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

课程报告

开课学期：2023 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：56 地点：T2612

学生班级：5 班

学生学号：210110503

学生姓名：陈睿玮

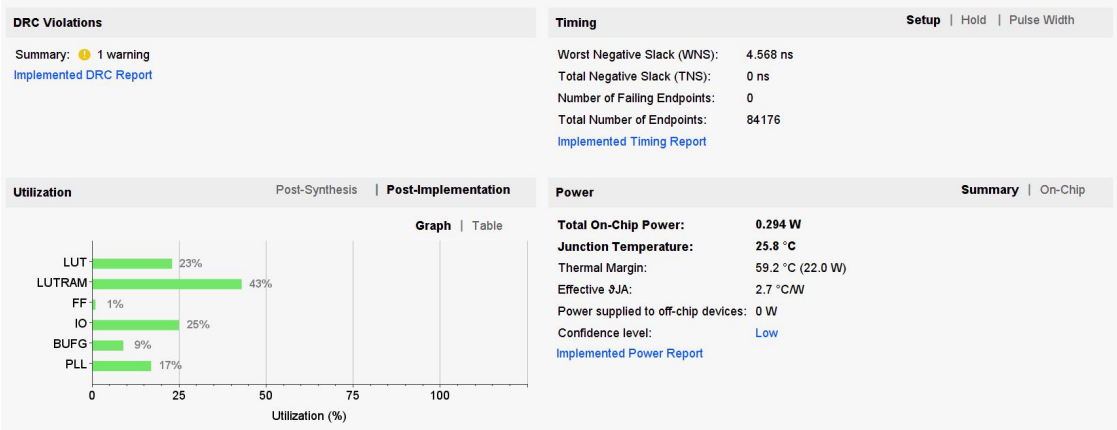
评阅教师：

报告成绩：

实验与创新实践教育中心制

2023 年 7 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）
单周期 CPU 和流水线 CPU 都实现了全部 37 条指令。 单周期 CPU 频率为 25MHz，流水线 CPU 频率为 75MHz。
设计的主要特色（除基本要求以外的设计）
实现了包括选做指令在内的全部 37 条指令。 流水线 CPU 在处理非 LOAD 类型指令产生的数据冒险时，使用前递机制，高效解决了数据冒险。在处理 LOAD 类型指令产生的 ID/EX 数据冒险时，将 IF/ID、ID/EX 寄存器暂停一个时钟周期，并在 MEM 阶段将数据前递回来。在处理控制冒险时，使用暂停法，将 IF/ID 寄存器暂停 3 个时钟周期，然后根据 ALU 传回的结果进行是否跳转的操作。
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

DRC Violations

Summary: 2 warnings

Implemented DRC Report

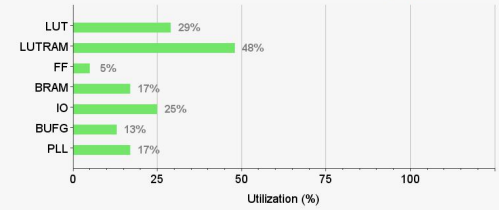
Timing Setup | Hold | Pulse Width

Worst Negative Slack (WNS):	0.457 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	98034

