实验二：单周期CPU设计

1. 实验目的

1. 掌握RISCV指令格式与编码，掌握RISCV汇编转换成机器代码的方法。
2. 掌握单周期CPU的数据通路与控制器设计方法。
3. 掌握硬件描述语言与EDA工具。
4. 掌握基本测试与调试方法。

2. 实验环境

> *Vivado 2019.2*
>
> *RARS 1_6*
>
> *jdk-11.0.19_windows-x64*
>
> *Windows 11 22H2 22621.1972*

3. 实验内容

1. 设计并实现单周期处理器，支持包含RISC-V RV32I整数指令（LW，SW，ADD，SUB， ORI，BEQ）在内的指令子集等。
2. 测试程序：完成2后在该cpu上实现对5个整数的排序，仿真测试结 果与RARS中运行结果对比正确。
3. 乐学提交实验报告、工程源代码和测试程序。

4. 实验步骤

4.1 确定指令集指令条数、指令格式及编码

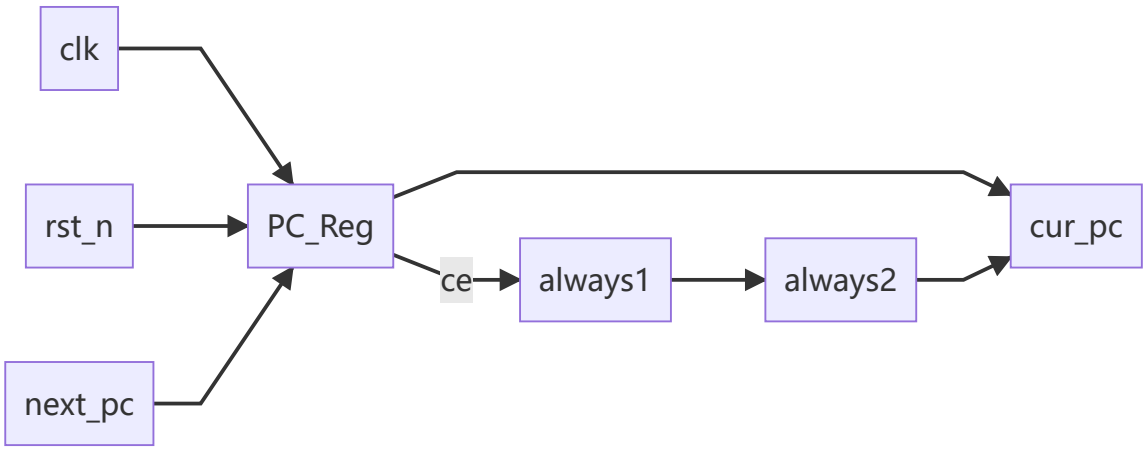
按照序号 1 填充表格 1。1 条指令一行，可以扩充，最后给出指令总数目。

序号	助记符	操作码	功能	描述	funct3	funct7	类型
1	ADD	0110011	R[rd]←R[rs1]+R[rs2]; PC←PC+4	加法	000	0000000	R
2	SUB	0110011	R[rd]←R[rs1]-R[rs2]; PC←PC+4	减法	000	0100000	R
3	ORI	0010011	R[rd]←R[rs1]|{imm[11:0]}; PC←PC+4	逻辑或立即数	110	\	I
4	LUI	0110111	R[rd]←{imm[31:12], 12'b0}; PC←PC+4	加载高位立即数	\	\	U
5	SW	0100011	M[{R[rs1]+imm[11:0]}]←R[rs2]; PC←PC+4	存储字节	010	\	S
6	LW	0000011	R[rd]←M[{R[rs1]+imm[11:0]}]; PC←PC+4	加载字节	010	\	I
7	BEQ	1100011	if(R[rs1]==R[rs2]) PC←PC+{imm[11:0], 1'b0}; else PC←PC+4	分支相等	000	\	B
8	JAL	1101111	R[rd]←PC+4; PC←PC+{imm[20],imm[10:1],imm[11],imm[19:12],1'b0}	跳转并链接	\	\	J
总数	8						

4.2 设计并实现 CPU 数据通路

4.2.1 寄存器堆

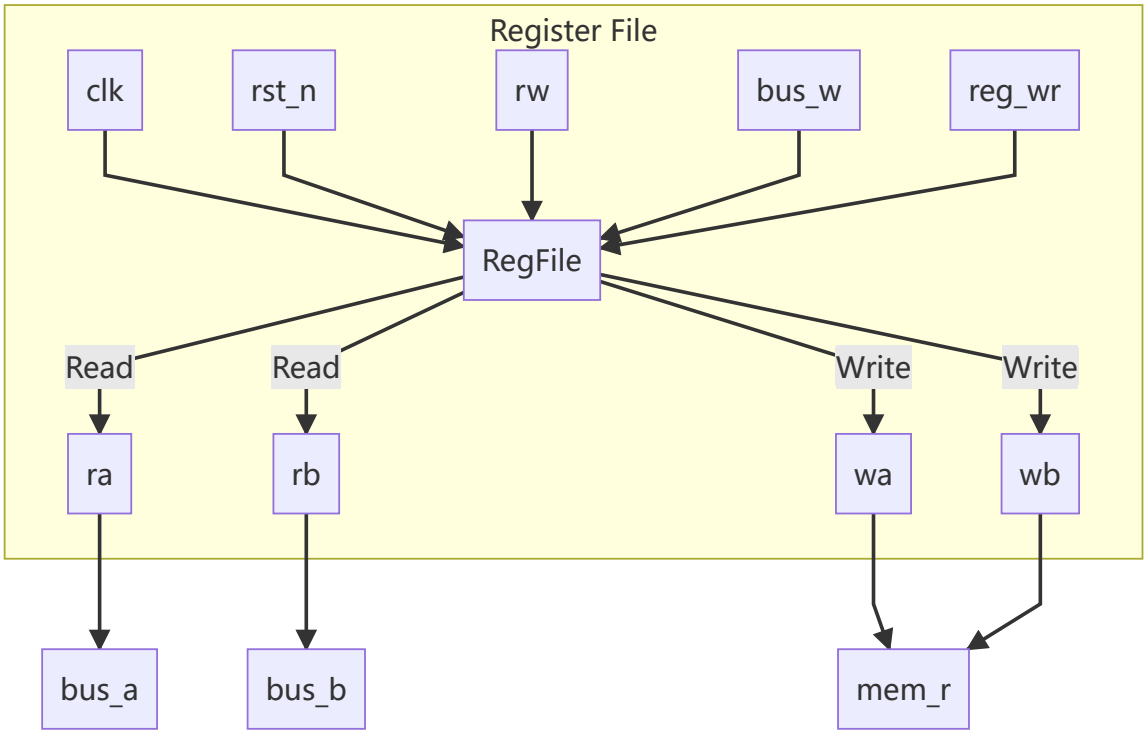
4.2.1.1 PC寄存器



PC 寄存器，用于存储下一条指令地址并输出当前指令地址。

输入端口	方向	位宽	描述
clk	输入	1	时钟信号
rst_n	输入	1	复位信号
next_pc	输入	32	下一条指令地址
ce	输入	1	时钟使能信号
cur_pc	输出	32	当前指令地址

