

# 2022 年计算机组成原理单周期型 32 位 CPU 设计项目展示

狄农雨 周诏盟 柯念昕 唐鑫

2022 年 4 月 15 日

# 设计环境

设计语言：Verilog 硬件描述语言

仿真环境：Vivado 2018.3

选用指令集：RISC-V (RV32I、RV32M)

# 设计目标

设计一个兼容 32 位 RISC-V 指令集的单周期 CPU。

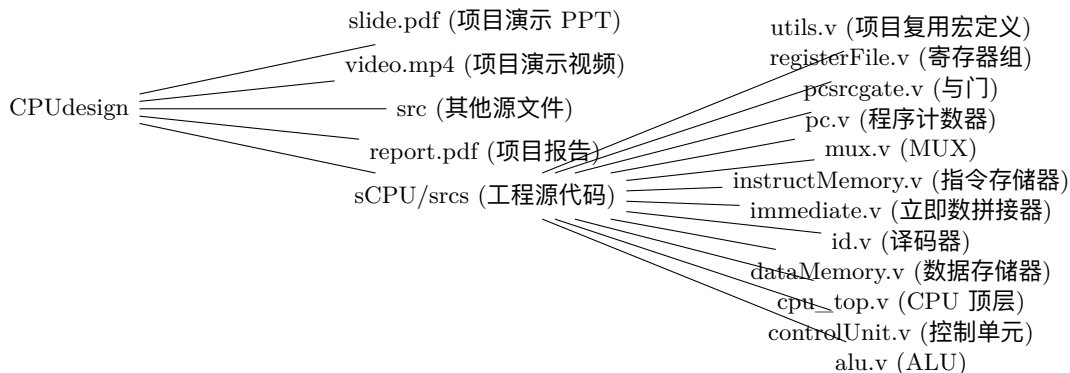
要求覆盖 RISC-V 基本指令集中与 MIPS 中 R 型、I 型计算型、I 型取数型、I 型存数型、I 型条件判断型、J 型对应的指令。

前仿（功能仿真）成功，并给出性能指标，包括：频率（或者等效频率）、已实现指令的条数和 CPI 等。

（附加）使用 C 语言编写简单程序，覆盖到 CPU 所支持的分指令，使用 RISC-V 交叉编译（gcc-riscv）为汇编源码，使用 RISC-V 模拟器翻译为机器指令并执行。

（附加）使用 FPGA 内部静态存储器，将机器指令写入 coe 文件，将 coe 文件导入指令存储器（和数据存储器），仿真运行，并观察数据存储器写回的结果。

# 文件目录结构



# 指令选取

根据 RISC-V 官方提供的资料, RISC-V 指令集共有六种类型, 分别为: R 型、I 型、S 型、B 型、U 型、J 型, 其对应的指令格式如图所示。

31	27	26	25	24	20	19	15	14	12	11	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 10:5]				rs2		rs1		funct3		imm[4:1 11]		opcode		B-type
imm[31:12]										rd		opcode		U-type
imm[20 10:1 11 19:12]										rd		opcode		J-type

Figure: RISC-V 基本指令格式

# 指令选取

根据类型覆盖要求，我们选取了 RV32I、RV32M 中的部分指令，具体指令及其格式如图所示。

31	27	26	25	24	20	19	15	14	12	11	7	6	0	
imm[31:12]										rd		0110111	LUI	
imm[31:12]										rd		0010111	AUIPC	
imm[20 10:1 11 19:12]										rd		1101111	JAL	
imm[11:0]						rs1	000			rd		1100111	JALR	
imm[12 10:5]				rs2		rs1	000			imm[4:1 11]		1100011	BEQ	
imm[12 10:5]				rs2		rs1	001			imm[4:1 11]		1100011	BNE	
imm[12 10:5]				rs2		rs1	100			imm[4:1 11]		1100011	BLT	
imm[12 10:5]				rs2		rs1	101			imm[4:1 11]		1100011	BGE	
imm[12 10:5]				rs2		rs1	110			imm[4:1 11]		1100011	BLTU	
imm[12 10:5]				rs2		rs1	111			imm[4:1 11]		1100011	BGEU	
imm[11:0]						rs1	010			rd		0000011	LW	
imm[11:5]				rs2		rs1	010			imm[4:0]		0100011	SW	