Implemented Timing Report

Utilization Post-Synthesis | **Post-Implementation**

Graph | Table



Power **Summary** | On-Chip

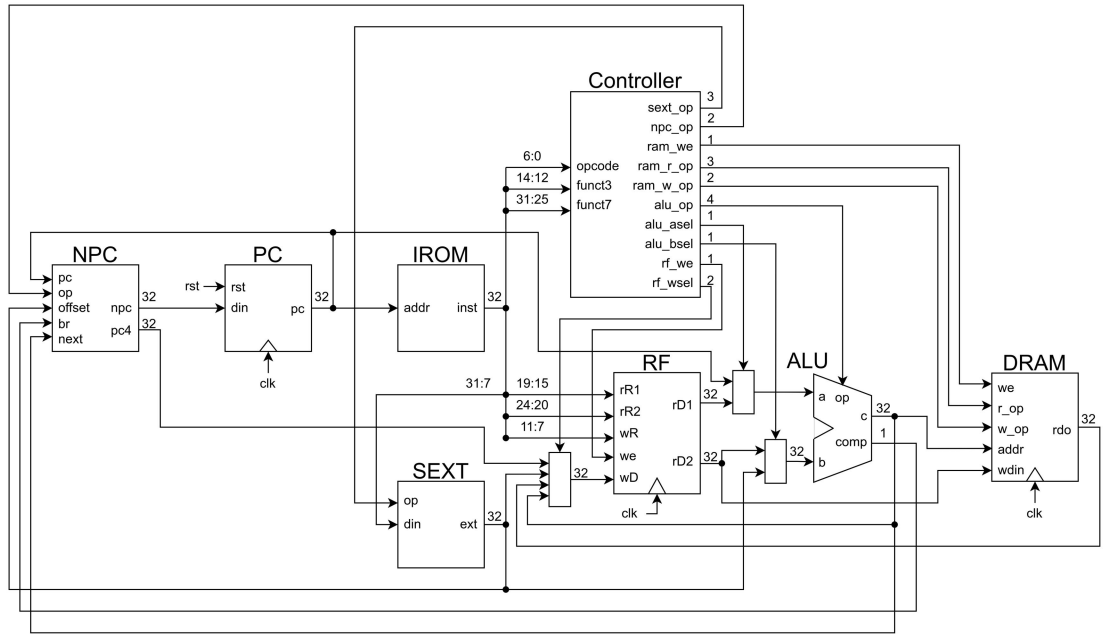
Total On-Chip Power:	**0.384 W**
Junction Temperature:	**26.0 °C**
Thermal Margin:	59.0 °C (21.9 W)
Effective ϑJA:	2.7 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

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。



- **NPC 模块**

  根据 PC 信号和跳转信号产生下一条 PC 信号。

- **PC 模块**

  接受 NPC 模块的 npc 信号，在时钟上升沿更新 PC 信号。

- **IROM 模块**

  将输入的 PC 信号作为地址，从 ROM 中读出相应的指令。

- **Controller 模块**

  根据指令的 opcode，funct3，funct7 信号，产生一系列控制信号。

- **SEXT 模块**

  根据 sext_op 信号，将指令对应部分的立即数扩展为 32 位。

- **RF 模块**

  读出指令中对应寄存器的数，在上升时钟沿向寄存器堆写入写回数据。其中，

读操作为组合逻辑，写操作为时序逻辑。

- **ALU 模块**

  将两个操作数进行算术运算、逻辑运算、移位运算或比较操作，产生对应的结果信号。

- **DRAM 模块**

  存储数据，根据地址和读写操作信号进行读取或写入操作，针对不同的 LOAD 类型指令和 S 类型指令需要对数据进行不同的加工。其中，读操作为组合逻辑，写操作为时序逻辑。

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。

NPC 模块

NPC			
信号	位宽	I/O	功能描述
pc	32	I	接受 PC.pc 信号，结合跳转信号，用于产生下一条 pc 信号
op	2	I	NPC 模块的控制信号，用于控制 NPC 模块产生 npc
offset	32	I	接受 SEXT.ext 信号，是基于 PC.pc 信号跳转的偏移量
br	1	I	接受 ALU.comp 信号，用于判断 B 型指令是否发生跳转
next	32	I	接受 ALU.c 信号，用于 JALR 指令这种直接给定 npc 的跳转指令
npc	32	O	产生下一条 pc 信号
pc4	32	O	产生 pc+4 信号

PC 模块

PC			
信号	位宽	I/O	功能描述
clk	1	I	时钟信号
rst	1	I	复位信号
din	32	I	接受 NPC.npc 信号
pc	32	O	根据 din 信号在每个上升时钟沿更新 pc 信号

IROM 模块

IROM			
信号	位宽	I/O	功能描述
addr	32	I	接受 PC.pc 信号作为指令地址
inst	32	O	从对应指令地址中取出指令

SEXT 模块

SEXT			
信号	位宽	I/O	功能描述
op	3	I	SEXT 模块的控制信号，用于立即数扩展
din	25	I	接受 IROM.inst 中立即数的信号
ext	32	O	根据 sext_op 信号对立即数进行扩展

RF 模块

RF			
信号	位宽	I/O	功能描述
clk	1	I	时钟信号
rst	1	I	复位信号，用于清空寄存器堆
rR1	5	I	接受 IROM.inst[19:15]，用于指定指令需要读取的寄存

