哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

课程报告

开课学期：2025 夏季

课程名称：计算机设计与实践

项目名称：基于 miniRV 的 SoC 设计

项目类型：综合设计型

课程学时：48 地点：T2507

学生班级：

学生学号：

学生姓名：

评阅教师：

报告成绩：

实验与创新实践教育中心制

2025 年 6 月

注：本设计报告中各个部分如果页数不够，请同学们自行扩页。原则上一定要把报告写详细，能说明设计的成果、特色和过程。报告应该详细叙述整体设计，以及设计中的每个模块。设计报告将是评定每个人成绩的重要组成部分（**设计内容及报告写作**都作为评分依据）。

设计概述（罗列出所有实现的指令，以及单周期/流水线 CPU 频率）

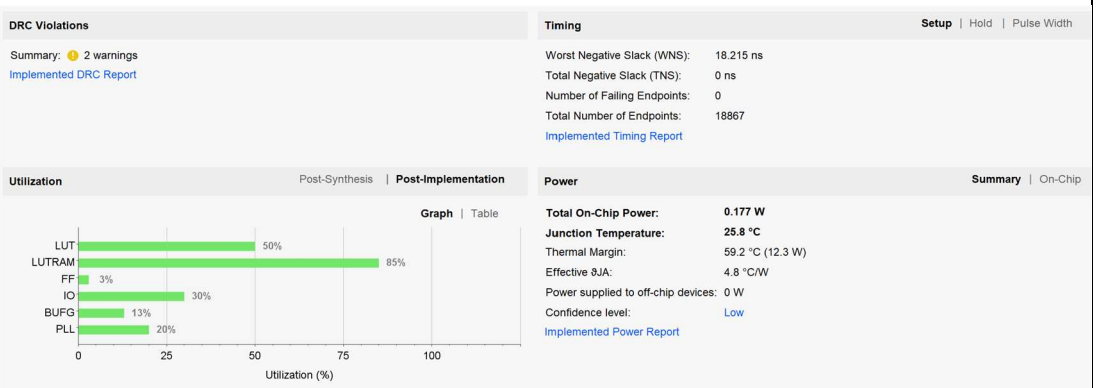
实现指令：单周期与流水线 CPU 均实现了 24 条指令：add，sub，and，or，xor，sll，srl，sra，addi，andi，ori，xori，slli，srli，srai，lw，jalr，sw，beq，bne，bge，blt，lui，jal。
时钟频率：单周期 CPU 频率达 25MHz，流水线 CPU 频率达 100MHz。

设计的主要特色（除基本要求以外的设计）

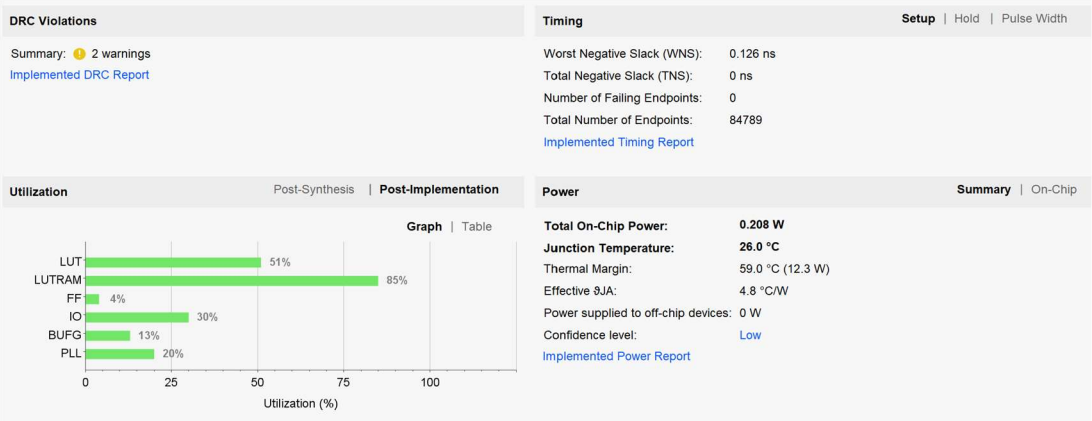
1. **实现高时钟频率理想流水线 CPU：**
   在原有单周期 CPU 的基础上，通过添加流水线寄存器、调试优化数据通路和控制信号逻辑，成功将 CPU 时钟频率从 25MHz 提高至 100MHz。
2. **实现必做指令和选做指令：**
   单周期与流水线 CPU 在实现 21 条必做指令的基础上，还额外实现了 3 条选做指令：sub，bne，bge，使得编写汇编程序时具有更高的自由度。
3. **精简模块以降低耦合度：**
   在流水线 CPU 中抽离出跳转逻辑，单独设计 JUMP_DETECT 模块，简化了 NPC 模块的输入端口信号，在节省了硬件资源的同时降低了模块间的耦合度。