31	27	26	25	24	20	19	15	14	12	11	7	6	0	
imm[11:0]					rs1		000		rd		0010011			ADDI
imm[11:0]					rs1		010		rd		0010011			SLTI
imm[11:0]					rs1		011		rd		0010011			SLTIU
imm[11:0]					rs1		100		rd		0010011			XORI
imm[11:0]					rs1		110		rd		0010011			ORI
imm[11:0]					rs1		111		rd		0010011			ANDI
0000000				shamt	rs1		001		rd		0010011			SLLI
0000000				shamt	rs1		101		rd		0010011			SRLI
0100000				shamt	rs1		101		rd		0010011			SRAI
0000000				rs2	rs1		000		rd		0110011			ADD
0100000				rs2	rs1		000		rd		0110011			SUB
0000000				rs2	rs1		001		rd		0110011			SLL
0000000				rs2	rs1		010		rd		0110011			SLT
0000000				rs2	rs1		011		rd		0110011			SLTU
0000000				rs2	rs1		100		rd		0110011			XOR
0000000				rs2	rs1		101		rd		0110011			SRL
0100000				rs2	rs1		101		rd		0110011			SRA

31	27	26	25	24	20	19	15	14	12	11	7	6	0	
0000000				rs2		rs1		110		rd		0110011		OR
0000000				rs2		rs1		111		rd		0110011		AND
0000001				rs2		rs1		000		rd		0110011		MUL
0000001				rs2		rs1		001		rd		0110011		MULH
0000001				rs2		rs1		010		rd		0110011		MULHSU
0000001				rs2		rs1		011		rd		0110011		MULHU
0000001				rs2		rs1		100		rd		0110011		DIV
0000001				rs2		rs1		101		rd		0110011		DIVU
0000001				rs2		rs1		110		rd		0110011		REM
0000001				rs2		rs1		111		rd		0110011		REMU

