

2021 年春季学期  
计算机组成原理课程

32 位 RISC-V 单周期 CPU 核

设  
计  
报  
告

负责人： 梁泽成 联系方式：17326369106

李骏祥 赵琪润 张芝林 杜晓涛

班级：2019 级计算机二班 第七组 2021 年 5 月 20 日

# 一、项目简述

## 1. 项目环境与级别

设计语言：Verilog 硬件描述语言

仿真环境：Vivado 2020.1版本

## 2. 任务目标

课程设计要求：设计一个兼容32位MIPS或RISC-V指令集的CPU核

指令集： RISC-V指令集（RV32IM）

设计目标：

- 1、使用哈佛结构的单周期CPU
- 2、支持RV32I基础整数指令集中除“状态与控制”类指令外的所有指令
- 3、支持RV32M 乘除法扩展指令集
- 4、能够运行简单的小程序，且充分利用RV32IM指令集。

## 3. 参考资料

- 1、2018 RISC-V 手册 一本开源指令集的指南 DAVID PATTERSON, ANDREW WATERMAN 翻译:勾凌睿,黄成,刘志刚
- 2、康振邦-RISCV指令集分类（译码版）\_v4\_2020.1.25
- 3、所有课程PPT（指导老师：何安平）

## 4. 项目文件夹结构

第七组.zip

|                   |                     |
|-------------------|---------------------|
| ——project.zip     | #vivado项目文件夹打包      |
| ——PPT.pptx        |                     |
| ——报告.pdf          |                     |
| ——<src>           | #源代码文件夹             |
| ——<verilog>       | #cpu设计文件源码文件夹       |
| ——<tb>            | #TESTBENCH文件源码文件夹   |
| ——id_tb.v         | #译码器测试文件            |
| ——TESTBENCH.v     | #cpu整体测试文件          |
| ——<test>          | #cpu整体测试小程序源码文件夹    |
| ——<PrimeNumber>   | #求素数小程序源码、汇编、机器码文件夹 |
| ——<InverseModulo> | #求逆元小程序源码、汇编、机器码文件夹 |

## 二、实现细节

### 1、指令集选取

RISC-V指令集可以分为六种类型（R I S B U J）：

|            |           |    |    |         |    |            |        |    |          |         |   |        |   |        |  |
|------------|-----------|----|----|---------|----|------------|--------|----|----------|---------|---|--------|---|--------|--|
| 31         | 30        | 25 | 24 | 21      | 20 | 19         | 15     | 14 | 12       | 11      | 8 | 7      | 6 | 0      |  |
| funct7     |           |    |    | rs2     |    | rs1        | funct3 |    | rd       |         |   | opcode |   | R-type |  |
| imm[11:0]  |           |    |    |         |    | rs1        | funct3 |    | rd       |         |   | opcode |   | I-type |  |
| imm[11:5]  |           |    |    | rs2     |    | rs1        | funct3 |    | imm[4:0] |         |   | opcode |   | S-type |  |
| imm[12]    | imm[10:5] |    |    | rs2     |    | rs1        | funct3 |    | imm[4:1] | imm[11] |   | opcode |   | B-type |  |
| imm[31:12] |           |    |    |         |    |            |        |    | rd       |         |   | opcode |   | U-type |  |
| imm[20]    | imm[10:1] |    |    | imm[11] |    | imm[19:12] |        |    | rd       |         |   | opcode |   | J-type |  |

本设计选取基本指令集RV32I的除“状态与控制”外的所有指令（37条）：

| 31                    | 25 24 | 20 19 | 15 14 | 12 11       | 7 6     | 0       |         |
|-----------------------|-------|-------|-------|-------------|---------|---------|---------|
| imm[31:12]            |       |       |       | rd          |         | 0110111 | U lui   |
| imm[31:12]            |       |       |       | rd          |         | 0010111 | U auipc |
| imm[20 10:1 11 19:12] |       |       |       | rd          |         | 1101111 | J jal   |
| imm[11:0]             |       | rs1   | 000   | rd          |         | 1100111 | I jalr  |
| imm[12 10:5]          | rs2   | rs1   | 000   | imm[4:1 11] | 1100011 |         | B beq   |
| imm[12 10:5]          | rs2   | rs1   | 001   | imm[4:1 11] | 1100011 |         | B bne   |
| imm[12 10:5]          | rs2   | rs1   | 100   | imm[4:1 11] | 1100011 |         | B blt   |
| imm[12 10:5]          | rs2   | rs1   | 101   | imm[4:1 11] | 1100011 |         | B bge   |
| imm[12 10:5]          | rs2   | rs1   | 110   | imm[4:1 11] | 1100011 |         | B bltu  |
| imm[12 10:5]          | rs2   | rs1   | 111   | imm[4:1 11] | 1100011 |         | B bgeu  |
| imm[11:0]             |       | rs1   | 000   | rd          |         | 0000011 | I lb    |
| imm[11:0]             |       | rs1   | 001   | rd          |         | 0000011 | I lh    |
| imm[11:0]             |       | rs1   | 010   | rd          |         | 0000011 | I lw    |
| imm[11:0]             |       | rs1   | 100   | rd          |         | 0000011 | I lbu   |
| imm[11:0]             |       | rs1   | 101   | rd          |         | 0000011 | I lhu   |
| imm[11:5]             | rs2   | rs1   | 000   | imm[4:0]    | 0100011 |         | S sb    |
| imm[11:5]             | rs2   | rs1   | 001   | imm[4:0]    | 0100011 |         | S sh    |
| imm[11:5]             | rs2   | rs1   | 010   | imm[4:0]    | 0100011 |         | S sw    |
| imm[11:0]             |       | rs1   | 000   | rd          |         | 0010011 | I addi  |
| imm[11:0]             |       | rs1   | 010   | rd          |         | 0010011 | I slti  |
| imm[11:0]             |       | rs1   | 011   | rd          |         | 0010011 | I sltiu |
| imm[11:0]             |       | rs1   | 100   | rd          |         | 0010011 | I xori  |
| imm[11:0]             |       | rs1   | 110   | rd          |         | 0010011 | I ori   |
| imm[11:0]             |       | rs1   | 111   | rd          |         | 0010011 | I andi  |
| 0000000               | shamt | rs1   | 001   | rd          |         | 0010011 | I slli  |
| 0000000               | shamt | rs1   | 101   | rd          |         | 0010011 | I srli  |
| 0100000               | shamt | rs1   | 101   | rd          |         | 0010011 | I srai  |
| 0000000               | rs2   | rs1   | 000   | rd          |         | 0110011 | R add   |
| 0100000               | rs2   | rs1   | 000   | rd          |         | 0110011 | R sub   |
| 0000000               | rs2   | rs1   | 001   | rd          |         | 0110011 | R sll   |
| 0000000               | rs2   | rs1   | 010   | rd          |         | 0110011 | R slt   |
| 0000000               | rs2   | rs1   | 011   | rd          |         | 0110011 | R sltu  |
| 0000000               | rs2   | rs1   | 100   | rd          |         | 0110011 | R xor   |
| 0000000               | rs2   | rs1   | 101   | rd          |         | 0110011 | R srl   |
| 0100000               | rs2   | rs1   | 101   | rd          |         | 0110011 | R sra   |
| 0000000               | rs2   | rs1   | 110   | rd          |         | 0110011 | R or    |
| 0000000               | rs2   | rs1   | 111   | rd          |         | 0110011 | R and   |