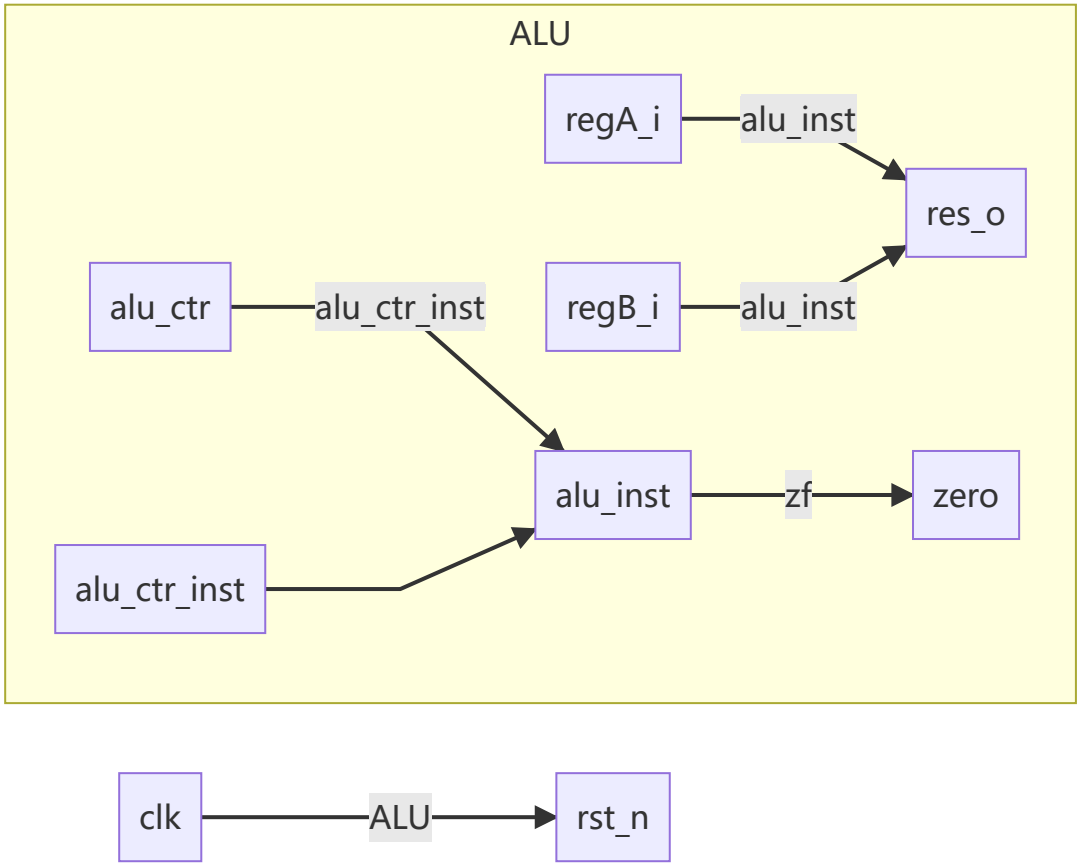
4.2.1.2 寄存器文件



寄存器文件（Register File），用于存储和读取 CPU 中使用的寄存器的值。

输入端口	方向	位宽	描述
rst_n	输入	1	复位信号
clk	输入	1	时钟信号
rw	输入	5	寄存器选择信号
bus_w	输入	32	数据输入端口
reg_wr	输入	1	写使能信号
bus_a	输出	32	输出端口 A 的值
bus_b	输出	32	输出端口 B 的值

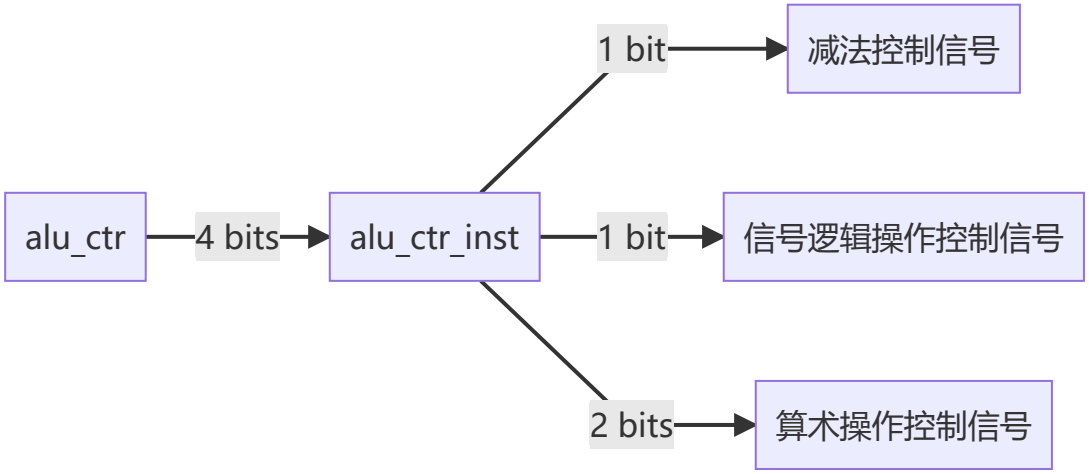
4.2.2 ALU



ALU（算术逻辑单元），它基于 4 位控制信号 `alu_ctr` 对两个 32 位输入 `regA_i` 和 `regB_i` 执行算术和逻辑操作。ALU 模块实例化了两个子模块：`alu_ctr` 和 `alu`。

输入端口	方向	位宽	描述
alu_ctr	输入	4	ALU 控制信号
regA_i	输入	32	输入端口 A 的值
regB_i	输入	32	输入端口 B 的值
clk	输入	1	时钟信号
rst_n	输入	1	复位信号
zf	输出	1	零标志
res_o	输出	32	操作结果的值