			器号
rR2	5	I	接受 IROM.inst[24:20]，用于指定指令需要读取的寄存器号
wR	5	I	接受 IROM.inst[11:7]，用于指定指令需要写入的寄存器号
we	1	I	写入使能信号
wD	32	I	接受由多路选择器选择信号作为写入数据
rD1	32	O	从寄存器堆中对应 rR1 的寄存器读出的数据
rD2	32	O	从寄存器堆中对应 rR2 的寄存器读出的数据

ALU 模块

ALU			
信号	位宽	I/O	功能描述
op	4	I	ALU 模块的控制信号，用于控制 ALU 对操作数进行运算
a	32	I	接受多路选择器选择的 PC.pc 或 RF.rD1 信号，作为第一个操作数
b	32	I	接受多路选择器选择的 SEXT.ext 或 RF.rD2 信号，作为第二个操作数
c	32	O	根据控制信号，将 a 和 b 进行运算后得到的结果
comp	1	O	根据控制信号，将 a 和 b 进行比较操作后得到的结果

Controller 模块

Controller			
信号	位宽	I/O	功能描述
opcode	7	I	接受 IROM.inst[6:0]，作为指令的 opcode
funct3	3	I	接受 IROM.inst[14:12]，作为指令的 funct3
funct7	7	I	接受 IROM.inst[31:25]，作为指令的 funct7

sext_op	3	O	SEXT 模块的控制信号
npc_op	2	O	NPC 模块的控制信号
ram_we	1	O	DRAM 模块的写使能信号
ram_r_op	3	O	DRAM 模块的读操作控制信号，用于针对不同 LOAD 类型的指令读出的数据进行加工
ram_w_op	2	O	DRAM 模块的写操作控制信号，用于针对不同 S 类型的指令对写入的数据进行加工
alu_op	4	O	ALU 模块的控制信号
alu_asel	1	O	ALU 模块 a 信号的选择信号
alu_bsel	1	O	ALU 模块 b 信号的选择信号
rf_we	1	O	RF 模块的写使能信号
rf_wsel	2	O	RF 模块的 wD 信号的选择信号

DRAM 模块

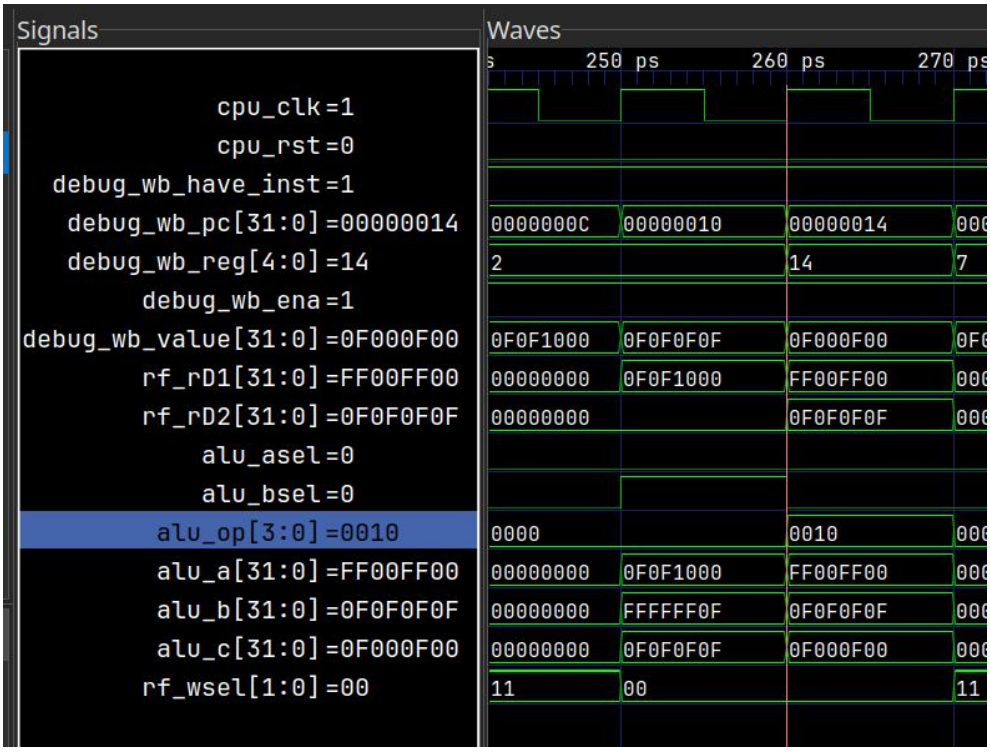
DRAM			
信号	位宽	I/O	功能描述
r_op	3	I	DRAM 模块的读操作控制信号，用于针对不同 LOAD 类型的指令读出的数据进行加工
w_op	2	I	DRAM 模块的写操作控制信号，用于针对不同 S 类型的指令对写入的数据进行加工
we	1	I	DRAM 模块的写使能信号
addr	32	I	接受 ALU.c，作为读地址或写地址
wdin	32	I	接受 RF.rD2，作为写入的数据
rdo	32	O	从 addr 地址处读出的数据