选取扩展指令集RV32M的所有指令（8条）：

| 31      | 25 24 | 20 19 | 15 14 12 11 | 7 6 | 0       |          |
|---------|-------|-------|-------------|-----|---------|----------|
| 0000001 | rs2   | rs1   | 000         | rd  | 0110011 | R mul    |
| 0000001 | rs2   | rs1   | 001         | rd  | 0110011 | R mulh   |
| 0000001 | rs2   | rs1   | 010         | rd  | 0110011 | R mulhsu |
| 0000001 | rs2   | rs1   | 011         | rd  | 0110011 | R mulhu  |
| 0000001 | rs2   | rs1   | 100         | rd  | 0110011 | R div    |
| 0000001 | rs2   | rs1   | 101         | rd  | 0110011 | R divu   |
| 0000001 | rs2   | rs1   | 110         | rd  | 0110011 | R rem    |
| 0000001 | rs2   | rs1   | 111         | rd  | 0110011 | R remu   |

R I S B类型相关指令的具体含义可以参考RISC-V手册，此处不再赘述

对于U J类型的指令，需要特别关注，内容如下：

#### JAL, JALR 指令

(1) 指令汇编格式

```
jal    rd, label
jalr   rd, rs1, imm
```

(2) 指令详解

该组指令为无条件跳转指令，即一定会发生跳转：

- jal 指令使用 20 位立即数（有符号数）作为偏移量（offset）。该偏移量乘以 2，然后与该指令的 PC 相加，生成得到最终的跳转目标地址，因此仅可以跳转到前后 1MB 的地址区间。jal 指令将其下一条指令的 PC（即当前指令 PC+4）的值写入其结果寄存器 rd 中。  
注意：在实际的汇编程序编写中，跳转的目标往往使用汇编程序中的 label，汇编器会自动根据 label 所在的地址计算出相对的偏移量赋予指令编码。
- jalr 指令使用 12 位立即数（有符号数）作为偏移量，与操作数寄存器 rs1 中的值相加得到最终的跳转目标地址。jalr 指令将其下一条指令的 PC（即当前指令 PC+4）的值写入其结果寄存器 rd。

#### LUI, AUIPC 指令

(1) 指令汇编格式

```
lui    rd, imm
auipc  rd, imm
```

(2) 指令详解

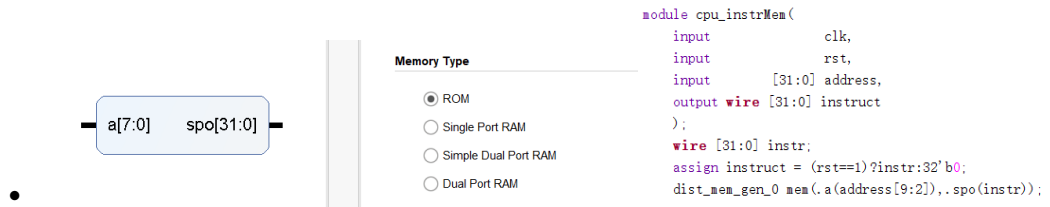
- lui 指令将 20 位立即数的值左移 12 位（低 12 位补 0）成为一个 32 位数，将此数写回寄存器 rd 中。
- auipc 指令将 20 位立即数的值左移 12 位（低 12 位补 0）成为一个 32 位数，将此数与该指令的 PC 值相加，将加法结果写回寄存器 rd 中。

对于U J类型的指令，除jalr外其所用到的PC为本条指令的PC值，应此在设计程序计数器时，应当考虑在内。对于jalr指令所用的PC+4，可以通过ALU计算得出。

## 2、模块设计

### (1) 存储器类设计

- 指令存储器（暂定1MB）采用ip核生成的方式（Distriuted Memory）
  - Ip核的数据位宽设置32位，coe文件详见测试文件夹，ip核不使用任何输入输出寄存器和流水线



- 数据存储器（32MB）为了便于修改数据，采用Verilog代码的方式生成：

```

always @(posedge clk or negedge rst) begin //写入存储器
    if(~rst) begin
        //求素数程序测试时使用
        mem[0]<=32'd25; //取前25个素数
        /* //求乘法逆元程序测试时使用
        mem[0]<=32'd17; // a = 17
        mem[1]<=32'd59; // p = 59
        // 17x=1(mod59) -> x = 7
        */
    end
    else if(((address != 32'b0) && (EnableWrite == 1'b1))) begin
        mem[addr] <= dataW;
    end
end

module cpu_dataMem(
    input rst,
    input clk,
    input EnableWrite,
    input [31:0] address,
    input [31:0] dataW,
    output reg [31:0] dataR
);
    reg [31:0] mem[0:8191];
    wire [12:0] addr;
    assign addr = address[14:2];