4.2.2.1 ALU控制模块

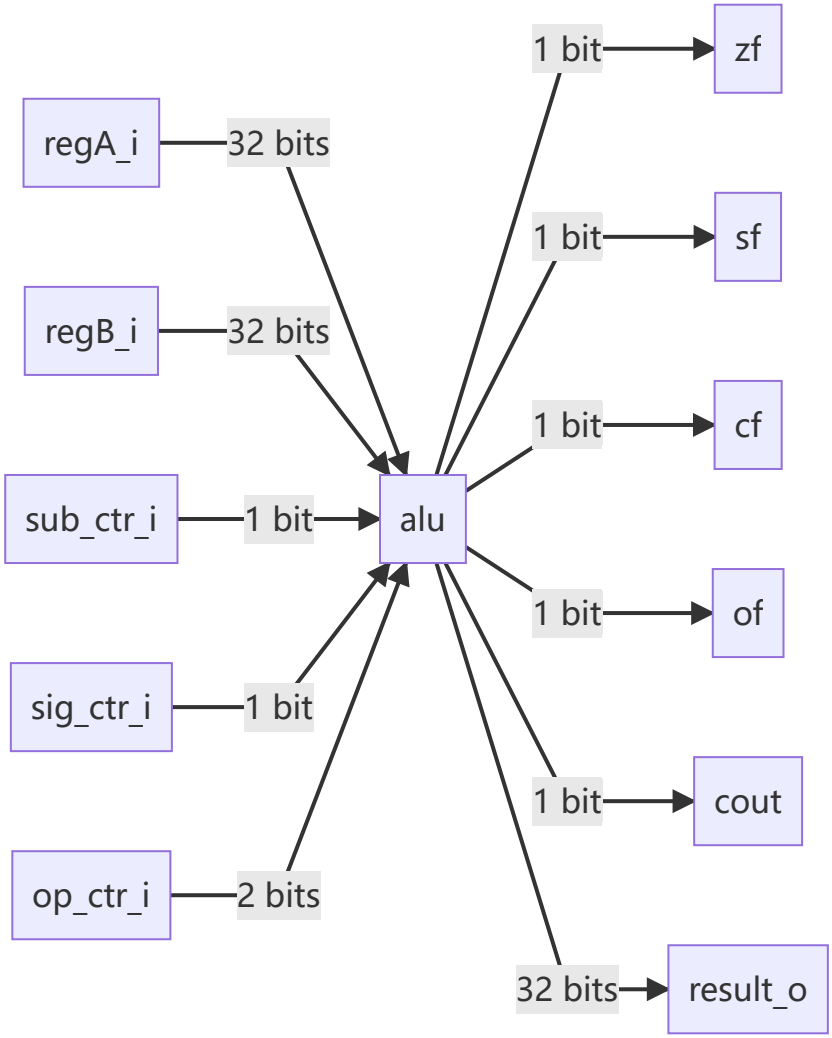


ALU 控制模块，它将 4 位的 `alu_ctr` 输入转换为三个控制信号：`sub_ctr`、`sig_ctr` 和 `op_ctr`。

输入端口	方向	位宽	描述
alu_ctr	输入	4	ALU 控制信号

输出端口	方向	位宽	描述
sub_ctr	输出	1	减法控制信号
sig_ctr	输出	1	信号逻辑操作控制信号
op_ctr	输出	2	算术操作控制信号

4.2.2.2 ALU

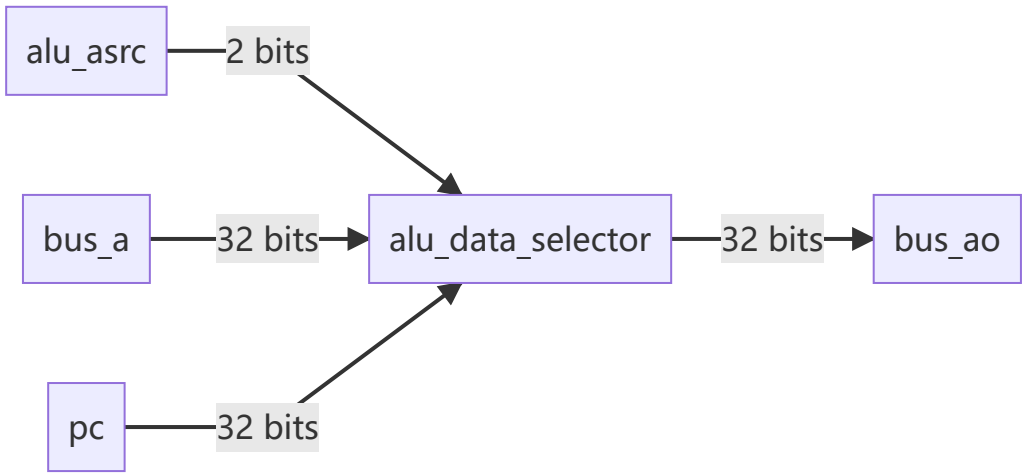


ALU，它执行指定的算术或逻辑操作，并生成一些标志位和输出结果。

输入端口	方向	位宽	描述
regA_i	输入	32	寄存器 A 的输入
regB_i	输入	32	寄存器 B 的输入
sub_ctr_i	输入	1	减法控制信号
sig_ctr_i	输入	1	信号逻辑操作控制信号
op_ctr_i	输入	2	算术操作控制信号

输出端口	方向	位宽	描述
zf	输出	1	零标志位
sf	输出	1	符号标志位
cf	输出	1	进位标志位
of	输出	1	溢出标志位
cout	输出	1	最高有效位进位标志位
result_o	输出	32	操作结果的输出