资源使用、功耗数据截图（Post Implementation；含单周期、流水线 2 个截图）

单周期：



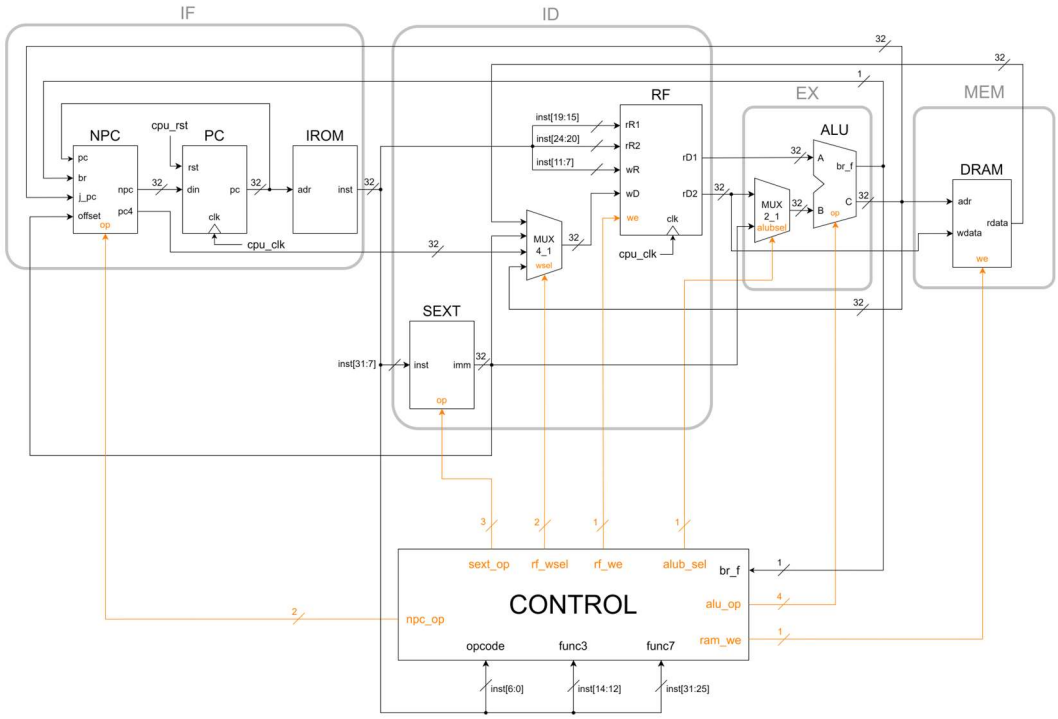
流水线：



1 单周期 CPU 设计与实现

1.1 单周期 CPU 数据通路设计

要求：贴出完整的单周期数据通路图，无需画出模块内的具体逻辑，但要标出模块的接口信号名、模块之间信号线的信号名和位宽，并用文字阐述各模块的功能。



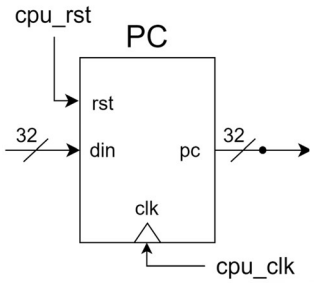
模块功能：

- PC：程序计数器，用于存储当前指令的地址，在指令执行过程中，pc 值会根据时序逻辑自动更新。
- NPC：下一程序计数器，通过组合逻辑生成下一条指令的 pc 值。
- IROM：指令存储器，用于存储程序的指令代码。它根据 pc 值提供的地址输出对应的 32 位指令。
- RF：寄存器堆，包含 32 个 32 位通用寄存器，其中 x0 寄存器硬件置零。可通过寄存器编号异步读取源寄存器数据（组合逻辑）或同步写入目标寄存器（时序逻辑）。
- SEXT：立即数扩展单元，根据指令编码中的立即数格式（I/S/B/U/J 型）进行符号扩展或零扩展，生成 32 位立即数。
- ALU：算术逻辑单元，根据控制信号对操作数执行算术运算（加/减/比较）、逻辑运算（与/或/异或）、移位操作（逻辑/算术移位）等运算，并输出运算结果及跳转标志信号。
- DRAM：数据存储器，可进行读写操作，用于存储数据。
- CONTROL：控制单元，根据 opcode、func3、func7 以及来自 ALU 的跳转标志生成各个模块的控制信号。

1.2 单周期 CPU 模块详细设计

要求：以表格的形式列出各个部件的接口信号、位宽、功能描述等，并结合图、表、核心代码等形象化工具和手段，详细描述各个部件的关键实现。

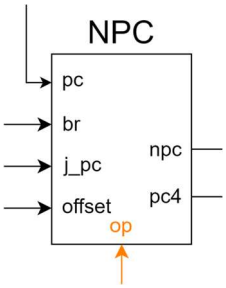
- **PC**



复位信号有效时，pc 寄存器初始化为 0；正常执行指令时，pc 寄存器在每个时钟周期更新为下一条指令地址。

信号名称	信号类型	位宽	功能描述
clk	input	1	时钟信号
rst	input	1	复位信号
din	input	32	下一条指令地址
pc	output	32	当前指令地址

- **NPC**



（1） pc4 始终输出 pc + 4 的值

（2） npc 根据控制信号 op 输出下一条指令地址：

- jal：npc = pc + offset
- jalr：npc = j_pc
- B 型指令：npc = br ? pc + offset : pc + 4
- 其他：npc = pc + 4

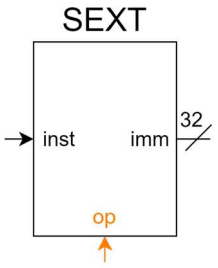
信号名称	信号类型	位宽	功能描述
pc	input	32	当前指令地址
br	input	1	分支指令跳转标志
j_pc	input	32	jalr 指令跳转地址
op	input	2	控制信号：控制 npc 输出方式
offset	input	32	跳转偏移量
npc	output	32	下一条指令地址
pc4	output	32	当前指令地址 + 4

- **IROM**

指令存储器，根据 PC 提供的指令地址输出对应的 32 位指令。

信号名称	信号类型	位宽	功能描述
adr	input	32	指令地址
inst	output	32	指令

- **SEXT**

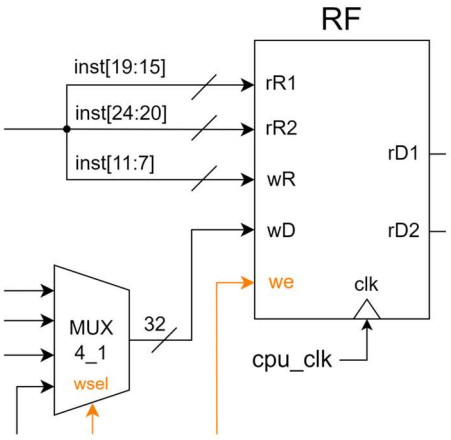


根据控制信号 op 以不同方式对输入的指令立即数进行扩展。

信号名称	信号类型	位宽	功能描述
op	input	3	控制信号：选择扩展方式
inst	input	25	指令立即数部分 inst[31:7]
imm	output	32	扩展后的立即数

```
always @(*) begin
    case (op_i)
        `SEXT_OP_B: imm_o = inst_i[24]? {20'hFFFFF,inst_i[0],inst_i[23:18],inst_i[4:1],1'b0} : {20'h00000,inst_i[0],inst_i[23:18],inst_i[4:1],1'b0};
        `SEXT_OP_I: imm_o = inst_i[24]? {20'hFFFFF,inst_i[24:13]} : {20'h00000,inst_i[24:13]};
        `SEXT_OP_SHIFT: imm_o = {27'd0,inst_i[17:13]};
        `SEXT_OP_U: imm_o = {inst_i[24:5],12'h000};
        `SEXT_OP_J: imm_o = inst_i[24]? {12'hFFF,inst_i[12:5],inst_i[13],inst_i[23:14],1'b0} : {12'h000,inst_i[12:5],inst_i[13],inst_i[23:14],1'b0};
        `SEXT_OP_S: imm_o = inst_i[24]? {20'hFFFFF,inst_i[24:18],inst_i[4:0]} : {20'h00000,inst_i[24:18],inst_i[4:0]};
        default: imm_o =32'h00000000;
    endcase
end
```