```

## (2) 寄存器组设计

- 根据RISC-V CPU要求，定义32个32位寄存器，其中0号寄存器为只读寄存器，且值恒为0

```

integer i; //循环用
reg [31:0] regF[0:31]; //32个寄存器

always @(posedge clk or negedge rst) begin //写
    if(~rst)begin
        for(i=0;i<32;i=i+1) //寄存器上电归0
            regF[i] <= 32'b0;
        regF[2] <= 32'd2048; // riscv定义x2寄存器为堆栈指针寄存器(sp)，上电置为2048
    end
    else if(rst && enableWrite && RD != 5'b0) regF[RD]<=busD;
end

always @(*) begin//读
    if(~rst)begin
        busA<=32'b0;
        busB<=32'b0;
    end else begin
        if (RA == 5'b0 )busA<=32'b0;
        else busA<=regF[RA];
        if (RB == 5'b0 ) busB<=32'b0;
        else busB<=regF[RB];
    end
end

```

- 对于寄存器组，在复位时初始化所有值置为0，但是对于对栈寄存器sp，其值暂时置为2048，以开拓堆栈空间，运行程序。

## (3) 选择器设计

- 程序部分选择器除选择功能外，还可根据不同的信号，将输入扩展不同的输出。
- 如访存指令：寄存器的选择器，可以选择不同的输入，也可扩展（或压缩）存储器的输出数据。（`lb lbu lh lhu lu`指令适用）

- ```

assign out = (op_MUXtoreg_alu_mem == `MUXtoReg_ALU )? ALU_result :
              (op_MUXtoreg_alu_mem == `MUXtoReg_MEM_F )? mem_data :
              (op_MUXtoreg_alu_mem == `MUXtoReg_MEM_HS)? {{24{mem_data[7]}}, mem_data[7:0]} :
              (op_MUXtoreg_alu_mem == `MUXtoReg_MEM_S )? {{16{mem_data[15]}}, mem_data[15:0]} :
              (op_MUXtoreg_alu_mem == `MUXtoReg_MEM_HU)? {24'b0, mem_data[7:0]} :
              (op_MUXtoreg_alu_mem == `MUXtoReg_MEM_U )? {16'b0, mem_data[15:0]} : {32'b0};

```
- 对于写回的三条指令，则根据不同的信号进行符号位扩展，以写入正确的内容
- ```

assign out = (op_MUXtodataMem_rs2 == `MUX_sw ) ? rs2 :
              (op_MUXtodataMem_rs2 == `MUX_sb ) ? {{24{rs2[7]}}, rs2[7:0]} :
              (op_MUXtodataMem_rs2 == `MUX_sh ) ? {{16{rs2[15]}}, rs2[15:0]} : {32'b0} ;

```
- Jarl指令，rs2选择器直接输出十六进制4，以实现PC+4
  - 关于各选择器的控制信号含义，代码已有详尽注释，此处略

#### (4) ALU设计

- ALU为组合逻辑，在上一次课程设计内容上增减
  - ALU使用控制码区分不同的运算（switch-case结构）
  - RV32I指令集运算使用上一次课程设计内容
  - RV32M指令集的乘除取余运算使用自带\*/%运算符
  - 其余添加了几个特殊的运算：
    - 取rs2输入：不做运算，输出rs2
  - 对于运算结果的标志位信号，仅保留0标志，对于溢出进位等内容，RISC-V处理器未作特殊要求，不做保留

#### (5) 程序计数器设计

- PC采用时序逻辑结构，若复位信号使能，则输出0地址
- 若复位后，当前PC（curPC）在clk上升沿更新为“下一个PC”
- 相关控制信号如下

| 控制信号                  | PC值             | 备注          |
|-----------------------|-----------------|-------------|
| BRANCH_NOCONDITION    | CurPC=NowPC+imm | 无条件跳转       |
| BRANCH_ZERO && zero   | CurPC=NowPC+imm | ALU结果为0则跳转  |
| BRANCH_NZERO && ~zero | CurPC=NowPC+imm | ALU结果为非0则跳转 |
| 其它情况                  | CurPC=NowPC+4   | PC+4        |

## (6) 译码器设计

- 译码器采用行为级描述的方式编写代码

- 译码器的主要功能:

- 根据所给输入指令:

- 划分op码、func3、funct7码
- 给出RA RB RD的寄存器地址
- 给出所需立即数
- 根据指令给出各控制信号的值:
  - ALU控制信号
  - 分支控制信号
  - ALU输入选择信号 (rs1和pc)
  - ALU输入选择信号 (rs2和立即数)
  - PC选择信号 (rs2和立即数)
  - 存储器输入扩展信号 (rs2扩展)
  - 寄存器组选择信号 (alu结果和存储器输出)

```
module cpu_id(  
    input wire rst,  
    input wire [31:0] instr,  
    output wire [4:0] RA,  
    output wire [4:0] RB,  
    output wire [4:0] RD, //writeAddress  
    output reg [31:0] imm, // 立即数  
  
    output reg regWriteEnable, // Reg写使能  
    output reg memWriteEnable, // Mem写使能  
  
    output reg [1:0] op_branch,  
    output reg [5:0] op_alu,  
    output reg op_MUXtoALU_rs1_pc,  
    output reg op_MUXtoPC_rs1_pc,  
    output reg [1:0] op_MUXtoALU_rs2_imm,  
    output reg [1:0] op_MUXtodataMem_rs2,  
    output reg [2:0] op_MUXtoreg_alu_mem  
);
```

| 译码器输入控制信号          | 控制信号命名              | R       | I<br>Imm/load | S       | B       | U<br>lui/auipc | J<br>Jal/jalr   |
|--------------------|---------------------|---------|---------------|---------|---------|----------------|-----------------|
| 存储器写使能             | MemWriteEnable      | Disable | Disable       | Enable  | DISABLE | DISABLE        | DISABLE         |
| 寄存器写使能             | regWriteEnable      | Enable  | Enable        | Disable | DISABLE | Enable         | Enable          |
| 分支控制信号             | op_branch           | DEFAULT | DEFAULT       | DEFAULT | ##      | DEFAULT        | NOCONDITION     |
| ALU操作码             | op_alu              | ##      | ##/<br>ADD    | ADD     | ##      | ADD/<br>RS2    | ADD4            |
| ALU输入选择信号 (rs1和pc) | op_MUXtoALU_rs1_pc  | Rs1     | Rs1           | Rs1     | Rs1     | DEFAULT        | Pc              |
| ALU输入选择信号(rs2和imm) | op_MUXtoALU_rs2_imm | Rs2     | Imm           | Imm     | Rs2     | imm            | DEFAULT         |
| PC选择信号 (rs1和pc)    | op_MUXtoPC_rs1_pc   | pc      | pc            | pc      | Pc      | pc             | DEFAULT<br>/Rs1 |
| 存储器输入扩展信号 (rs2)    | op_MUXtodataMem_rs2 | DEFAULT | DEFAULT       | ##      | ##      | DEFAULT        | DEFAULT         |
| 寄存器组写入选择信号         | op_MUXtoreg_alu_mem | alu     | Alu/<br>Mem   | DEFAULT | DEFAULT | alu            | alu             |