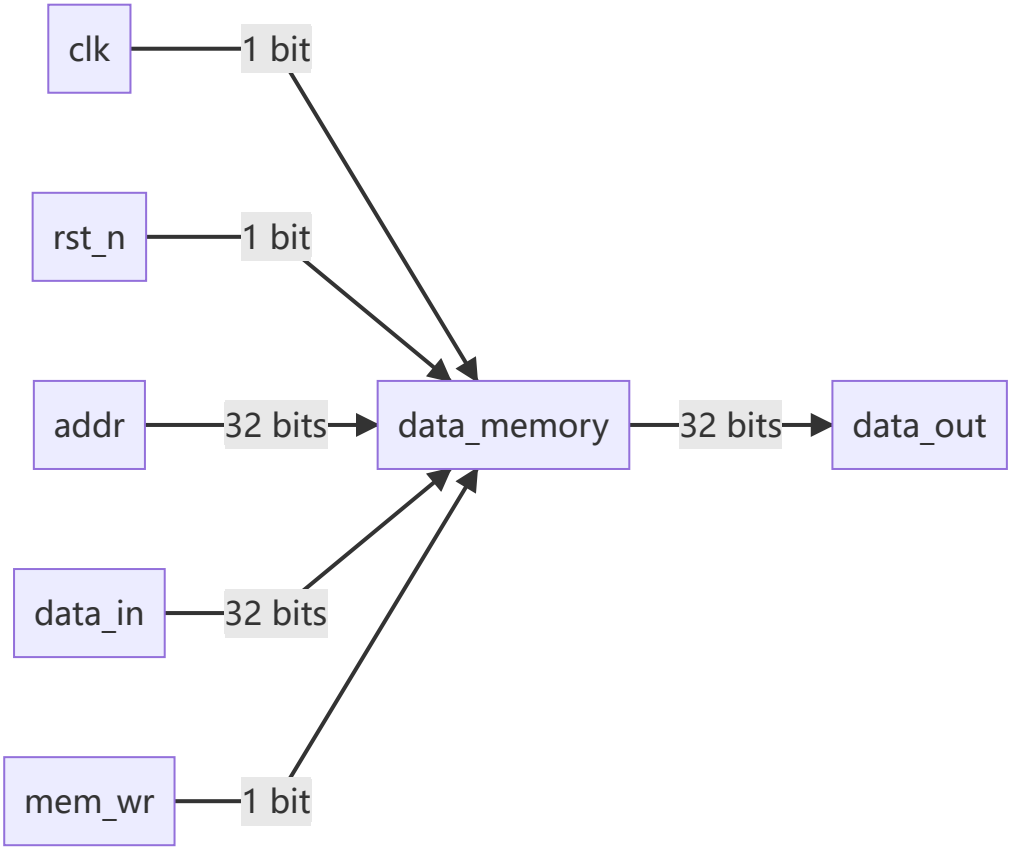
4.2.2.3 数据输入选择器



ALU 数据输入选择器，用于从两个输入端口中选择一个作为 ALU 的第一个操作数。

输入端口	方向	位宽	描述
alu_asrc	输入	2	ALU 数据输入选择信号
bus_a	输入	32	数据输入端口
pc	输入	32	数据输入端口
bus_ao	输出	32	数据输出端口

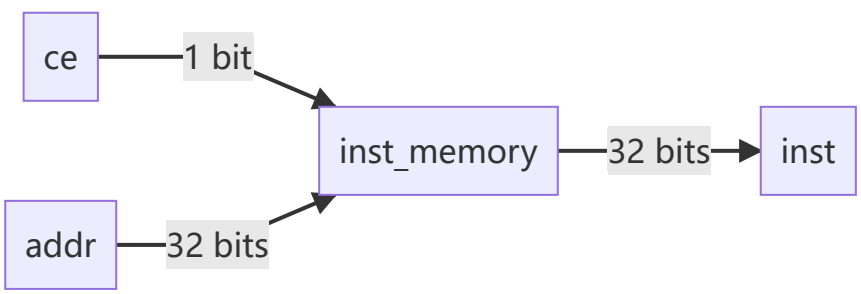
4.2.3 数据存储器



数据存储器，用于存储和读取数据。该模块有四个输入和输出端口，分别是时钟信号 `clk`、复位信号 `rst_n`、读写地址 `addr`、写入数据 `data_in`、写使能信号 `mem_wr` 和读出数据 `data_out`。

输入/输出端口	方向	位宽	描述
clk	输入	1	时钟信号
rst_n	输入	1	复位信号
addr	输入	32	读写地址
data_in	输入	32	写入数据
mem_wr	输入	1	写使能信号
data_out	输出	32	读出数据