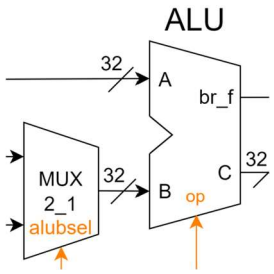
- **RF**



复位信号有效时，寄存器数组中数据均置零。读数据时，根据读地址 rR1、rR2 通过组合逻辑读出寄存器数组中的数据 rf[rR1]、rf[rR2]，并通过 rD1、rD2 输出。写数据时，若写使能 we 有效且写地址 wR 不为 0，则将写数据 wD 通过时序逻辑写入目标寄存器；若 wR 为 0，则写入无效，x0 寄存器中数据始终保持为 0。内部信号 wD 由多路选择器选择输入数据，选择信号为 rf_wsel，待选数据为存储器读数据 dram、ALU 运算结果 aluc、当前指令地址加四 pc4、扩展后立即数 imm。

信号名称	信号类型	位宽	功能描述
clk	input	1	时钟信号
rst	input	1	复位信号
rR1	input	5	读地址 1
rR2	input	5	读地址 2
we	input	1	写使能
wR	input	5	写寄存器
aluc	input	32	ALU 运算结果
dram	input	32	存储器读数据
imm	input	32	SEXT 扩展后立即数
pc4	input	32	当前指令地址 +4
rf_wsel	input	2	多路选择信号：选择写数据来源
rD1	output	32	读数据 1
rD2	output	32	读数据 2

- **ALU**



根据控制信号 op 选择运算方式，对操作数 A、B 进行相应运算，输出运算结果 C 和跳转标志信号 br_f。其中操作数 B 由多路选择器选择输入数据，选择信号为 alub_sel，待选数据为 RF 读数据 rD2 和 SEXT 扩展后立即数 imm。

信号名称	信号类型	位宽	功能描述
A	input	32	操作数 A
B	input	32	操作数 B：与 RF.rD2 连接
imm	input	32	操作数 B：SEXT 扩展后立即数
op	input	4	控制信号：选择运算方式
alub_sel	input	1	多路选择信号：选择操作数 B 数据来源
br_f	output	1	分支指令跳转标志
C	output	32	运算结果

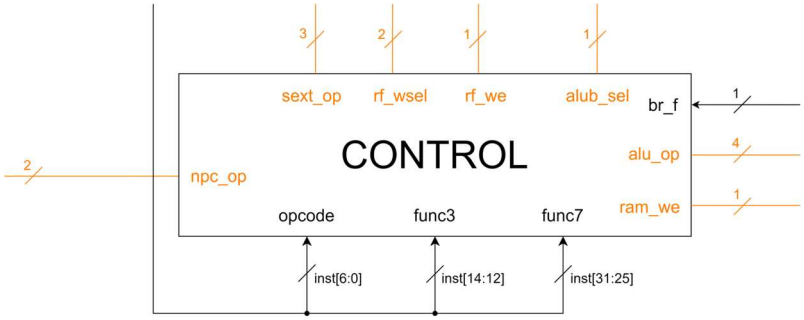
```
wire [4:0]  shamt = B[4:0];
wire [31:0] A = A_i;
wire [31:0] B = alub_sel_i ? imm_i : B_i;

always @(*) begin
    case (op_i)
        `ALU_OP_ADD: C_o = A + B;
        `ALU_OP_SUB: C_o = A + (~B) + 1;
        `ALU_OP_AND: C_o = A & B;
        `ALU_OP_OR:  C_o = A | B;
        `ALU_OP_XOR: C_o = A ^ B;
        `ALU_OP_SRA: C_o = $signed(A) >>> shamt;
        `ALU_OP_SLL: C_o = A << shamt;
        `ALU_OP_SRL: C_o = A >> shamt;
        `ALU_OP_BNE: br_f_o = (A != B) ? 1'b1 : 1'b0;
        `ALU_OP_BEQ: br_f_o = (A == B) ? 1'b1 : 1'b0;
        `ALU_OP_BGE: begin
            C_o = A + (~B) + 1;
            br_f_o = (~C_o[31]) ? 1'b1 : 1'b0;
        end
        `ALU_OP_BLT: begin
            C_o = A + (~B) + 1;
            br_f_o = (C_o[31]) ? 1'b1 : 1'b0;
        end
        default: C_o = 32'd0;
    endcase
end
```

- **DRAM**

数据存储器，可进行读写操作，用于存储数据。

信号名称	信号类型	位宽	功能描述
clk	input	1	时钟信号
adr	input	32	访存地址
wdata	input	32	写数据
we	input	1	写使能
rdata	output	32	读数据

- **CONTROL**



根据由 32 位指令 inst 得到的 opcode、func3、func7 以及分支指令跳转标志 br_f，生成各个模块的控制信号。

信号名称	信号类型	位宽	功能描述
inst	input	32	指令
br_f	input	1	分支指令跳转标志
rf_we	output	1	控制 RF 写使能
alu_op	output	4	控制 ALU 运算方式
ram_we	output	1	控制 DRAM 写使能
sext_op	output	3	控制 SEXT 扩展方式
rf_wsel	output	2	选择 RF 写数据来源
npc_op	output	2	控制 NPC 输出指令地址方式
alub_sel	output	1	选择 ALU 操作数 B 数据来源

1.3 单周期 CPU 仿真及结果分析

要求：包含**逻辑运算**、**访存**、**分支跳转**三类指令的仿真截图及波形分析；每类指令的截图和分析中，至少包含 1 条具体指令；截图需包含信号名和关键信号。