Figure: 选取的 RISC-V 指令及对应格式



# 指令功能描述

本项目中涉及到的所有指令的功能在官方手册第 18-25、51-52 页有详细描述，在此不再赘述。

## 完整数据通路

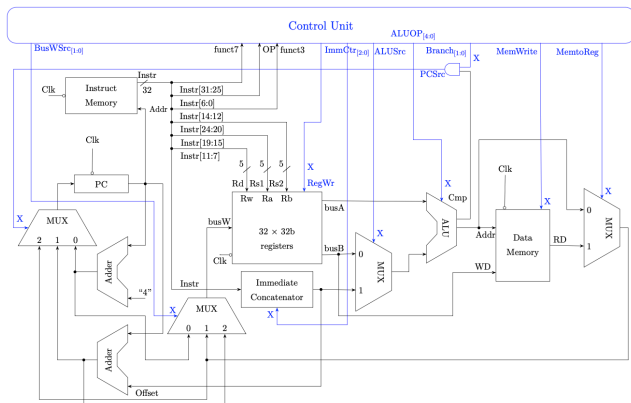


Figure: 完整数据通路

## R 型数据通路

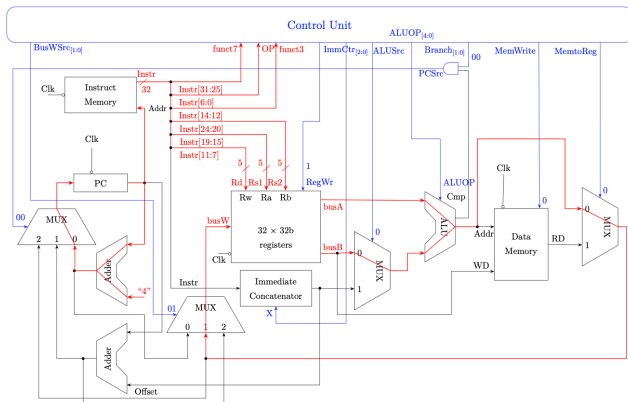


Figure: R 型数据通路

## I 型数据通路

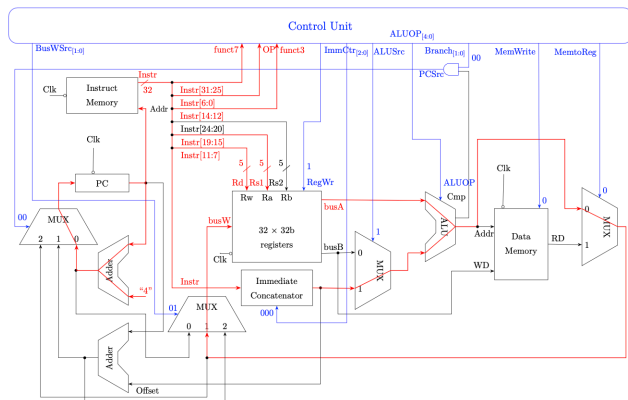


Figure: I 型数据通路

## I 型数据通路 (lw)

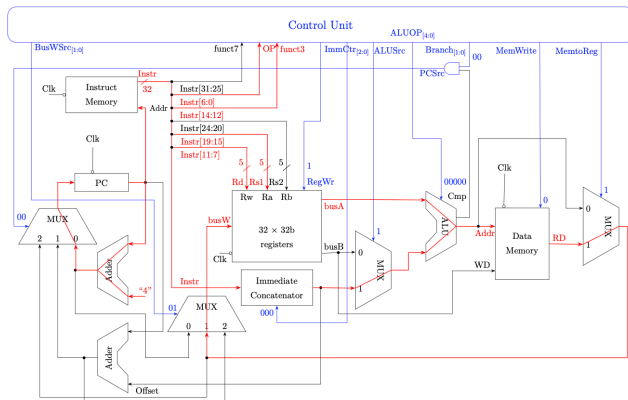


Figure: I 型数据通路 (lw)

## S 型数据通路 (SW)

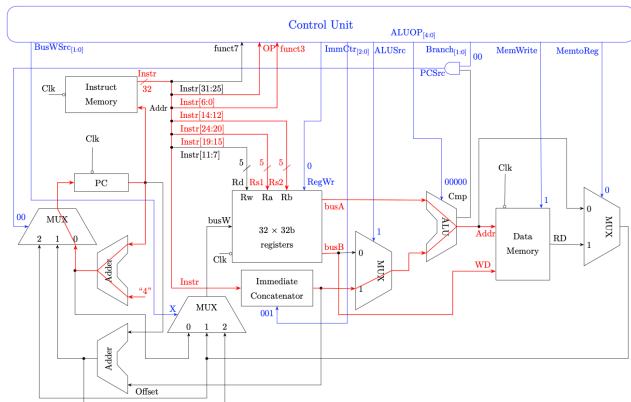


Figure: S 型数据通路 (sw)