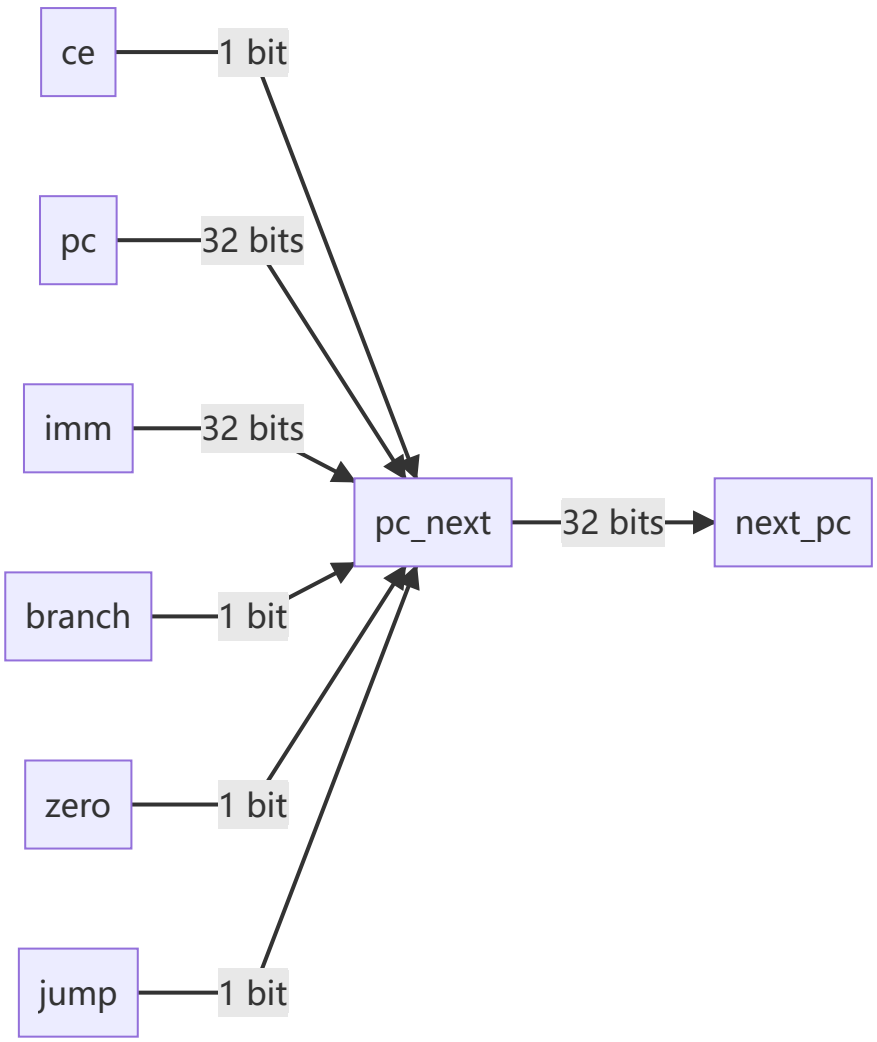
4.2.4 指令存储器



指令存储器，用于存储和读取指令。该模块有三个输入和输出端口，分别是使能信号 `ce`、读取地址 `addr` 和读出指令 `inst`。

输入/输出端口	方向	位宽	描述
ce	输入	1	使能信号
addr	输入	32	读取地址
inst	输出	32	读出指令

4.2.5 指令地址计算器

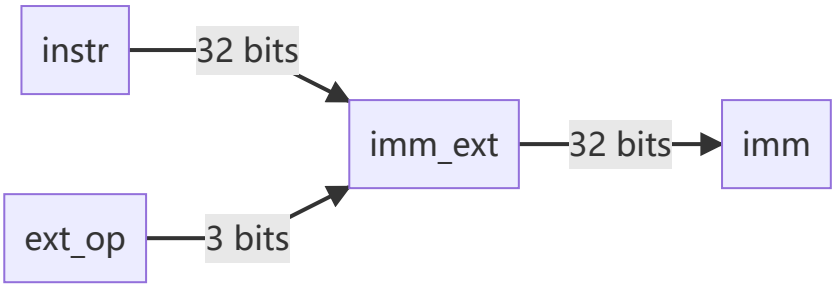


下一条指令地址计算器，用于计算下一条指令的地址并输出到 `next_pc` 端口。

输入/输出端口	方向	位宽	描述
ce	输入	1	时钟使能信号
pc	输入	32	当前指令地址

输入/输出端口	方向	位宽	描述
imm	输入	32	立即数
branch	输入	1	分支条件信号
zero	输入	1	零标志信号
jump	输入	1	跳转信号
next_pc	输出	32	下一条指令地址

4.2.6 立即数扩展器



立即数扩展器，用于将指令中的立即数进行扩展，并将结果输出到 `imm` 端口。

ext_op	扩展操作
3'b000	符号扩展20位立即数
3'b001	符号扩展20位无符号立即数
3'b010	零扩展5位立即数并符号扩展20位立即数
3'b011	零扩展1位立即数并符号扩展6位立即数，并将其组合成32位立即数
3'b100	符号扩展12位立即数，并将其组合成32位立即数