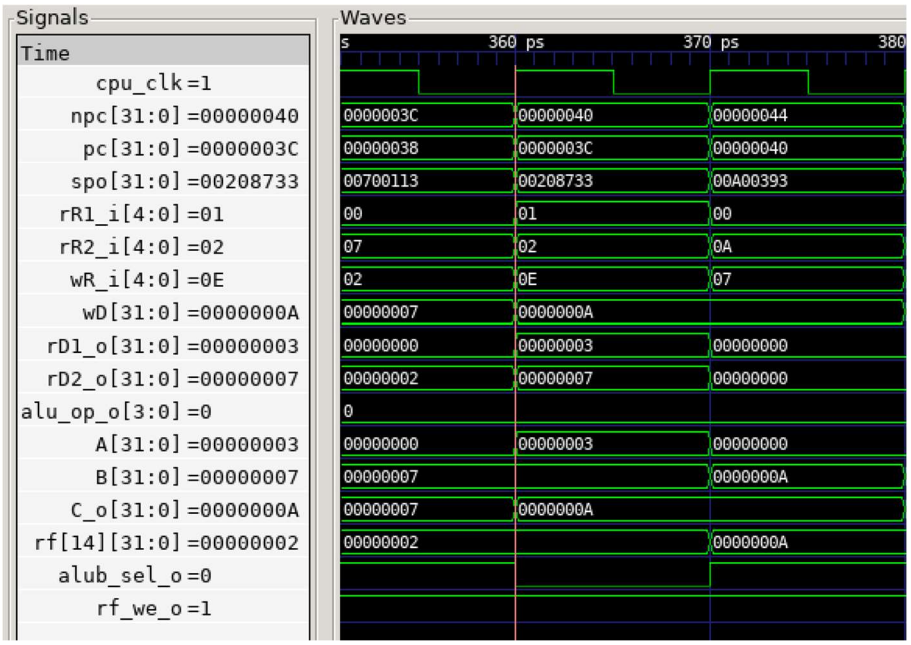
1. 逻辑运算

- add 指令：

（1） 测试用例

```
00000034 <test_4>:
  34: 00300093            addi   x1,x0,3
  38: 00700113            addi   x2,x0,7
  3c: 00208733            add    x14,x1,x2
  40: 00a00393            addi   x7,x0,10
  44: 00400193            addi   x3,x0,4
  48: 48771e63            bne    x14,x7,4e4 <fail>
```

（2） 仿真截图



（3） 波形分析

由波形可知，当 pc = 0000003C 时，CPU 从 IROM 中取出指令 00208733，npc 输出下一条指令的地址 00000040，显然为顺序执行。指令拆分后可得 rR1= 01（x1），rR2 = 02（x2），wR = 0E（x14），传给 RF 后取出寄存器数据 rD1 = 00000003，rD2 = 00000007。此时 alu_op 和 alub_sel 均为 0，ALU 执行加法运算且操作数 A 来自 rD1、操作数 B 来自 rD2，输出 A + B 的运算结果 C = 0000000A，并将该结果传至 wD。此时 rf_we 有效，RF 在下一时钟周期将 wD 写入 x14 寄存器。

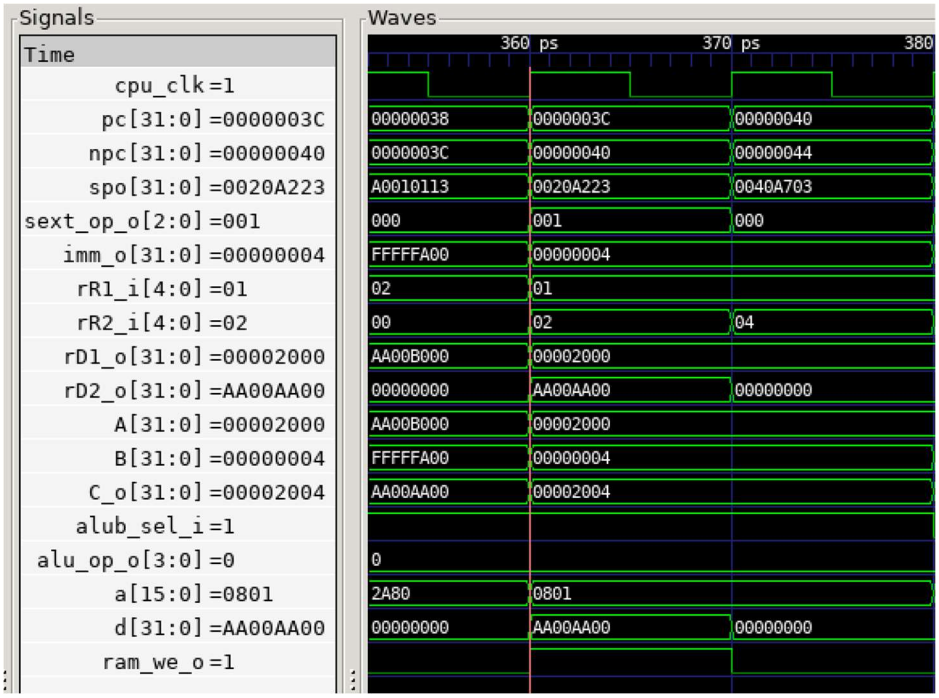
2. 访存

- sw 指令：

（1） 测试用例

```
0000002c <test_3>:
  2c: 000020b7            lui   x1,0x2
  30: 00008093            addi  x1,x1,0 # 2000 <begin_signature>
  34: aa00b137            lui   x2,0xaa00b
  38: a0010113            addi  x2,x2,-1536 # aa00aa00 <_end+0xaa0089d0>
  3c: 0020a223            sw    x2,4(x1)
  40: 0040a703            lw    x14,4(x1)
  44: aa00b3b7            lui   x7,0xaa00b
  48: a0038393            addi  x7,x7,-1536 # aa00aa00 <_end+0xaa0089d0>
  4c: 00300193            addi  x3,x0,3
  50: 42771c63            bne   x14,x7,488 <fail>
```

（2） 仿真截图



（3） 波形分析

由波形可知，当 pc = 0000003C 时，CPU 从 IROM 中取出指令 0020A223，npc 输出下一条指令的地址 00000040，为顺序执行。指令拆分后可得 rR1 = 01（x1），rR2 = 02（x2），传给 RF 后取出寄存器数据 rD1 = 00002000，rD2 = AA00AA00。此时 sext_op = 001，SEXT 以 S 型扩展方式输出立即数 imm = 00000004。此时 alu_op = 0，alub_sel = 1，ALU 执行加法运算且操作数 A 来自 rD1、操作数 B 来自 imm，输出 A + B 的运算结果 C = 00002004，即访存地址。地址 C 和写数据 rD2 传至 DRAM，此时 DRAM 写使能 ram_we = 1，则将 rD2 = AA00AA00 存储至 DRAM 的地址 C[17:2] = 0801 处。