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。

- 逻辑运算（AND）

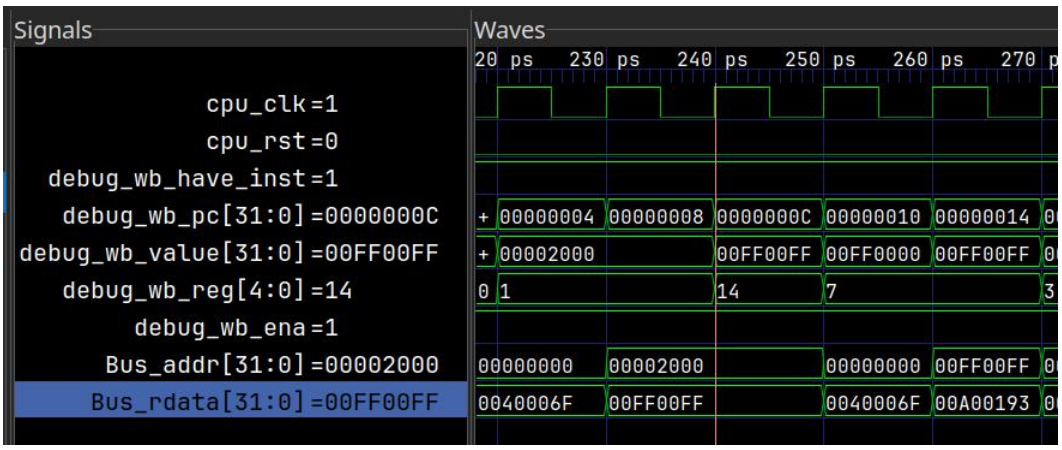
```
00000004 <reset_vector>:
   4:   ff0100b7          lui x1,0xff010
   8:   f0008093          addi    x1,x1,-256 # ff00ff00 <_end+0xff00df00>
   c:   0f0f1137          lui x2,0xf0f1
  10:   f0f10113          addi    x2,x2,-241 # f0f0f0f <_end+0xf0eef0f>
  14:   0020f733          and x14,x1,x2
  18:   0f0013b7          lui x7,0xf001
  1c:   f0038393          addi    x7,x7,-256 # f000f00 <_end+0xeffef00>
  20:   00200193          addi    x3,x0,2
  24:   48771c63          bne x14,x7,4bc <fail>
```



在 260ps 时，指令为 and x14,x1,x2，wb_pc 为 14，rf_rD1 为 FF00FF00，rf_rD2 为 0F0F0F0F。alu_asel 为 0，在 rf_rD1 和 pc 之间选择了 rf_rD1 作为 ALU 的 a 操作数。alu_bsel 为 0，在 rf_rD2 和 SEXT.ext 之间选择了 rf_rD2 作为 b 操作数。alu_op 为 0010，说明 alu 进行 and 运算。alu_c 为 0F000F00，结果正确。wb_value 为 0F000F00，wb_reg 为 14，wb_ena 为 1，写回结果也正确。

- 访存

```
00000004 <reset_vector>:
   4:   000020b7          lui x1,0x2
   8:   00008093          addi    x1,x1,0 # 2000 <begin_signature>
   c:   0000a703          lw  x14,0(x1)
  10:   00ff03b7          lui x7,0xff0
  14:   0ff38393          addi    x7,x7,255 # ff00ff <_end+0xfee0ef>
  18:   00200193          addi    x3,x0,2
  1c:   26771a63          bne x14,x7,290 <fail>
```