DEFAULT 表示默认或者与此项无关

##表示根据具体指令功能决定

### (7) 其他文件说明

- 宏定义文件 `define.v` 定义了Verilog代码中RISC-V CPU中常见的常量内容，以确保不因输入错误产生bug，同时使得程序更易阅读。
- 节选部分如下图：

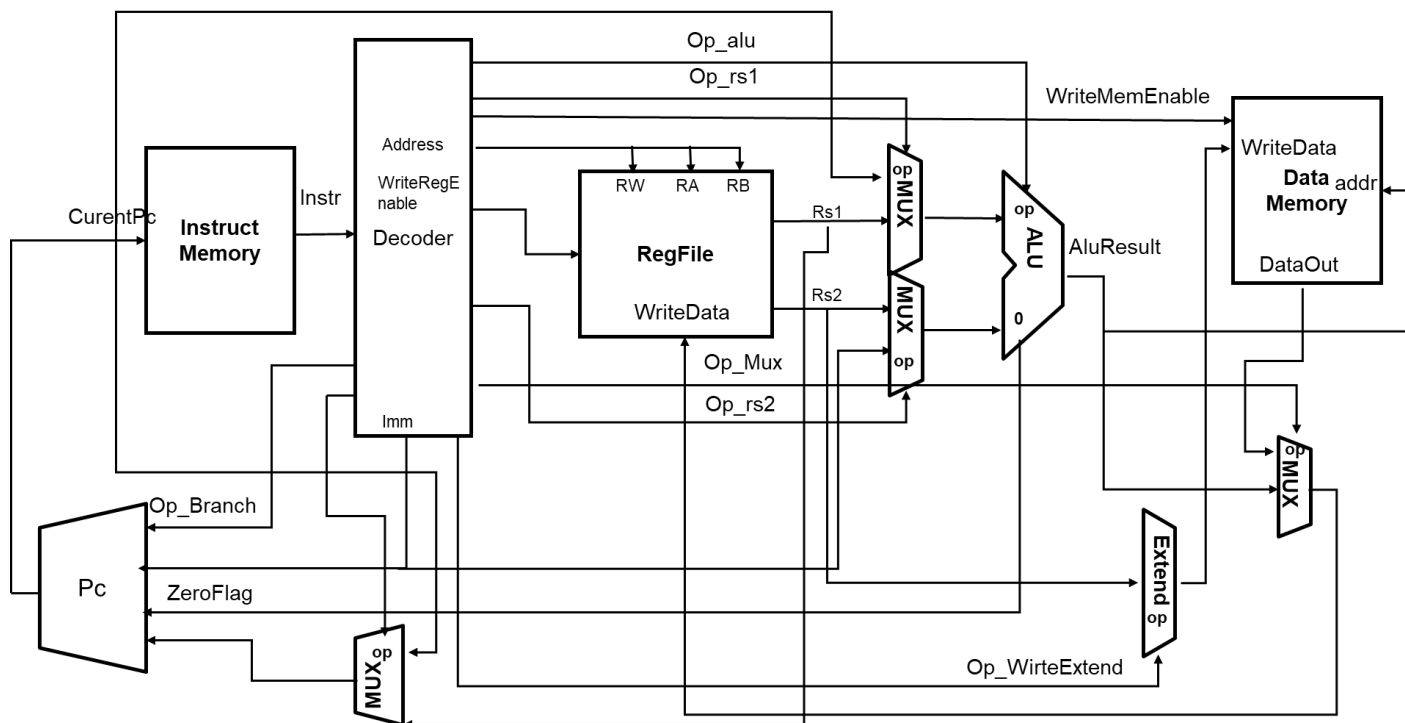
```
`define ENABLE 1'b1
`define DISABLE 1'b0
//指令op:
`define inst_op_Rtype      7'b0110011 //R型
`define inst_op_Itype_imm  7'b0010011 //I型 立即数计算
`define inst_op_Itype_load  7'b0000011 //I型 立即数加载
`define inst_op_Stype      7'b0100011 //S型
`define inst_op_Btype      7'b1100011 //B型
`define inst_op_Utype_lui   7'b0110111 //U型 lui
`define inst_op_Utype_aupic 7'b0010111 //U型 aupic
`define inst_op_Jtype_jal   7'b1101111 //J型 jal
`define inst_op_Jtype_jalr  7'b1100111 //J型 jalr

//Rtype
`define Rtype_funct7_common 7'b0000000 //R型常用运算
`define Rtype_funct7_subra  7'b0100000 //R型常用运算
`define Rtype_funct7_muldiv 7'b0000001 //R型乘除运算

`define Rcommon_funct3_add  3'b000 //add
`define Rcommon_funct3_and  3'b111 //and
`define Rcommon_funct3_or   3'b110 //or
`define Rcommon_funct3_xor  3'b100 //xor
`define Rcommon_funct3_srl  3'b101 //srl
`define Rcommon_funct3_sll  3'b001 //sll
`define Rcommon_funct3_slt  3'b010 //slt
`define Rcommon_funct3_sltu 3'b011 //sltu

`define Rsubra_funct3_sra  3'b101 //sra
`define Rsubra_funct3_sub  3'b000 //sub
```