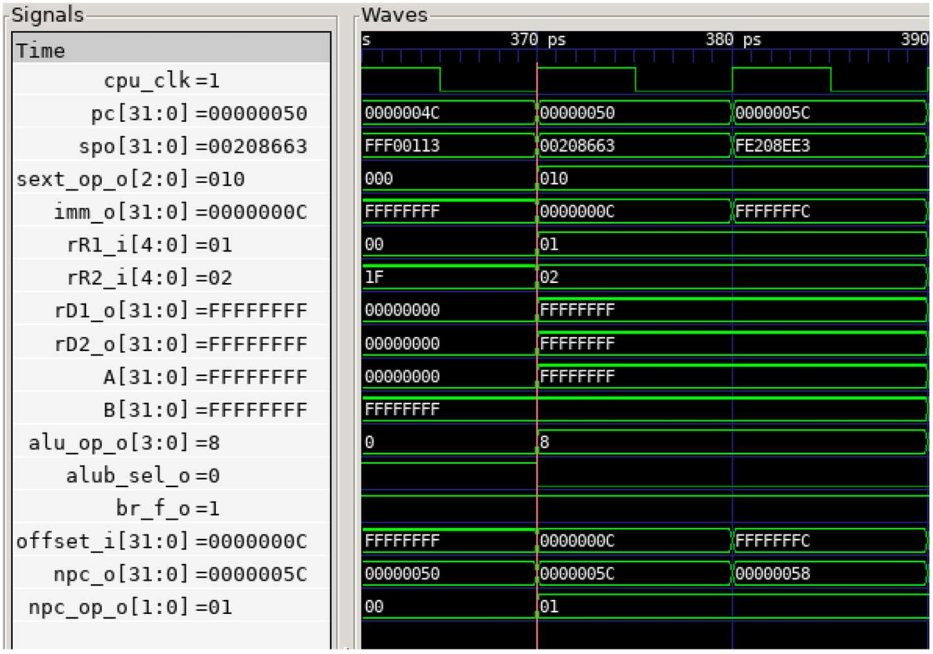
3. 分支跳转

- beq 指令：

（1） 测试用例

```
00000044 <test_4>:
  44: 00400193            addi   x3,x0,4
  48: fff00093            addi   x1,x0,-1
  4c: fff00113            addi   x2,x0,-1
  50: 00208663            beq    x1,x2,5c <test_4+0x18>
  54: 26301863            bne    x0,x3,2c4 <fail>
  58: 00301663            bne    x0,x3,64 <test_5>
  5c: fe208ee3            beq    x1,x2,58 <test_4+0x14>
  60: 26301263            bne    x0,x3,2c4 <fail>
```

（2） 仿真截图



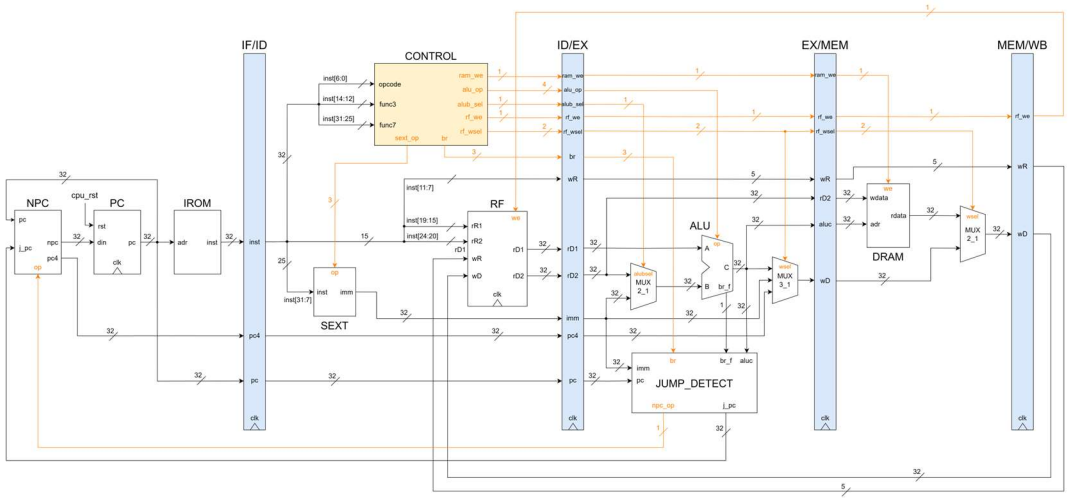
（3） 波形分析

由波形可知，当 pc = 00000050 时，CPU 从 IROM 中取出指令 00208663。指令拆分后可得 rR1 = 01（x1），rR2 = 02（x2），传给 RF 后取出寄存器数据 rD1 = FFFFFFFF，rD2 = FFFFFFFF。此时 sext_op = 010，SEXT 以 B 型扩展方式输出立即数 imm = 0000000C，即跳转偏移量。此时 alu_op = 8，alub_sel = 0，ALU 执行比较运算且操作数 A 来自 rD1、操作数 B 来自 rD2，判断 A == B 得到分支指令跳转标志 br_f = 1，即需要跳转。该标志信号分别传至 NPC 和 CONTROL，CONTROL 生成控制信号 npc_op = 01，控制 NPC 以 B 型方式输出 npc，即 npc = pc + offset = 0000005C，故 pc 在下一时钟周期更新为 0000005C，成功实现跳转。

2 流水线 CPU 设计与实现

2.1 流水线 CPU 数据通路

要求：贴出完整的流水线数据通路图，无需画出模块内的具体逻辑，但要标出模块的接口信号名、模块之间信号线的信号名和位宽，并用文字阐述各模块的功能。
此外，数据通路图应当能体现出流水线是如何划分的，并用文字阐述每个流水级具备什么功能、需要完成哪些操作。



模块功能：

- PC：程序计数器，用于存储当前指令的地址，在指令执行过程中，pc 值会根据时序逻辑自动更新。
- NPC：下一程序计数器，通过组合逻辑生成下一条指令的 pc 值。
- IROM：指令存储器，用于存储程序的指令代码。它根据 pc 值提供的地址输出对应的 32 位指令。
- RF：寄存器堆，包含 32 个 32 位通用寄存器，其中 x0 寄存器硬件置零。可通过寄存器编号异步读取源寄存器数据（组合逻辑）或同步写入目标寄存器（时序逻辑）。
- SEXT：立即数扩展单元，根据指令编码中的立即数格式（I/S/B/U/J 型）进行符号扩展或零扩展，生成 32 位立即数。
- ALU：算术逻辑单元，根据控制信号对操作数执行算术运算（加/减/比较）、逻辑运算（与/或/异或）、移位操作（逻辑/算术移位）等运算，并输出运算结果及跳转标志信号。
- JUMP_DETECT：跳转控制单元，判断当前指令是否为 jal/jalr/B 型指令、是否需要跳转并计算出跳转地址赋值给 NPC。
- DRAM：数据存储器，可进行读写操作，用于存储数据。
- CONTROL：控制单元，根据 opcode、func3、func7 生成各个模块的控制信号。
- REG_IF_ID：IF 和 ID 之间的流水线寄存器。
- REG_ID_EX：ID 和 EX 之间的流水线寄存器。
- REG_EX_MEM：EX 和 MEM 之间的流水线寄存器。
- REG_MEM_WB：MEM 和 WB 之间的流水线寄存器。

各级流水线功能及操作：

（1） 取指阶段（IF）

- NPC 根据控制信号 npc_op 选择输出 npc 的数据来源并输出 pc4。
- PC 接收到 npc 后在下一时钟周期更新 pc 输出。
- IROM 根据 pc 值取出相应指令。

