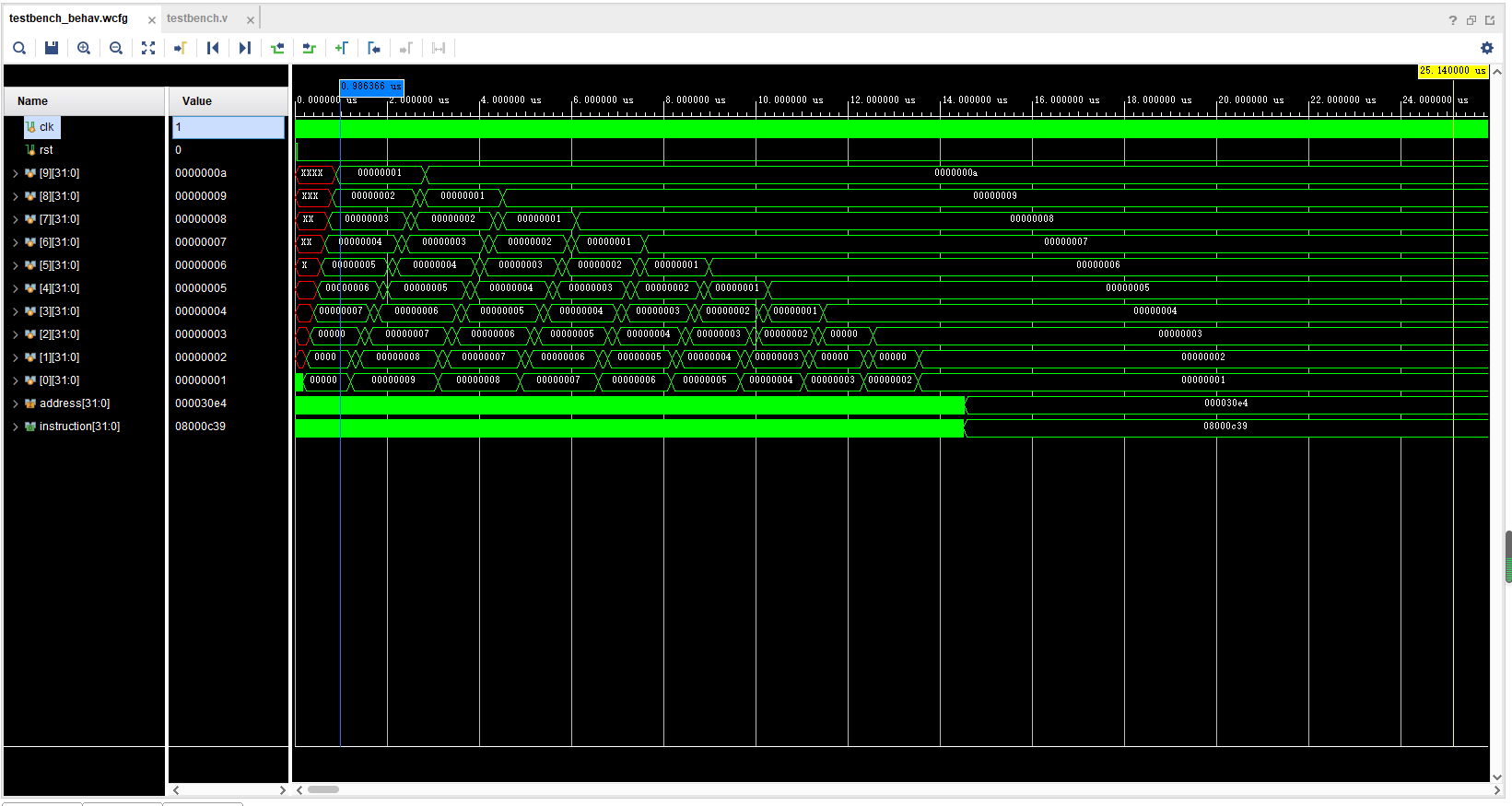
01294826

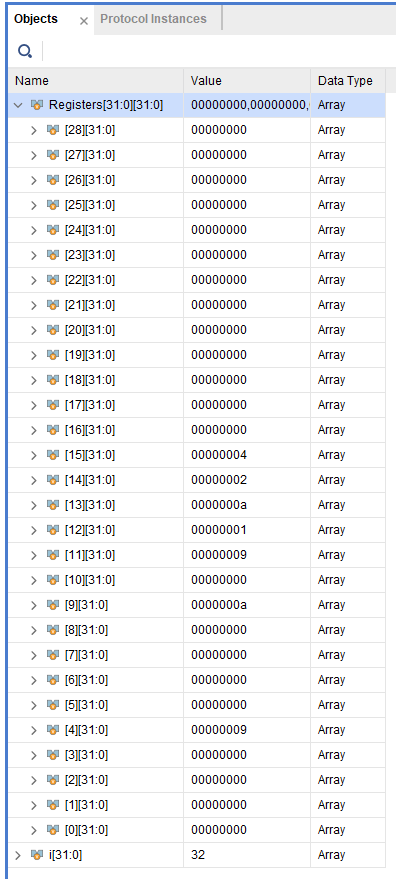
2409000a

08000c39

08000c39

1. 测试结果如下(regFile和dm在运行结束后的数据值)





## 实验时间安排

1.  配置vivado，了解vivado的使用方法，学习verilog语法。——2小时  
2.  了解实验需求，复习教材相关内容，学习各个模块的连接，自己尝试写出对应的几个小模块。——6小时  
3.  将各个小模块在数据通路中整合起来，修改完善接口，实现完整的数据通路。——4小时  
4.  完成控制器内容，将各个部件的控制信号组合在一起，达到控制所有部件的运行的目的。——3小时  
5.  使用vivado和VS code编写代码，利用助教给出的代码进行测试。——4小时  
6.  调试代码并不断完善程序。——1小时  
7.  总时长：20小时

1. **心得体会**

本实验让我们学会了如何使用 vivado 做简单的单周期处理器的开发。本实验的难点之一在于了解原理以及整体构造，熟悉各模块的功能，了解各个模块是如何连接的以及如何使用控制信号完成模块的使用。只有原理清晰之后，才能够着手写代码、对代码有一个整体的认知，在后期调试过程中才能够有清晰的分析。通过实验明白了只是通过书本学到的东西是不够的，只有通过自己动手，发现并解决问题，才能对知识点有更深入的认知，也会有自己的理解。  
经过本次单周期处理器的实验，我们收获到了很多，接触到了一些新东西：学会了新工具vivado的使用、开始慢慢掌握了新的语言verilog的语法，对单周期CPU有了更深入的理解。