4.3 确定控制信号

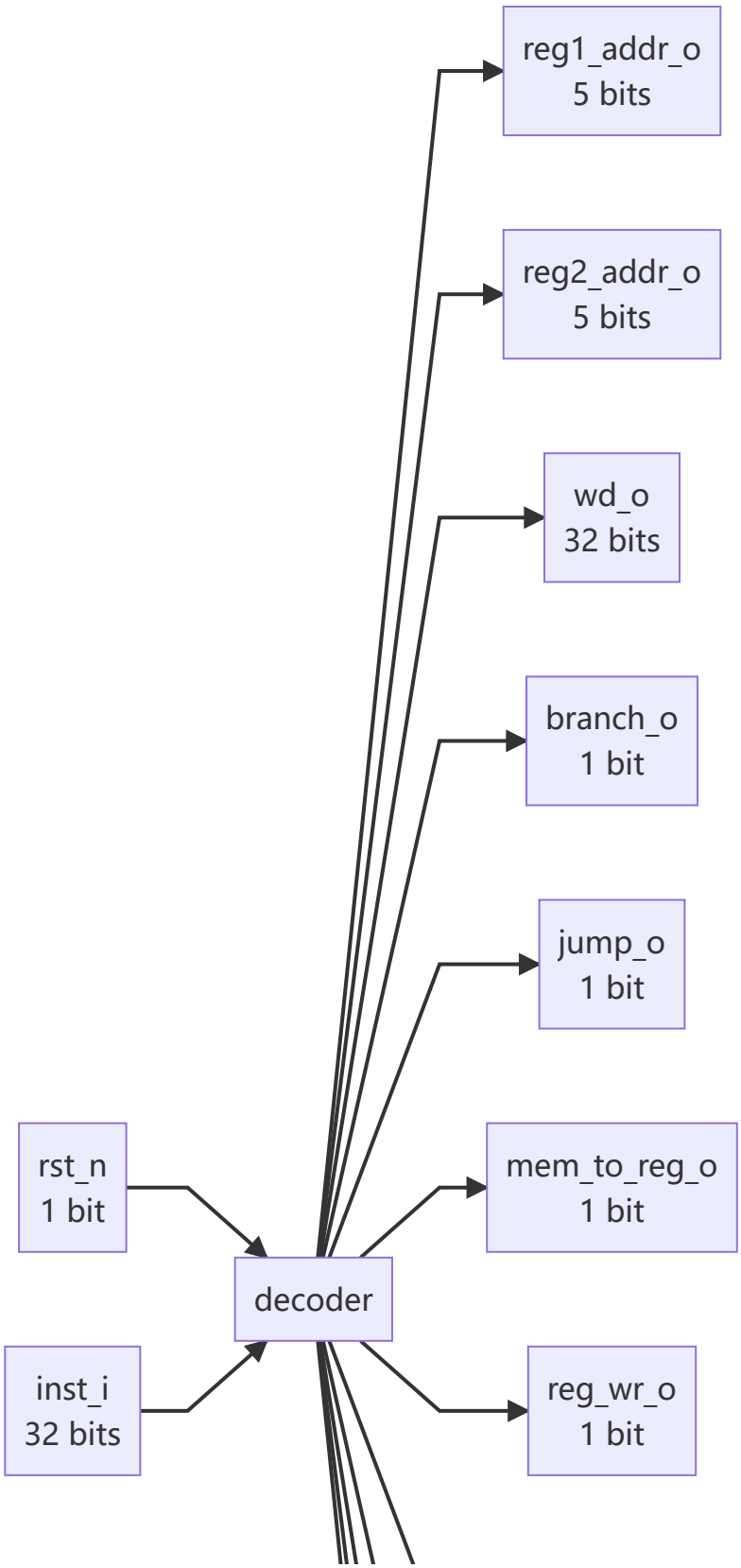
根据数据通路设计结果，填充并扩展表格

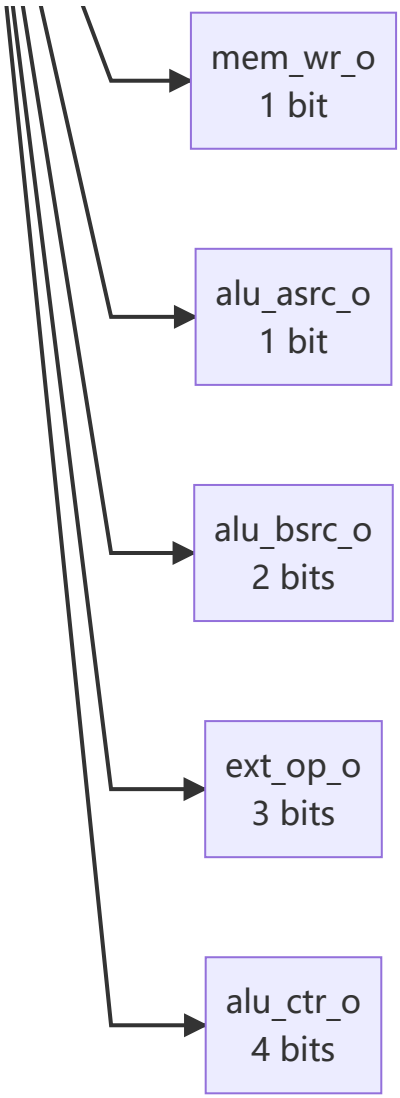
指令	C1	C2	C3	alu_op	branch	dmem_we	reg_we
ADD	0	1	10	001	00	0	1
SUB	0	1	10	010	00	0	1
ORI	0	0	10	100	00	0	1
LUI	0	0	00	000	00	0	1
SW	1	0	00	001	00	1	0
LW	0	0	11	001	00	0	1
BEQ	0	0	00	000	01	0	0

指令	C1	C2	C3	alu_op	branch	dmem_we	reg_we
JAL	0	0	01	000	10	0	1

4.4 设计控制单元

4.4.1 指令译码器



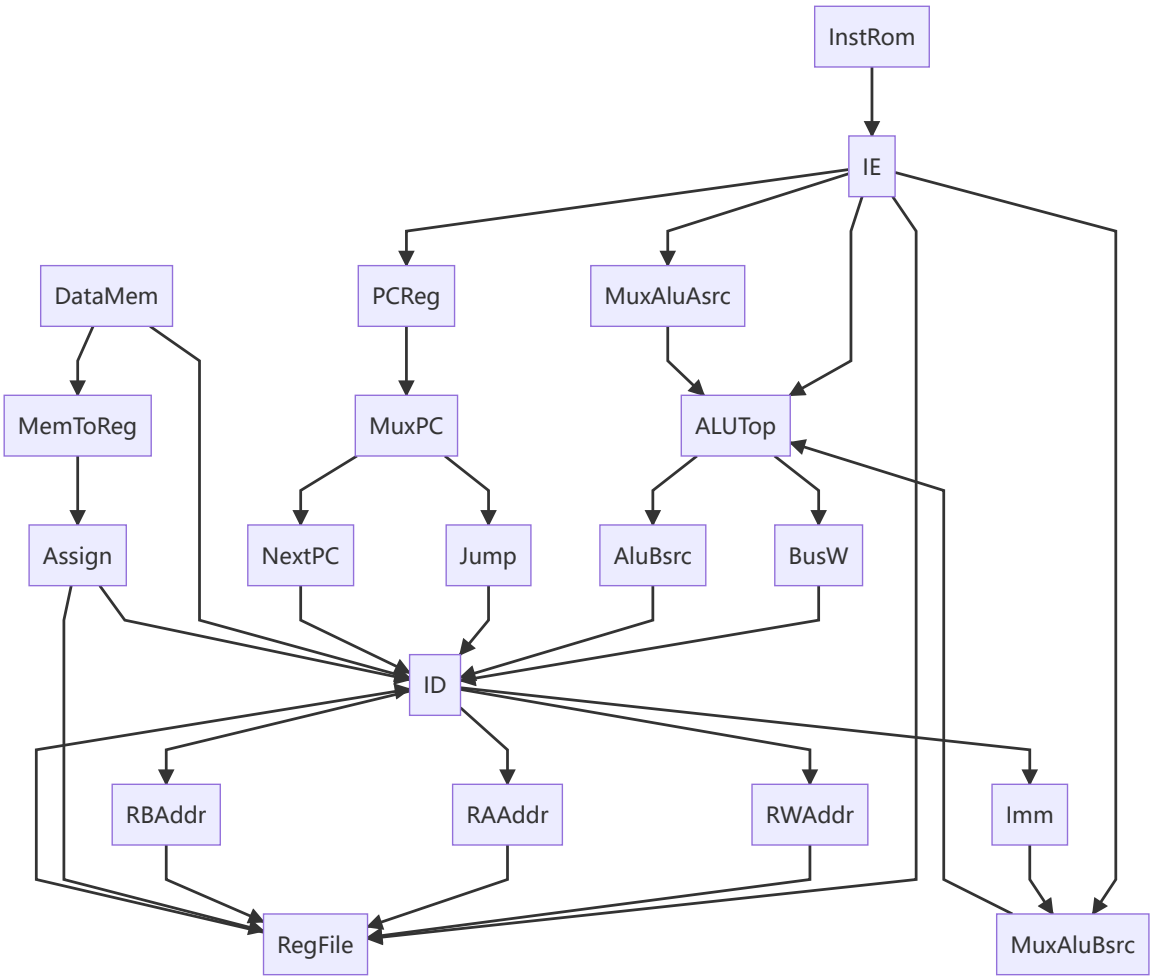


指令译码器，用于解析指令并产生控制信号。该模块有两个输入和十一个输出端口，分别是复位信号 `rst_n`、输入指令 `inst_i`，以及输出控制信号 `reg1_addr_o`、`reg2_addr_o`、`wd_o`、`branch_o`、`jump_o`、`mem_to_reg_o`、`reg_wr_o`、`mem_wr_o`、`alu_asrc_o`、`alu_bsrc_o`、`ext_op_o` 和 `alu_ctr_o`。

输入/输出端口	方向	位宽	描述
rst_n	输入	1	复位信号
inst_i	输入	32	输入指令
reg1_addr_o	输出	5	读写寄存器 1 的地址
reg2_addr_o	输出	5	读写寄存器 2 的地址
wd_o	输出	32	写入寄存器的数据
branch_o	输出	1	分支操作信号
jump_o	输出	1	跳转操作信号
mem_to_reg_o	输出	1	内存读操作信号
reg_wr_o	输出	1	寄存器写操作信号