（2） 译码阶段（ID）

- RF 根据分解指令得到的 rR1、rR2 取出操作数 rD1、rD2。
- CONTROL 根据分解指令得到的 opcode、func3、func7 生成控制信号。
- SEXT 根据控制信号 sext_op 扩展指令中的立即数。

（3） 执行阶段（EX）

- ALU 根据控制信号 alu_op 对操作数 A、B 进行相应运算，并输出运算结果 C 和分支指令跳转标志 br_f。
- 二选一多路选择器根据控制信号 alub_sel 选择 ALU 操作数 B 的数据来源。
- JUMP_DETECT 根据控制信号 br 和 br_f 判断跳转指令类型及是否需要跳转，需要时生成控制信号 npc_op 和跳转地址 j_pc。
- 三选一多路选择器根据控制信号 rf_wsel 选择写回数据 wD 的数据来源。

（4） 访存阶段（MEM）

- DRAM 根据写使能 ram_we、写数据 wdata 和地址 adr 执行相应的访存操作。
- 二选一多路选择器根据控制信号 rf_wsel 选择写回数据 wD 的数据来源。

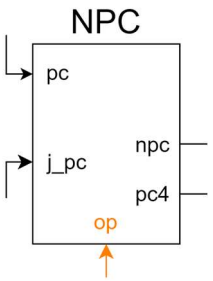
（5） 写回阶段（WB）

- 将写寄存器 wR、写数据 wD 和写使能 rf_we 传回 RF，RF 将数据写回至对应的寄存器。

2.2 流水线 CPU 模块详细设计

要求：以表格的形式列出所有**与单周期不同**的部件的接口信号、位宽、功能描述等，并结合图、表、核心代码等，详细描述这些部件的关键实现。此外，如果实现了冒险控制，必须结合数据通路图，详细说明数据冒险、控制冒险的解决方法。

- **NPC**



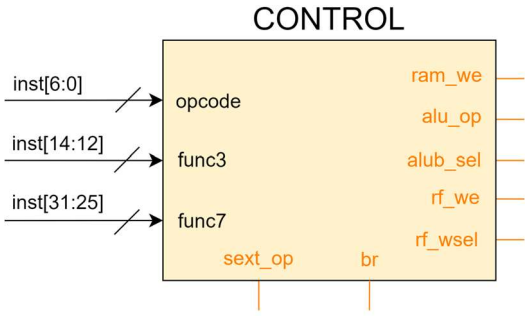
（1） pc4 始终输出 pc + 4 的值

（2） npc 根据控制信号 op 输出下一条指令地址：

- op 为 0 时，npc = pc + 4
- op 为 1 时，npc = j_pc

信号名称	信号类型	位宽	功能描述
pc	input	32	当前指令地址
j_pc	input	32	跳转地址
op	input	1	控制信号：控制 npc 输出方式
npc	output	32	下一条指令地址
pc4	output	32	当前指令地址 ＋4

- **CONTROL**

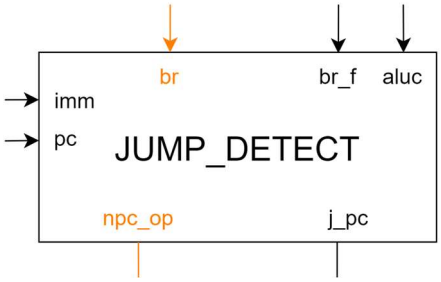


根据由 32 位指令 inst 得到的 opcode、func3、func7 生成各个模块的控制信号。

信号名称	信号类型	位宽	功能描述
inst	input	32	指令
rf_we	output	1	控制 RF 写使能
alu_op	output	4	控制 ALU 运算方式
ram_we	output	1	控制 DRAM 写使能
sext_op	output	3	控制 SEXT 扩展方式
br	output	3	跳转指令类型标志

rf_wsel	output	2	选择 RF 写数据来源
alub_sel	output	1	选择 ALU 操作数 B 数据来源

- **JUMP_DETECT**



跳转控制单元，根据跳转类型 br 和跳转标志 br_f 生成跳转控制信号 npc_op 和跳转地址 j_pc，支持三种指令：jal、jalr、B 型指令。

信号名称	信号类型	位宽	功能描述
imm	input	32	SEXT 扩展后立即数
pc	input	32	当前指令地址
br	input	3	跳转指令类型标志
br_f	input	1	跳转标志
aluc	input	32	ALU 运算结果
npc_op	output	1	跳转控制信号
j_pc	output	32	跳转目标地址

- **流水线寄存器**

寄存器名称	寄存信号
REG_IF_ID	pc，pc4，inst
REG_ID_EX	rD1，rD2，wR，imm，pc4，pc，br，rf_wsel，rf_we，alub_sel，alu_op，ram_we
REG_EX_MEM	rD2，wR，wD，aluc，rf_wsel，rf_we，ram_we
REG_MEM_WB	wR，wD，rf_we

各级流水线寄存器的实现逻辑基本一致，这里以 REG_IF_ID 中寄存 pc 信号的实现代码为例：

```
always @ (posedge clk_i or posedge rst_i) begin
    if (rst_i)    pc_o <= 32'b0;
    else          pc_o <= pc_i;
end
```

2.3 流水线 CPU 仿真及结果分析

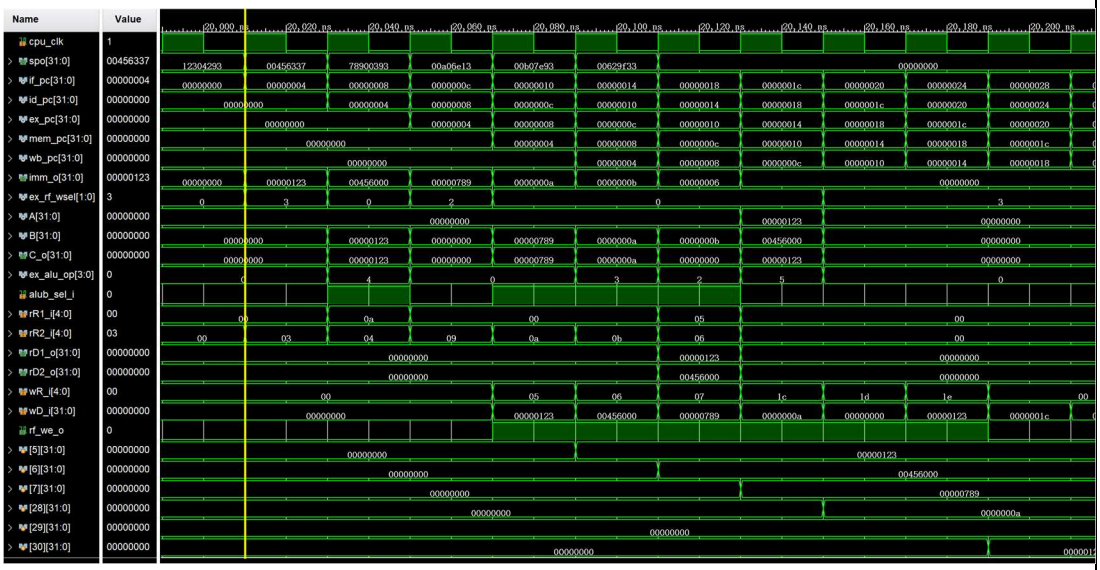
要求：包含控制冒险和数据冒险三种情形的仿真截图，以及波形分析。若仅实现了理想流水，则此处贴上理想流水的仿真截图及详细的波形分析。