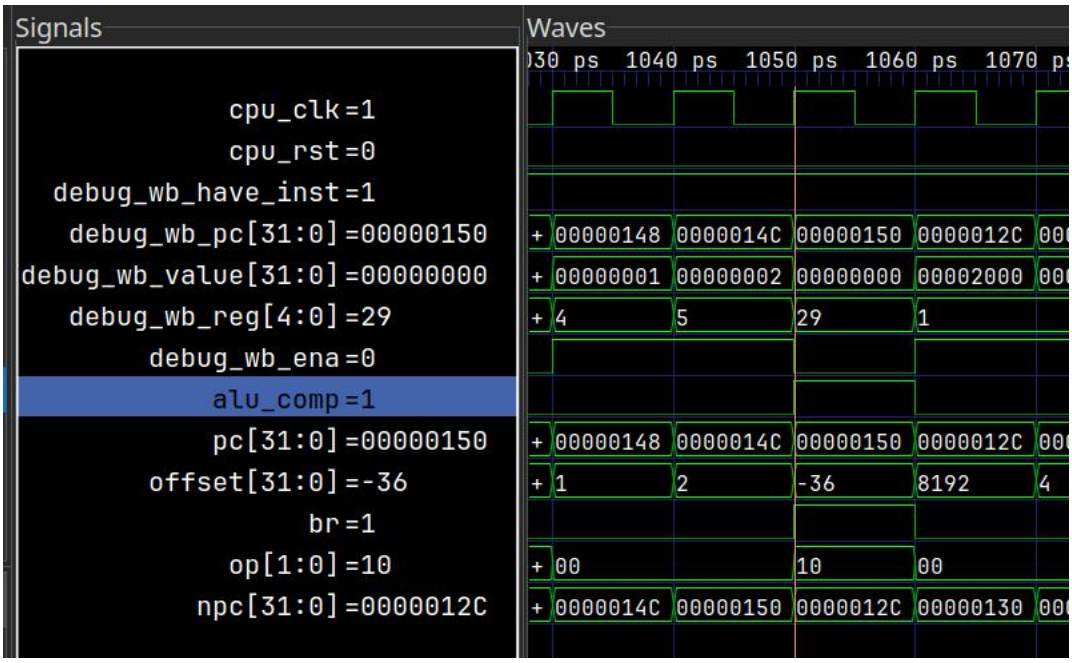
在 220ps 时,指令为 lui x1,0x2,wb_value 为 00002000,wb_reg 为 1,wb_ena 为 1，说明向 x1 寄存器写入值正确。

在 230ps 时，指令为 addi x1,x1,0，写入相关的信号保持不变。

在 240ps 时，指令为 lw x14,0(x1)，Bus_addr 为 00002000，与 reg[x1]+0 的值一致,Bus_rdata 为 00FF00FF,与 wb_value 一致,wb_reg 为 14,wb_ena 为 1，说明向 x14 寄存器写入了正确的数据。

- 分支跳转

```
  148:   00120213          addi    x4,x4,1 # 1 <_start+0x1>
  14c:   00200293          addi    x5,x0,2
  150:   fc521ee3          bne x4,x5,12c <test_12+0x8>
```