## B 型数据通路

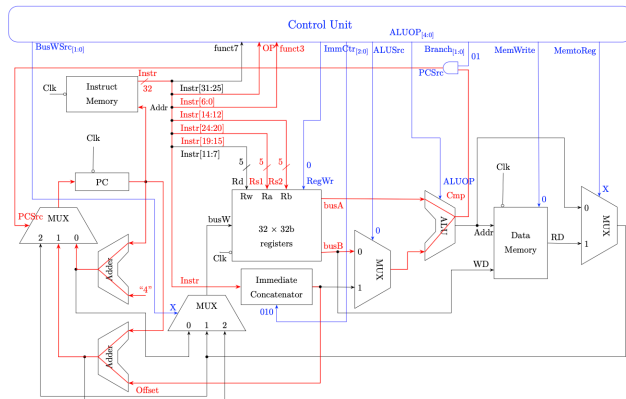


Figure: B 型数据通路

## U 型数据通路

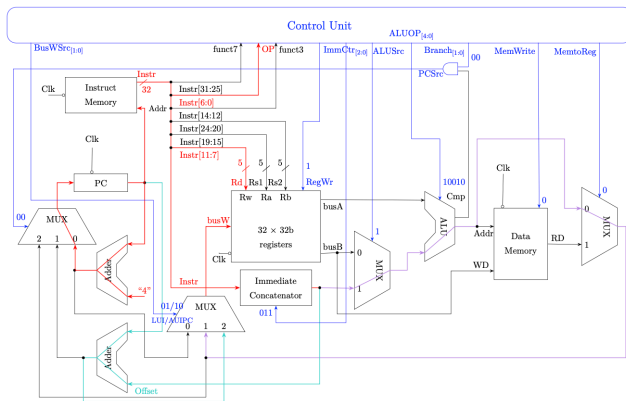


Figure: U 型数据通路（红色为共用、紫色为 LUI 通路、青色为 AUIPC 通路）



## J 型数据通路

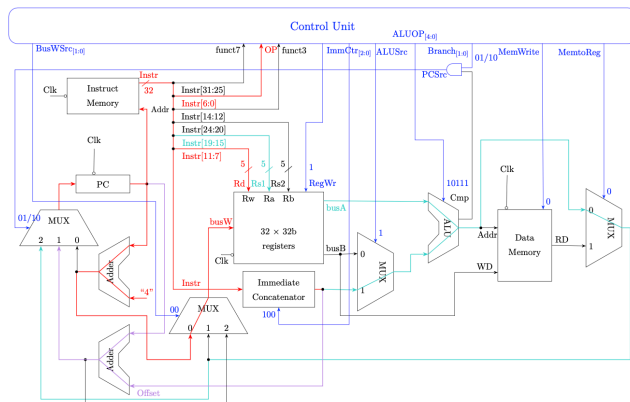


Figure: J 型数据通路 (红色为共用、紫色为 JAL 通路、青色为 JALR 通路)

# 立即数拼接器

由 RISC-V 的 spec 手册提供的信息，立即数有 5 类，分别为 I 型、S 型、B 型、U 型、J 型，如图所示。为了简化数据通路图，我们将五类立即数的处理放进立即数拼接器中，通过控制信号 ImmCtr[2:0] 来控制生成的立即数类型。

31	30	20 19		12	11	10	5	4	1	0	
— inst[31] —						inst[30:25]	inst[24:21]		inst[20]		I-immediate
— inst[31] —						inst[30:25]	inst[11:8]		inst[7]		S-immediate
— inst[31] —				inst[7]		inst[30:25]	inst[11:8]		0		B-immediate
inst[31]	inst[30:20]		inst[19:12]		— 0 —						U-immediate
— inst[31] —			inst[19:12]		inst[20]	inst[30:25]	inst[24:21]		0		J-immediate

Figure: 立即数类型

# 程序计数器

由于 PC 的值需要单独存放并根据时钟信号进行修改，故将 PC 对应的寄存器模块化，且该模块只完成 PC 的存储和输出。

# 译码器

由单周期 CPU 的工作流程，我们需要使 CPU 在取指后完成译码工作。结合数据通路图，译码后需要分为 7 条线输出到其他模块，为了避免顶层代码杂乱，此处进行模块化处理。

# 控制单元

考虑到选取的指令条数较多，控制信号的情况较为复杂，故选择将控制单元设计为专用模块，根据 OP、funct3、funct7 的输入对 CPU 内所有模块进行控制。各类型指令对应的操作信号如图。其中，X 表示与当前信号无关，TBD 表示取决于具体的输入。各项操作信号与数据通路中所标信号名一致。

信号	BusWSrc	RegWr	ImmCtr	ALUSrc	ALUOP	Branch	Cmp	MemWrite	MemtoReg
R 型	01	1	XXX	0	TBD	00	XX	0	0
I 型	01	1	000	1	TBD	00	XX	0	0
lw	01	1	000	1	00000	00	XX	0	1
S 型	XX	0	001	1	00000	00	XX	1	0
B 型	XX	0	010	0	TBD	01	00 / 11	0	X
U 型 (LUI / AUIPC)	01 / 10	1	011	1	10010	00	00 / 11	0	0
J 型 (JAL / JALR)	00	1	100	1	00000	01 / 10	00 / 11	0	0

Figure: 控制信号表

# 存储器

存储器在 CPU 中共有三个，分别是指令存储器、寄存器组和数据存储器。

为了方便导入 coe 文件，指令存储器选用了基于 Vivado 提供的 dist\_mem\_gen IP 核的实现方式（ROM，数据宽度 32 位，深度 4096）。在预加载 coe 文件作为操作指令的情况下，指令存储器可以以只读方式工作。