理想流水线

（1） 测试用例

```
xori    t0, zero, 0x123
lui     t1, 0x456
addi    t2, zero, 0x789
ori     t3, zero, 0xA
andi    t4, zero, 0xB
sll     t5, t0, t1
```

（2） 仿真截图



（3） 波形分析

① 由波形可知，当 if_pc＝00000000 时，CPU 从 IROM 中取出指令 12304293，此时第一条指令处于 IF 阶段。

② 当 if_pc＝00000004 时，CPU 从 IROM 中取出指令 00456337，此时第二条指令处于 IF 阶段。此时 id_pc = 00000000，第一条指令处于 ID 阶段，SEXT 输出扩展后的立即数 imm＝00000123，RF 根据 rR1＝00（x0）取出 rD1＝00000000，成功译码。

③ 当 if_pc＝00000008 时，CPU 从 IROM 中取出指令 78900393，此时第三条指令处于 IF 阶段。此时 id_pc = 00000004，第二条指令处于 ID 阶段，SEXT 输出扩展后的立即数 imm＝00456000，成功译码。此时 ex_pc＝00000000，第一条指令处于 EX 阶段，alub_sel＝1，ALU 的操作数 B 选择上一阶段 imm 为数据来源，即 B＝00000123；由 alu_op＝4，A＝00000000 知 ALU 计算 C＝A ^ B＝00000123，成功执行。

④ 当 if_pc＝0000000c 时，CPU 从 IROM 中取出指令 00a06e13，此时第四条指令处于 IF 阶段。此时 id_pc = 00000008，第三条指令处于 ID 阶段，SEXT 输出扩展后的立即数 imm＝00000789，RF 根据 rR1＝00（x0）取出 rD1＝00000000，成功译码。此时 ex_pc＝00000004，第二条指令处于 EX 阶段，rf_wsel＝2，wD 选择上一阶段 imm 为数据来源，即 wD＝00456000，成功执行。此时 mem_pc＝00000000，第一条指令处于 MEM 阶段。

⑤ 当 if_pc＝00000010 时，CPU 从 IROM 中取出指令 00b07e93，此时第五条指令处于 IF 阶段。此时 id_pc = 0000000c，第四条指令处于 ID 阶段，SEXT 输出扩展后的立即

数 imm＝0000000a，RF 根据 rR1＝00（x0）取出 rD1＝00000000，成功译码。此时 ex_pc＝00000008，第三条指令处于 EX 阶段，alub_sel＝1，ALU 的操作数 B 选择上一阶段 imm 为数据来源，即 B = 00000789；由 alu_op = 0，A = 00000000 知 ALU 计算 C = A + B = 00000789，成功执行。此时 mem_pc＝00000004，第二条指令处于 MEM 阶段。此时 wb_pc = 00000000，第一条指令处于 WB 阶段，写回数据 wD＝00000123，写回寄存器 wR = 05（t0），同时写使能 rf_we＝1，RF 的 t0 寄存器 rf[5]在下一时钟周期被写为 00000123，成功写回。

⑥ 当 if_pc＝00000014 时，CPU 从 IROM 中取出指令 00629f33，此时第六条指令处于 IF 阶段。此时 id_pc = 00000010，第五条指令处于 ID 阶段，SEXT 输出扩展后的立即数 imm＝0000000b，RF 根据 rR1＝00（x0）取出 rD1＝00000000，成功译码。此时 ex_pc＝0000000c，第四条指令处于 EX 阶段，alub_sel＝1，ALU 的操作数 B 选择上一阶段 imm 为数据来源，即 B = 0000000a；由 alu_op = 3，A = 00000000 知 ALU 计算 C = A | B = 0000000a，成功执行。此时 mem_pc＝00000008，第三条指令处于 MEM 阶段。此时 wb_pc = 00000004，第二条指令处于 WB 阶段，写回数据 wD＝00456000，写回寄存器 wR = 06（t1），同时写使能 rf_we＝1，RF 的 t1 寄存器 rf[6]在下一时钟周期被写为 00456000，成功写回。

⑦ 当 id_pc＝00000014 时，第六条指令处于 ID 阶段，RF 根据 rR1＝05（t0）、rR2＝06（t1）取出 rD1＝00000123、rD2＝00456000，成功译码。此时 ex_pc＝00000010，第五条指令处于 EX 阶段，alub_sel＝1，ALU 的操作数 B 选择上一阶段 imm 为数据来源，即 B＝0000000b；由 alu_op＝2，A＝00000000 知 ALU 计算 C＝A & B＝00000000，成功执行。此时 mem_pc＝0000000c，第四条指令处于 MEM 阶段。此时 wb_pc＝00000008，第三条指令处于 WB 阶段，写回数据 wD＝00000789，写回寄存器 wR＝07（t2），同时写使能 rf_we＝1，RF 的 t2 寄存器 rf[7]在下一时钟周期被写为 00000789，成功写回。

⑧ 当 ex_pc＝00000014，第六条指令处于 EX 阶段，alub_sel＝0，ALU 的操作数 B 选择上一阶段 rD2 为数据来源，即 B＝00456000；由 alu_op＝5，A＝00000123 知 ALU 计算 C = A << B[4:0] = 00000123，成功执行。此时 mem_pc = 00000010，第五条指令处于 MEM 阶段。此时 wb_pc＝0000000c，第四条指令处于 WB 阶段，写回数据 wD＝0000000a，写回寄存器 wR = 1c（t3），同时写使能 rf_we＝1，RF 的 t3 寄存器 rf[28]在下一时钟周期被写为 0000000a，成功写回。

⑨ 当 mem_pc＝00000014，第六条指令处于 MEM 阶段。此时 wb_pc＝00000010，第五条指令处于 WB 阶段，写回数据 wD＝00000000，写回寄存器 wR＝1d（t4），同时写使能 rf_we＝1，RF 的 t4 寄存器 rf[29]在下一时钟周期被写为 00000000，成功写回。