### 3、数据通路

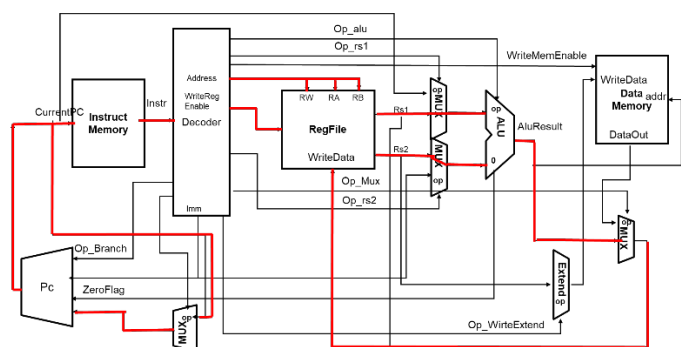


图中所有地址和数据线路均有标注，少数未标注的为控制信号。

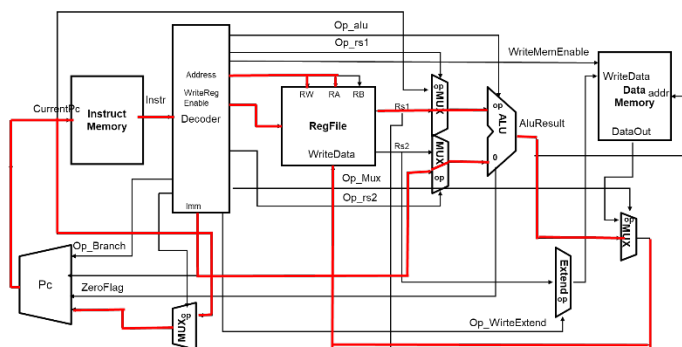
上图省略clk与rst



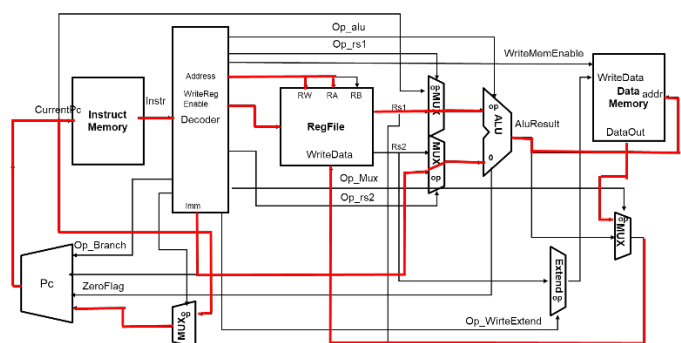
不同类型指令的数据通路如下:



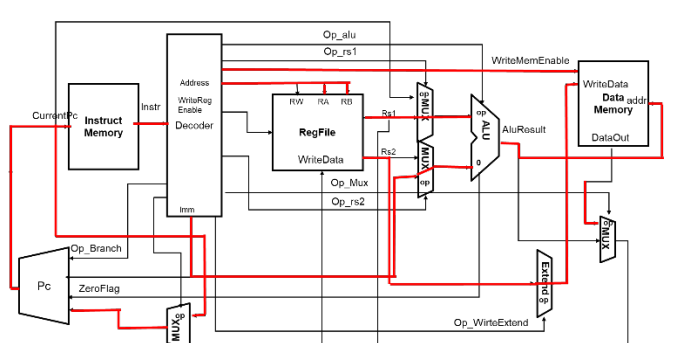
R型



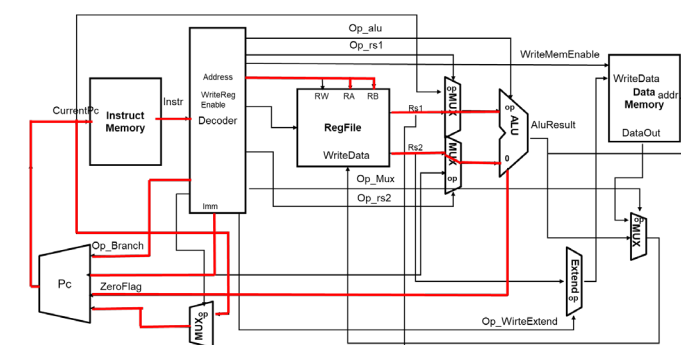
I (imm) 型



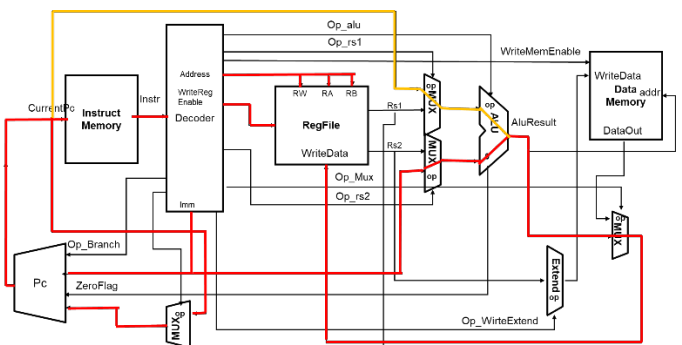
I (load) 型



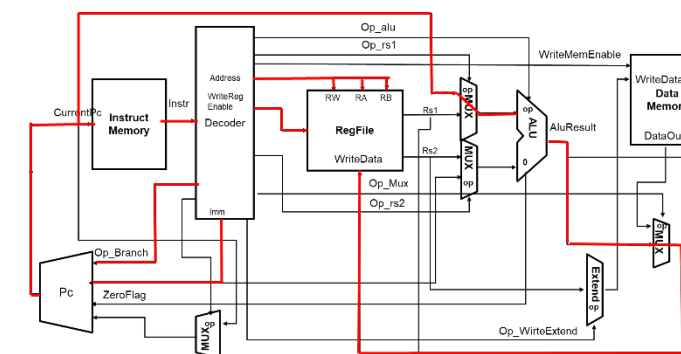
S型



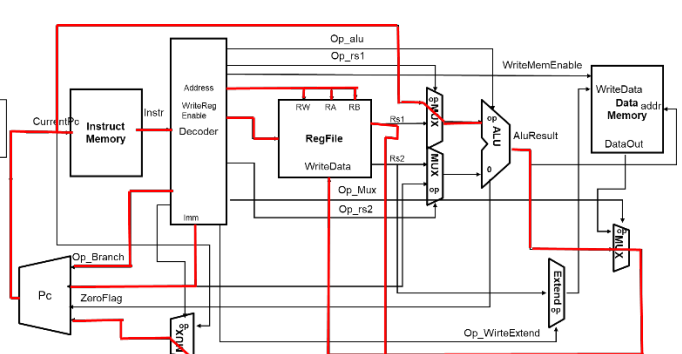
B型



U型 (黄色为auipc)