根据 RISC-V 手册 14 页所规定的内容，寄存器组中定义了 32 个 32 位寄存器，其中 x0 寄存器为只读寄存器，且值恒为 0。

考虑到 CPU 测试及运行需要，数据存储器的设计为 4096 个 32 位寄存器。寄存器组和数据存储器均采用 Verilog 代码定义生成。

# ALU

经过对所有指令执行的运算的分析，我们发现 ALU 需要处理 24 种不同的运算。ALU 对控制信号的处理采用了 switch 结构，根据控制信号 ALUOP 的值选择对应的运算。运算的实现采用 Verilog 提供的运算符完成。需要加以说明的是，ALU 将零标志改为了 cmp 标志，表示比较运算的结果，与控制信号 Branch 进行按位与运算，进而对 PC 前 MUX 进行控制，实现分支与跳转。

# 多路选择器

项目中的 MUX 有两种，分别是两路 32 位选择器和三路 32 位选择器，设计满足 CPU 内数据通路选择需求。



# 加法器

为了解决 PC 值的修改问题，在 CPU 中引入了加法器组件。用于处理  $PC=PC+4$  和  $PC=PC+offset$  两种情况的运算需求，功能为将两输入端的值求和输出，结果直接输出到 PC 前 MUX。

# 加法器

由于部分指令的操作码和功能码以及部分控制信号都是二进制常数，为了避免因部分人为写错造成的麻烦，我们将所有的常数值定义在 `utils.v` 中，并以其含义命名。这也一定程度上加快了项目代码的完成进程。

# 功能仿真

在前文中，我们提到了指令存储器使用 IP 核的原因。在仿真测试中，我们将经过交叉编译的汇编代码放入 RISC-V 模拟器 (<https://venus.cs61c.org/>) 中翻译为机器指令，存储在 coe 文件中，令指令存储器使用 coe 文件进行初始化。

为测试 CPU 的工作状况是否符合预期，我们使用汇编代码编写了简单的程序覆盖各类指令对 CPU 进行测试。代码经模拟器翻译后，写入 instrData.coe 文件中，经 IP 核加载到指令存储器中。

# 功能仿真

写入指令存储器后进行仿真测试，得到的结果如图。

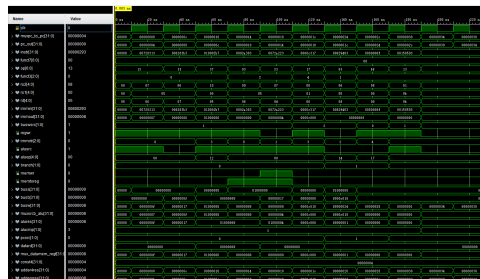


Figure: 仿真测试结果

运行结果与 RISC-V 模拟器运行结果比对后无错误，仿真期间没有错误信息报出，故可以认为功能仿真成功。

# 设计综合

运行设计综合后，得到 CPU 的整体结构图。

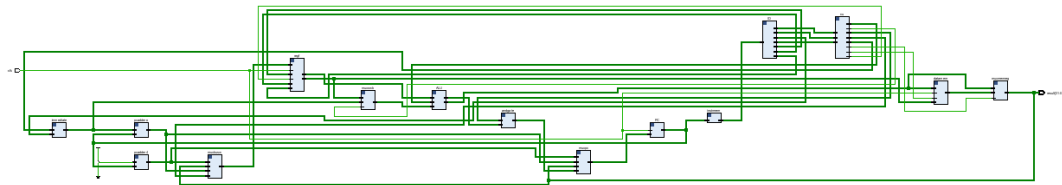


Figure: CPU 整体结构图

# 设计综合

在运行设计综合与布局布线后，可以得到相关性能指标。如图所示。可以得出 CPU 频率为：100.000MHz。实现指令条数为 39 条，CPI=1。

Clock Summary			
Name	Waveform	Period (ns)	Frequency (MHz)
clk	{0.000 5.000}	10.000	100.000

Figure: 时钟频率信息

Power		Summary   On-Chip
Total On-Chip Power:	0.165 W	
Junction Temperature:	26.9 °C	
Thermal Margin:	58.1 °C (4.9 W)	
Effective SJA:	11.5 °C/W	
Power supplied to off-chip devices:	0 W	
Confidence level:	Medium	
<a href="#">Implemented Power Report</a>		

Figure: 能耗信息

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 2.157 ns	Worst Hold Slack (WHS): 0.230 ns	Worst Pulse Width Slack (WPWS): 3.670 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 4672	Total Number of Endpoints: 4672	Total Number of Endpoints: 2177

All user specified timing constraints are met.