输入/输出端口	方向	位宽	描述
mem_wr_o	输出	1	内存写操作信号
alu_asrc_o	输出	1	ALU 操作数 A 的来源信号
alu_bsrc_o	输出	2	ALU 操作数 B 的来源信号
ext_op_o	输出	3	立即数扩展方式
alu_ctr_o	输出	4	ALU 控制信号

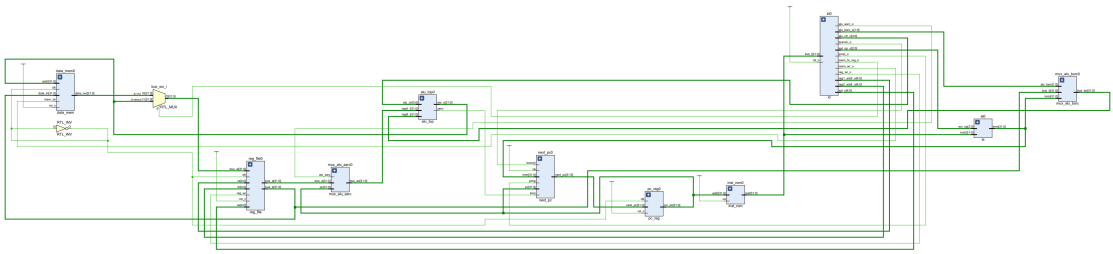
4.5 CPU数据通路的整合



基本的 CPU 架构，包含了许多常见的模块和电路，例如寄存器文件、ALU、指令解码、数据存储器和 PC 寄存器等。

输入信号	描述
clk	时钟信号
rst_n	复位信号
inst	指令
cur_pc	当前 PC 地址
next_pc	下一个 PC 地址
rw_addr	读写寄存器地址
ra_addr	寄存器 A 的地址
rb_addr	寄存器 B 的地址
imm	立即数
mem_wr	总线读写信号
reg_wr	寄存器写使能信号
alu_asrc	ALU 乘数来源信号
alu_ctr	ALU 操作码

输出信号	描述
data_out	总线读数据
bus_w	ALU 的结果
jump	是否进行跳转
next_pc	下一个 PC 地址
reg_wr	是否进行寄存器写入
mem_wr	是否进行内存写入
mem_to_reg	是否从内存中读出数据并写入寄存器
alu_bsrc	ALU 第二个操作数来源信号



5. 实验测试

RISC-V汇编源代码

```
.global _start
.data
    data_rom: .word 1, 5, 3, 4, 2
    inst_rom: .word 0x00000013, 0x00100513, 0x00008067, 0x00118593, 0x00b50533, 0xfe010113, 0x00a50523, 0x0000006f

.text
_start:
    # 初始化寄存器
    lui t0, %hi(data_rom)       # 将data_rom的高20位加载到寄存器t0
    addi t0, t0, %lo(data_rom)  # 将data_rom的低12位加到寄存器t0中
    lui t1, %hi(inst_rom)       # 将inst_rom的高20位加载到寄存器t1
    addi t1, t1, %lo(inst_rom)  # 将inst_rom的低12位加到寄存器t1中
    li t2, 5                    # 将数字5加载到寄存器t2

    # 排序
    loop:
        lw a0, 0(t0)            # 从t0地址处读取一个整数到寄存器a0
        lw a1, 4(t0)            # 从t0+4地址处读取一个整数到寄存器a1
        blt a0, a1, swap        # 如果a0小于a1，则跳转到swap标签
        nop                      # 否则不交换，继续执行
        j next                   # 跳转到next标签
        nop

    swap:
        sw a1, 0(t0)            # 将a1存储到t0地址处
        sw a0, 4(t0)            # 将a0存储到t0+4地址处
        j next                   # 跳转到next标签
        nop

    next:
        addi t0, t0, 8            # 将t0加上8，即跳到下一个整数的地址处
        lw a1, 0(t1)            # 从t1地址处读取排序后的整数个数到寄存器a1中
        bne a1, t2, loop        # 如果a1不等于5，则跳转到loop标签
        nop
        j exit                   # 否则跳转到exit标签
        nop

```

```
38      # 无限循环
39      exit:
40          j exit                  # 无限循环
41          nop
```

具体指令	funct7(31-25)	rs2(24-20)	rs1(19-15)	funct3(14-12)	Rd(11-7)	Op(6-0)
lui x5, 0x00010010	0000000	00000	00000	011	00101	0110111
addi x5, x5, 0	0000000	00000	00101	000	00101	0010011
lui x6, 0x00010010	0000000	00000	00000	011	00110	0110111
addi x6, x6, 20	0000000	00000	00110	000	00110	0010011
addi x7,x0, 5	0000000	00000	00000	000	00111	0010011
lw x10,0(x5)	0000000	00101	00000	010	01010	0000011
lw x11,4(x5)	0000000	00101	00000	010	01011	0000011
blt x10, x11, 0x00000010	0000000	01011	01010	100	00000	1100011
addi x0, x0,0	0000000	00000	00000	000	00000	0010011
jal x0, 0x00000018	0000000	00000	00000	001	00000	1101111
addi x0,x0,0	0000000	00000	00000	000	00000	0010011
sw x11,0(x5)	0000000	00101	00000	010	01011	0100011
sw x10,4(x5)	0000000	00101	01010	010	01010	0100011
jal x0, 0x00000008	0000000	00000	00000	001	00000	1101111
addi x0,x0,0	0000000	00000	00000	000	00000	0010011
addi x5, x5,8	0000000	00000	00101	000	00101	0010011
lw x11,0(x6)	0000000	00000	00110	010	01011	0000011
bne xll, x7, 0xfffffd0	0000000	00111	00110	001	00000	1100011
addi x0,x0,0	0000000	00000	00000	000	00000	0010011
jal x0,0x00000008	0000000	00000	00000	001	00000	1101111
addi x0,x0,0	0000000	00000	00000	000	00000	0010011

jal x0,0x00000000	0000000	00000	00000	001	00000	1101111

addi x0,x0,0	0000000	00000	00000	00000	00000	000
	funct7(31-	rs2(24-	rs1(19-	funct3(14-	Rd(11-	