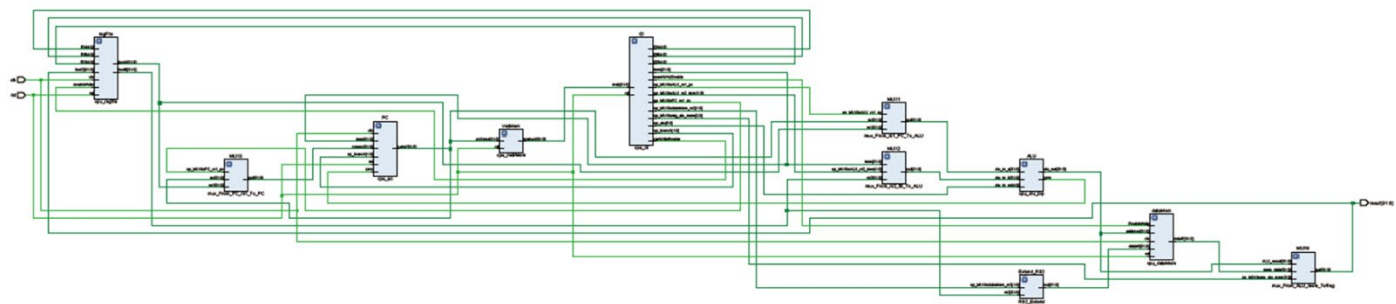
J (jal) 型



J (Jalr) 型

### 三、仿真测试

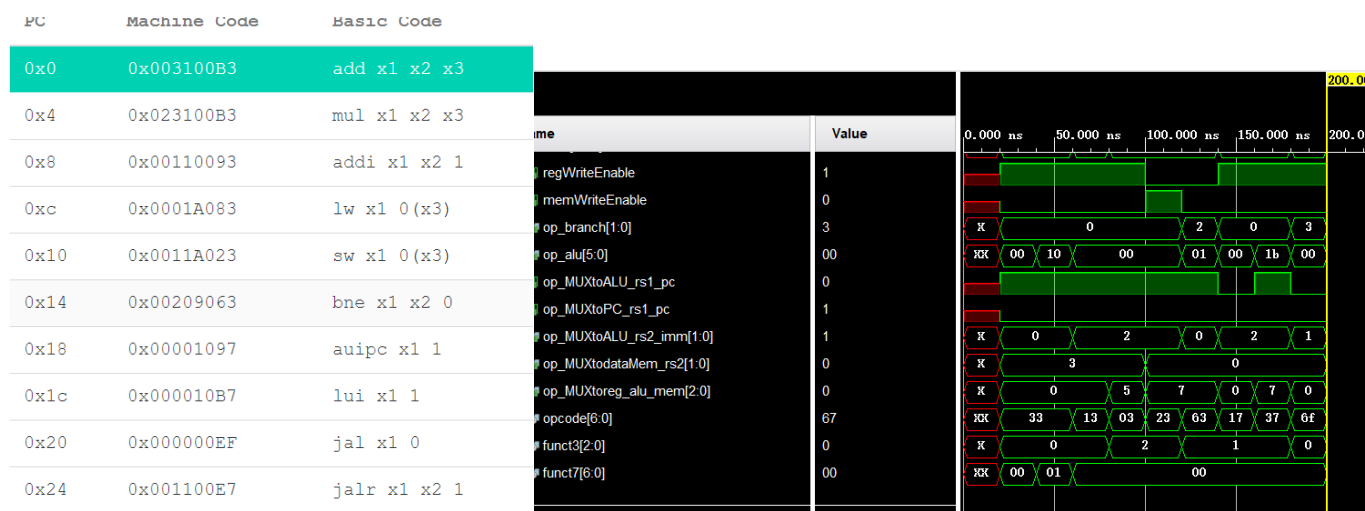
整体结构图如下：



#### 1、分模块测试

根据不同模块的功能进行测试，其中重点为译码器部分

- 译码器为本设计中最复杂的部分，也是最容易出错的部分，着重测试
- 采取TESTBENCH文件，使用不同的指令测试其控制信号
- 采用不同类型的指令各一条测试，如下图，对于每一条指令均能正常翻译指令生成对应的控制信号。



#### 2、CPU整体测试

具体步骤如下：

- (1) 编写两个小程序，要求预期覆盖到本CPU所支持的大部分指令
- (2) 使用C语言编写大部分内容，内联汇编实现读写存储器
- (3) 使用gcc-riscv 编译为汇编源码

编译环境与命令：

(xPack GNU RISC-V Embedded GCC, 64-bit) 10.1.0

```
riscv-none-embed-gcc -S prime.c -march=rv32im --specs=nosys.specs
```

#### (4) 使用riscv模拟器翻译为机器码

- 使用“UC Berkeley CS61C”课程提供的RISC-V模拟器，将汇编指令翻译为机器码，并模拟验证代码正确性
- 地址：<https://venus.cs61c.org/>
- 因为PC从0开始，因此将汇编程序的main段提到最前然后翻译为机器码

#### (5) 将机器码写入coe文件

#### (6) 将coe文件导入指令存储器

(7) 仿真运行，数据存储器输入不同的参数，观察数据存储器写回的的结果是否正确

#### 程序一：求素数：

从存储器首单元读出一个无符号十进制数，记为x

求前x个素数，并以此写入存储器 2~(x+1)个存储器单元内

为什么选这个程序：使用到了乘除法与分支跳转结构

C语言源代码 -> RISC-V汇编：

```
void main()
{
    int x = 2;
    int cnt = 0, pos = 0, des;

    __asm__ __volatile__ (
        "lw %[word], %[src1], %[src2]"
        : [word] "=r" (des)
        : [src1] "i" (0), [src2] "i" (0)
        ); //内联汇编 读取存储器首单元为素数数目

    while (cnt < des)
    {
        int i;
        int isPrime = 1;
        for (i = 2; i <= des; i++)
```

```
管理员: Windows PowerShell
Windows PowerShell
版权所有 (C) Microsoft Corporation. 保留所有权利。

尝试新的跨平台 PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32> E:
PS E:\> riscv-none-embed-gcc -S Prime.c -march=rv32im --specs=nosys.specs -O1
PS E:\> cat Prime.s
.file "Prime.c"
.option nopic
.text
.align 2
.globl main
.type main, @function

main:
    li    a1, 0
#APP
# 6 "Prime.c" 1
    lw    a1, 0, a1
# 0 "" 2
#NO_APP
    li    a0, 0
    li    a2, 0
    li    a4, 2
    li    a6, 2
    bgt   a1, zero, .L2
.LS:
#APP
# 40 "Prime.c" 1
    nop
# 0 "" 2
#NO_APP
    j     .LS
.L6:
    addi   a0, a0, 4
#APP
# 27 "Prime.c" 1
    sw    a4, 0, a0
# 0 "" 2
#NO_APP
```