Figure: 时间信息

Utilization				Post-Synthesis   Post-Implementation
				Graph   Table
Resource	Utilization	Available	Utilization %	
LUT	7720	53200	14.51	
LUTrAM	2096	17400	12.05	
FF	32	106400	0.03	
DSP	3	220	1.36	
ID	33	125	26.40	
BUFG	5	32	15.63	

Figure: 资源使用信息

## 程序描述

在程序运行测试方面，考虑到需要覆盖所有指令类型，在此编写的程序为求质数的乘法逆元。

乘法逆元的定义：在模  $p$  运算下，若  $a \times b \equiv 1 \pmod{p}$ ，则称  $a, b$  互为乘法逆元。

乘法逆元的求法：根据费马小定理，对于质数  $p$  和非  $p$  的倍数  $a$ ，有  $a^{p-1} \equiv 1 \pmod{p}$ 。由此， $a \cdot a^{p-2} \equiv 1 \pmod{p}$ ，也即  $a$  在模  $p$  意义下的乘法逆元为  $a^{p-2} \pmod{p}$ 。

程序目的：求 1 到  $n$  模  $p$  的乘法逆元，在此选取  $p = 4817$ ， $n$  从数据存储器的第一个位置读取。（源文件：src/inverse.c）

## 程序描述

程序运行过程：将 `inverse.c` 用 `gcc-riscv` 进行编译（编译命令：`riscv-none-embed-gcc inverse.c -S -march=rv32im -O1`），得到汇编命令文件 `inverse.s`。将 `inverse.s` 中的代码放入 RISC-V 模拟器进行翻译，得到机器码并按 `coe` 文件格式写到 `inverse.coe`。将指令存储器设置为从 `inverse.coe` 文件中读取，由于 `inverse.coe` 中对应 `main` 代码块起始语句的代码指令地址在 `0x40`，故在运行前还需要将 PC 的初值设置为 `32'H40`。在数据存储器中将 `data[0]` 设置为 `32'H8`，即 `n=8`。模拟运行。得到的数据如图。经验证，运行结果正确。



# 运行结果

Scope

×

Sources

—

□

□

□

🔍

⌵

⌶

⚙️

Name	Design U...	Block
▼  TESTBENCH	TESTBE...	Verilog
▼  cpu	cpu_top	Verilog
PC	pc	Verilog
>  instrmem	instructM...	Verilog
ID	id	Verilog
cu	controlUnit	Verilog
regf	registerF...	Verilog
immediate	immedia...	Verilog
ALU	alu	Verilog
andgate	pcsrcgate	Verilog
datamem	dataMem...	Verilog
muxsrcb	mux	Verilog
muxmemreg	mux	Verilog
muxbusw	mux3	Verilog
muxpc	mux3	Verilog

Objects

?

—

□

□

□

×

🔍

⌵

⌶

⚙️

Name	Value
clk	0
>  addr[31:0]	00000000
>  wd[31:0]	00000000
memwr	0
>  rd[31:0]	00000008
▼  data[0:4096][31:0]	00000008,0000
>  [0][31:0]	00000008
>  [1][31:0]	00000001
>  [2][31:0]	00000969
>  [3][31:0]	00000646
>  [4][31:0]	00000e1d
>  [5][31:0]	00000787
>  [6][31:0]	00000323
>  [7][31:0]	00001021
>  [8][31:0]	00001077

Figure: 程序运行结果

# 运行结果

$i$	$(inv[i])_{16}$	$(inv[i])_{10}$	$i \times inv[i]$	$i \times inv[i] \bmod p$
1	0x 0000 0001	1	1	1
2	0x 0000 0969	2409	4818	1
3	0x 0000 0646	1606	4818	1
4	0x 0000 0e1d	3613	14452	1
5	0x 0000 0787	1927	9635	1
6	0x 0000 0323	803	4818	1
7	0x 0000 1021	4129	28903	1
8	0x 0000 1077	4215	33720	1