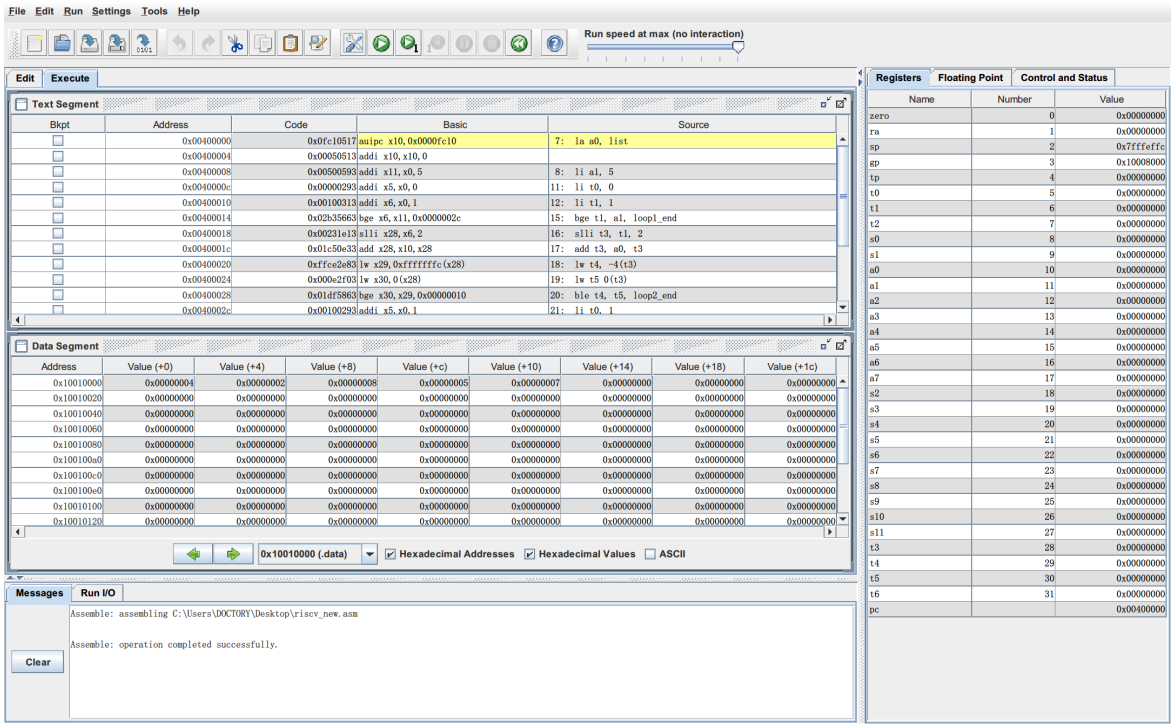
机器码

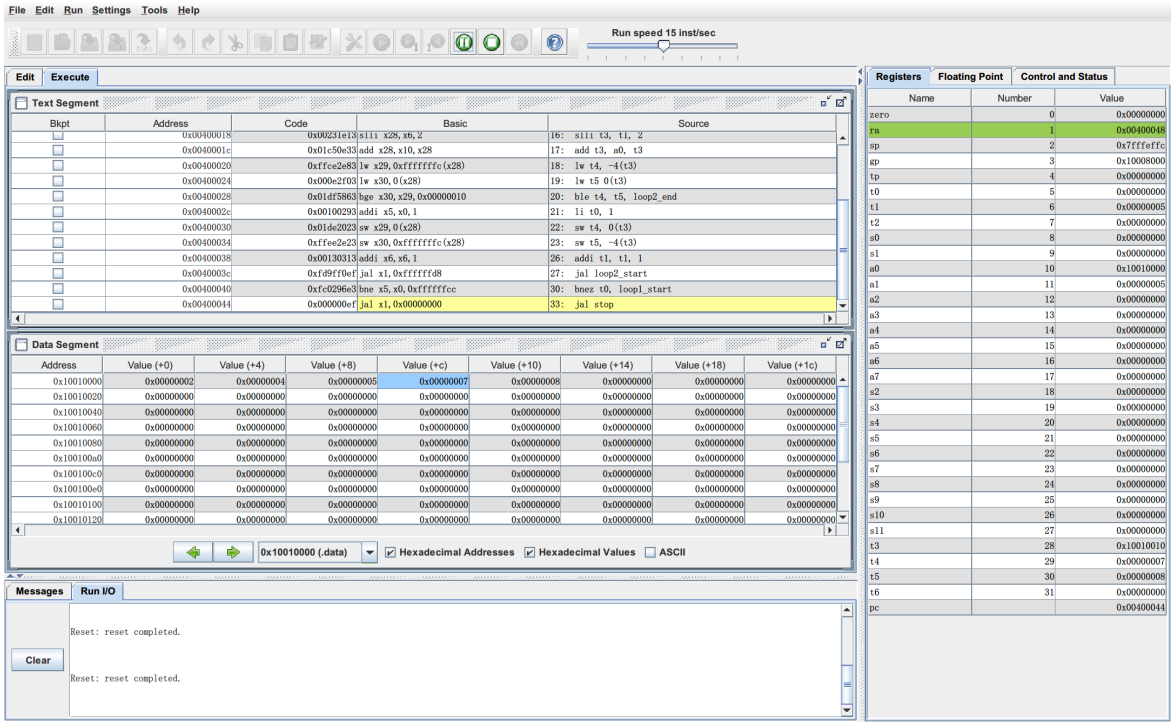
指令存储器：

```
100102b7
00028293
10010337
01430313
00500393
0002a503
0042a583
00b54863
00000013
0180006f
00000013
00b2a023
00a2a223
0080006f
00000013
00828293
00032583
fc7598e3
00000013
0080006f
00000013
0000006f
00000013
```

数据存储器：

```
00000001
00000005
00000003
00000004
00000002
```





6. 实验心得

这次实验让我学到了很多关于计算机体系结构、汇编语言、硬件描述语言和EDA工具方面的知识。在实验中，我掌握了RISCV指令格式与编码，学会了如何将RISCV汇编转换成机器代码，并且设计并实现了单周期处理器，支持包含LW、SW、ADD、SUB、ORI和BEQ等指令。我也学会了基本的测试与调试方法，编写了测试程序来验证CPU的正确性。

在实验中，我遇到了一些问题，比如在设计CPU的时候不知道如何连接各个模块的接口，以及在测试程序时出现了一些错误。但是，通过查找资料和向老师请教，我最终成功解决了这些问题，加深了自己对计算机系统的理解，也提高了自己的实践能力和解决问题的能力。

通过这次实验，我不仅掌握了硬件设计的基础知识和工具使用技巧，还加深了对计算机体系结构的理解，提高了自己的实践能力和解决问题的能力。我相信这些知识和技能对我的未来学习和工作都会有很大的帮助。