## RISC-V汇编 -> 机器码

```

1 main:
2     li a1,0
3     #APP
4     # 6 "sushu.c" 1
5     lw a1, 0, a1
6     # 0 "" 2
7     #NO_APP
8     li a0,0
9     li a2,0
10    li a4,2
11    li a6,2
        
```

| PC   | Machine Code | Basic Code     | Original Code   |
|------|--------------|----------------|-----------------|
| 0x0  | 0x00000593   | addi x11 x0 0  | li a1,0         |
| 0x4  | 0x0005A583   | lw x11 0(x11)  | lw a1, 0, a1    |
| 0x8  | 0x00000513   | addi x10 x0 0  | li a0,0         |
| 0xc  | 0x00000613   | addi x12 x0 0  | li a2,0         |
| 0x10 | 0x00200713   | addi x14 x0 2  | li a4,2         |
| 0x14 | 0x00200813   | addi x16 x0 2  | li a6,2         |
| 0x18 | 0x00B04C63   | blt x0 x11 24  | bgt a1,zero,.L2 |
| 0x1c | 0x00008067   | jalr x0 x1 0   | ret             |
| 0x20 | 0x00450513   | addi x10 x10 4 | addi a0,a0,4    |
| 0x24 | 0x00160613   | addi x12 x12 1 | addi a2,a2,1    |

Registers Memory Cache VDB

Integer (R) Floating (F)

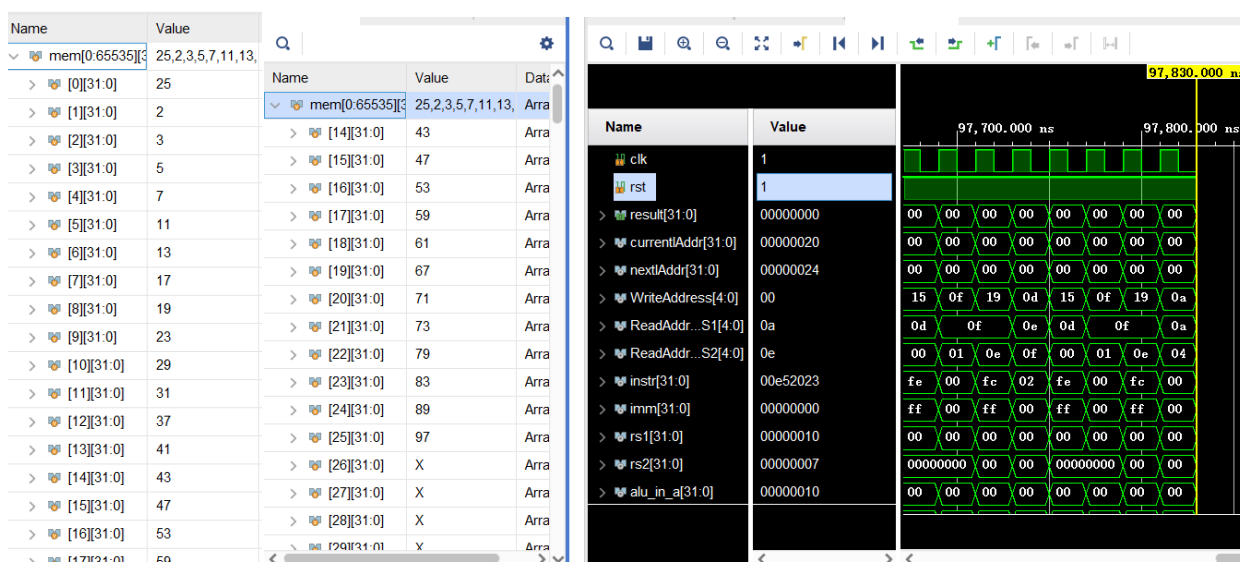
|         |            |
|---------|------------|
| zero    | 0x00000000 |
| ra (x1) | 0x00000000 |
| sp (x2) | 0x7FFFFFF0 |
| gp (x3) | 0x10000000 |
| tp (x4) | 0x00000000 |
| t0 (x5) | 0x00000000 |
| t1      | 0x00000000 |

写入coe文件，导入指令存储器，修改数据存储器参数，仿真测试：

```

always @ (posedge clk or negedge rst) begin //写入存储器
    if(~rst) begin
        //求素数程序测试时使用
        mem[0]<=32'd25; //取前25个素数
    end
end
        
```