Figure: 程序结果正确性检验

Name	Value	0 ns	20 ns	40 ns	60 ns	80 ns	100 ns	120 ns	140 ns	160 ns	180 ns	200 ns	220 ns	240 ns	260 ns	280 ns	300 ns				
clk	0																				
♥ mupc_in_pc[31:0]	00000044	010101	00100143	0010014c	01010150	01010154	00100159	0010015c	00100160	01010164	01010168	0010016c	01010170	01010174	01010178	0010017c	01010180				
♥ pc_out[31:0]	00000040	010101	00100144	00100148	0101014c	01010150	00100154	00100158	0010015c	01010160	01010164	01010168	0101016c	01010170	01010174	01010178	0010017c				
♥ inst[31:0]	161010113	1c111	00112c23	00112c23	00112c23	01112823	01112823	01412423	1010107b7	010107793	010107793	011010413	010114b7	0104b493	2414a113	2c4b493	101010b7				
♥ func0[6:0]	7f	7f																			
♥ op[6:0]	13	13	25																		
♥ func0[32:0]	0	0	7f																		
♥ rs2[4:0]	00	00	01	88	09	12	13	14	80				01	00	11	0f	00				
♥ rs1[4:0]	02	02																			
♥ rd[4:0]	02	02	1c	18	14	19	0c	80	8f				09	09	14	09	13				
♥ immim[31:0]	161010113	1c111	00112c23	00112c23	00112c23	01112823	01112423	01412423	1010107b7	010107793	010107793	011010413	010114b7	0104b493	2414a113	2c4b493	101010b7				
♥ immou[31:0]	imm0	ffff	0010111c	01010110	01010114	01010118	0010101c	00101010	10101010	01010108	01010108	01010103	01011310	01010100	01010101	01010101	10101010				
♥ buswarp[1:0]	1																				
♥ regw	1																				
♥ immich[2:0]	1	0																			
♥ aksrc	0																				
♥ akupc[4:0]	00																				
♥ branch[1:0]	0																				
♥ memw	0																				
♥ memtoreg	0																				
♥ busa[31:0]	00000000	010101	ffffffc0															00101010	10101010	01010108	01010101
♥ busb[31:0]	00000000																				
♥ busw[31:0]	imm0	ffff	ffffffc0	ffffffc0	ffffffc4	ffffffc0	ffffffc0	ffffffc0	ffffffc0	10101010	01010108	01010101	01010100	01010100	010112c1	010112cf	10101010				
♥ muxsrc_alu[31:0]	imm0	ffff	0010011c	01010118	01010114	01010110	0101010c	01010108	10101010	01010108	01010101	01010101	01011310	01010100	01010101	010112c1	10101010				
♥ akres[31:0]	imm0	ffff	ffffffc0	ffffffc0	ffffffc4	ffffffc0	ffffffc0	ffffffc0	ffffffc0	10101010	01010108	01010101	01010100	01010100	010112c1	010112cf	10101010				
♥ akucmp[1:0]	3																				
♥ pcsrc[1:0]	0																				
♥ datard[31:0]	00000000																				
♥ mux_datamem_reg[31:0]	imm0	ffff	ffffffc0	ffffffc0	ffffffc4	ffffffc0	ffffffc0	ffffffc0	ffffffc0	10101010	01010108	01010101	01010100	01010100	010112c1	010112cf	10101010				
♥ cons[4:3:0]	00000004																				
♥ adder4res[31:0]	00000044	010101	00100143	0101014c	01010150	01010154	00100159	0010015c	00100160	01010164	01010168	0101016c	01010170	01010174	01010178	0010017c	01010180				
♥ adder0res[31:0]	00000020	010101	01010104															01010108	01010101	01010100	01010100

Figure: 程序仿真测试截图

在本次设计项目中，我们组学习了单周期 CPU 的设计流程，对数据通路有了更加深入的理解。学习了 RISC-V 指令集的指令类型和基础指令，学习了 Vivado 的使用和 IP 核的调用，学习了 C 语言内联汇编的语法和交叉编译并翻译为机器码的方法，提高了调试过程中修复错误的能力。通过实践对课堂上学习到的计算机组成原理理论内容有了更加深刻的感悟。