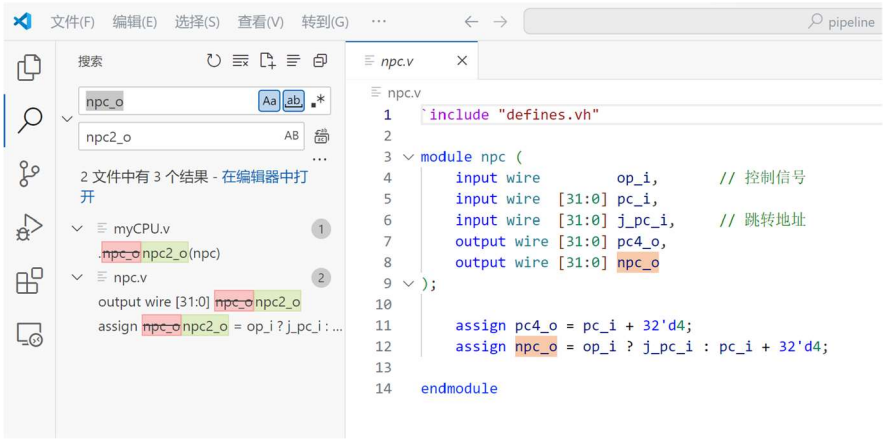
⑩ 当 wb_pc＝00000014，第六条指令处于 WB 阶段，写回数据 wD＝00000123，写回寄存器 wR = 1e（t5），同时写使能 rf_we＝1，RF 的 t5 寄存器 rf[30]在下一时钟周期被写为 00000123，成功写回。

3 设计过程中遇到的问题及解决方法

要求：包括设计过程中遇到的有价值的错误，或测试过程中遇到的有价值的问题。所谓有价值，指的是解决该错误或问题后，能够学到新的知识和技巧，或加深对已有知识的理解和运用。

1. 模块实例化及连线

在对部分模块的端口信号进行名称调整、位宽修改、增加删除等编辑后，需要及时在上层模块中同步修改对应的实例化代码和关联的 wire 型变量，否则无法成功编译。课程前期易犯此类错误且隐蔽性较强，排查难度高。此类问题的解决方法之一是在 VSCode 中编辑时通过搜索功能一键替换所有工程文件中的变量名，但此方法仅局限于修改或删除操作。根本还是要建立起对整个工程的系统性整体意识，修改代码时谨记“牵一发而动全身”，把握好模块间的层次关系，对照着数据通路图写代码是一个很好的习惯！



2. Trace 测试相关

在上传代码运行 Trace 测试前，需要对代码进行正确且全面的调整，否则会报错。首先，在 defines.vh 中取消注释 RUN_TRACE，这一点基本不易出错，这里不再赘述。其次，要将 IROM 和 DRAM 的输入地址更改为 16 位，原 14 位地址会导致比对失败。另外，单周期 Trace 时需要在 PC 模块中设置标志位在初始时停滞一个周期以保证 CPU 复位后执行的首条指令的地址为 0x00000000。最后，针对我所在班级使用的 EGO1 开发板，由于其复位信号为低电平有效，故 Trace 前一定要将 miniRV_SoC.v 中 myCPU 和 Bridge 实例化的复位信号改成高电平有效。

```
module pc (
    input wire        rst_i,
    input wire        clk_i,
    input wire [31:0] din_i,
    output reg [31:0] pc_o
);

`ifdef RUN_TRACE
    reg flag;   //设置标志位
    always @ (posedge clk_i or posedge rst_i) begin
        if(rst_i) begin
            pc_o <= 32'h00000000;
            flag = 1'b0;
        end else begin
            if (flag) begin
                pc_o <= din_i;
            end else begin
                flag = 1'b1;
            end
        end
    end
`else
```

```
    DRAM Mem_DRAM (
        .clk        (clk_bridge2dram),
`ifdef RUN_TRACE
        .a          (addr_bridge2dram[17:2]),
`else
        .a          (addr_bridge2dram[15:2]),
`endif


    myCPU Core_cpu (
`ifdef RUN_TRACE
        .cpu_rst            (fpga_rst),
`else
        .cpu_rst            (!fpga_rstn),
`endif
```

4 总结

要求：谈谈学完本课程后的个人收获以及对本课程的建议和意见。请在认真总结和思考后填写总结。

在计算机设计与实践这门课程中，我第一次从无到有地做出一个完整的项目，在这趟旅程即将结束之际，回顾这一个月，心中感慨万分。尽管在计算机组成原理的课程上已经学习了 CPU 的工作原理，但我相信大部分同学和我一样，对其中的各种运作细节了解得依然不够全面和透彻，对单周期和流水线概念的认知也仅仅停留在理论层面。然而，通过亲手设计与实践，我得以一窥其中复杂而又巧妙的结构细节，对流水线设计如何提高 CPU 运行效率也有了更加深刻的理解，这是仅靠理论学习无法获得的认知升级和宝贵经验。

亲手设计一个 CPU，初听很不可思议，做起来也确实不容易，每个阶段都充满了各种意想不到的问题。这门课的主题是“设计”和“实践”，从设计图表出发，CPU 的诞生之旅正式启程，起初我还嫌麻烦，后面才意识到这为编写代码和 debug 带来了极大的便捷。心中有图，看代码写代码也更加清晰了，特别是在梳理指令执行流程的时候。复习 RISC-V 指令集及后续基于此实现 CPU 的过程让我深刻理解了指令集架构如何在计算机硬件和软件之间架起承上启下的桥梁，同时在踩过无数坑后也是狠狠加深了对 Verilog 和硬件描述语言特性的掌握。在实现一个个模块和外设时，其实可以发现，有很多内容是曾经在数字逻辑设计和计算机组成原理的实验中涉及过的，这一刻过往的知识串联起来形成了一个系统性的框架，让我对整个 SoC 设计有了全局性的认识。

调试下板的过程确实非常痛苦，有时候寻找一个 bug 可以耗费整个下午，也许是高低电平弄反了、线接错了、位宽没对齐，或者是 Vivado 的各种环境问题…… 找不到问题源头的感觉相当折磨，对着波形调试半天终于找到 bug 时也是特别的兴奋激动。与高级程序语言相比，Verilog 调试起来还是太磨人心志了，只能像侦探一样从结果反推可能的原因，一个一个试探排除，最终锁定目标。这段日子相当有效地锻炼了我的耐心和发掘问题、解决问题的能力，好心态成就一切，浮躁粗心是大忌。

最后，感谢老师的指导，感谢自己的坚持与付出，这样一个苦乐交加的七月我想我会一直铭记。