数据存储器结果如下图，显然正确



CPU第4891周期写回第25个素数

程序二：

求乘法逆元（扩展欧几里得法）：

从存储器第一单元读出一个无符号十进制数，记为a

从存储器第二单元读出一个无符号十进制数，记为p

求a模p的乘法逆元x，并写入存储器第三单元内

为什么选这个程序：递归结构，存在大量的过程调用与存储器读写

写入coe文件，导入指令存储器，修改数据存储器参数，仿真测试：

```

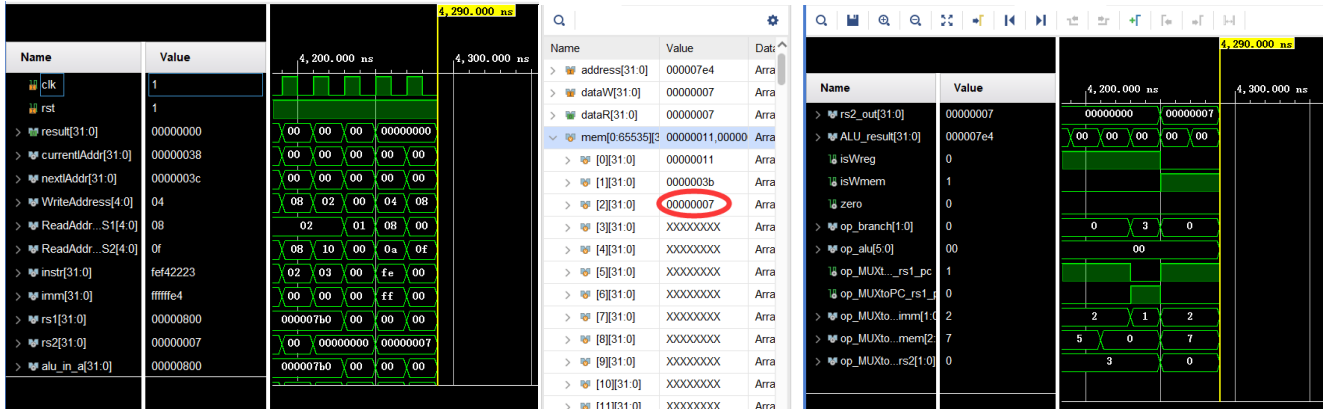
if(~rst) begin
    /* //求素数程序测试时使用
    mem[0]<=32'd10; //取前10个素数
    */
    //求乘法逆元程序测试时使用
    mem[0]<=32'd17; // a = 17
    mem[1]<=32'd59; // p = 59

```

// 17x=1(mod59) -> x = 7

结果正确  $17 \times 7 \equiv 1 \pmod{59}$

第281周期写回x=7



## 四、性能指标分析

- 支持45条指令，单周期CPI=1
- 资源使用情况如右图：
- 等效频率：

| Graph   Table |            |           |                |
|---------------|------------|-----------|----------------|
| Resource      | Estimation | Available | Utilization... |
| LUT           | 10090      | 32600     | 30.95          |
| LUTRAM        | 48         | 9600      | 0.50           |
| FF            | 16416      | 65200     | 25.18          |
| DSP           | 3          | 120       | 2.50           |
| IO            | 34         | 106       | 32.08          |
| BUFG          | 12         | 32        | 37.50          |

- 单周期CPU中，乘除取余等运算在一个周期内完成，因此频率会受到相当大的制约，性能较低。
- 时钟周期（clk周期）约束在75ns左右，综合后的时序报告未出现违例。

| Clock Summary |                |             |                 |
|---------------|----------------|-------------|-----------------|
| Name          | Waveform       | Period (ns) | Frequency (MHz) |
| clk           | {0.000 37.500} | 75.000      | 13.333          |

Tcl ConsoleMessagesLogReportsDesign RunsTiming x

Q

Design Timing Summary

General Information

Timer Settings

Design Timing Summary

Clock Summary (1)

Check Timing (1393)

Intra-Clock Paths

Inter-Clock Paths

Other Path Groups

Setup

Worst Negative Slack (WNS): 3.775 ns

Total Negative Slack (TNS): 0.000 ns

Number of Failing Endpoints: 0

Total Number of Endpoints: 33440

Hold

Worst Hold Slack (WHS): 0.147 ns

Total Hold Slack (THS): 0.000 ns

Number of Failing Endpoints: 0

Total Number of Endpoints: 33440

Pulse Width

Worst Pulse Width Slack (WPWS): 36.450 ns

Total Pulse Width Negative Slack (TPWS): 0.000 ns

Number of Failing Endpoints: 0

Total Number of Endpoints: 16513

All user specified timing constraints are met.

- RV32IM指令集CPU等效频率：13.33MHz

Q | — | | | | ● | Intra-Clock Paths - clk - Setup

| Name     | Slack | Levels | Routes | High Fanout | From              | To                 | Total... ▼ 1 | Logic Delay | Net Delay | Requirement | Source |
|----------|-------|--------|--------|-------------|-------------------|--------------------|--------------|-------------|-----------|-------------|--------|
| ↳ Path 1 | 3.775 | 314    | 315    | 155         | PC/curpc_reg[3]/C | PC/curpc_reg[31]/D | 71.101       | 47.466      | 23.635    | 75.0        | clk    |
| ↳ Path 2 | 3.783 | 314    | 315    | 155         | PC/curpc_reg[3]/C | PC/curpc_reg[29]/D | 71.093       | 47.458      | 23.635    | 75.0        | clk    |
| ↳ Path 3 | 3.830 | 314    | 315    | 155         | PC/curpc_reg[3]/C | PC/curpc_reg[30]/D | 71.046       | 47.411      | 23.635    | 75.0        | clk    |
| ↳ Path 4 | 3.853 | 314    | 315    | 155         | PC/curpc_reg[3]/C | PC/curpc_reg[28]/D | 71.023       | 47.388      | 23.635    | 75.0        | clk    |
| ↳ Path 5 | 3.864 | 313    | 314    | 155         | PC/curpc_reg[3]/C | PC/curpc_reg[27]/D | 71.012       | 47.377      | 23.635    | 75.0        | clk    |
| ↳ Path 6 | 3.872 | 313    | 314    | 155         | PC/curpc_reg[3]/C | PC/curpc_reg[25]/D | 71.004       | 47.369      | 23.635    | 75.0        | clk    |
| ↳ Path 7 | 3.910 | 313    | 314    | 155         | PC/curpc_reg[3]/C | PC/curpc_reg[26]/D | 70.957       | 47.322      | 23.635    | 75.0        | clk    |

## 五、总结

- 学习了单周期CPU核的架构：
  - 对课程内容有了更为深刻的理解，同时通过本设计，也感受到了单周期CPU在性能上的不足，相比于多周期和流水线CPU存在着巨大的提升空间。
- 学习了RISC-V指令集与其CPU的特性：
  - 通过对指令集手册的通读，了解到了RISC-V指令集的“优雅的”设计思想，相较于CISC指令集的精简、低功耗、模块化、可扩展等技术优势，有了直观的体会。
- 学习了RISC-V基本汇编：
  - 为了对设计的CPU核进行功能测试，学习了基于RISC-V的基本汇编内容，对其各寄存器的功能，函数调用的方式等有了大致的了解。