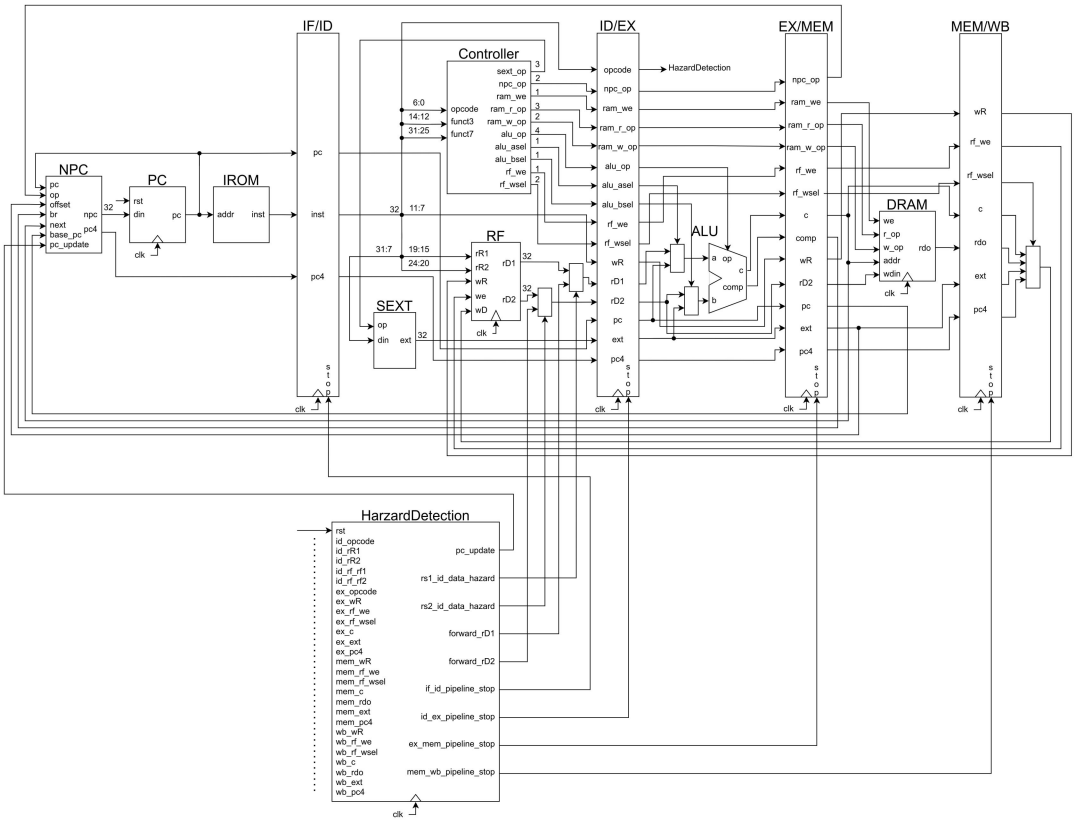


在 1030ps 时，指令为 addi x4,x4,1，写入数据说明向 x4 寄存器写入了 1。在 1040ps 时，指令为 addi x5,x0,2，写入数据说明向 x5 寄存器写入了 2。在 1050ps 时，指令为 bne x4,x5,12c，为伪指令，这时 alu_comp 为 1，说明对应指令的比较结果为真，即 x4 不等于 x5，在 NPC 中 npc_op 为 10，说明执行分支判断和跳转操作，br 为 1，说明需要执行分支跳转，offset 为-36，说明需要跳转到 pc+offset 处，npc 值为 0000012c，说明下一条 pc 值更新正确。

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。



- **HazardDetection 模块**
  接收不同流水级的信号，进行数据冒险和控制冒险的判断，向不同段寄存器发出暂停信号，生成前递数据。
- **IF 流水级**
  接受 MEM 流水级的信号实现跳转，产生 pc 信号，根据对应 pc 信号取出相应指令。
- **ID 流水级**
  根据指令生成对应的控制信号，生成扩展后的立即数，从 RF 中读出对应寄存器的数据，同时接收前递数据。
- **EX 流水级**
  根据传递来的数据进行 ALU 数据的选择，根据控制信号进行相应的运算。
- **MEM 流水级**
  从 EX 流水级中保持跳转信号，发送给 IF 流水级。从 DRAM 中读数据或写

数据。

- **WB 流水级**

  选择出向 RF 的写入数据，并向 RF 对应寄存器写入。

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、核心代码等形象化工具和手段，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 模块

NPC			
信号	位宽	I/O	功能描述
pc	32	I	接受 PC.pc 信号，结合跳转信号，用于产生下一条 pc 信号
op	2	I	NPC 模块的控制信号，用于控制 NPC 模块产生 npc
offset	32	I	接受 EX/MEM.ext 信号，是基于 EX/MEM.pc 信号跳转的偏移量
br	1	I	接受 EX/MEM.comp 信号，用于判断 B 型指令是否发生跳转
next	32	I	接受 EX/MEM.c 信号，用于 JALR 指令这种直接给定 npc 的跳转指令
base_pc	32	I	接受 EX/MEM.pc 信号，是跳转指令的 pc
pc_update	1	I	是否需要进行跳转指令的判断和更新 npc，若为 0，则 npc=pc+4
npc	32	O	产生下一条 pc 信号
pc4	32	O	产生 pc+4 信号

Controller 模块

Controller			
信号	位宽	I/O	功能描述
...	...	...	...
rf_rf1	1	O	判断是否需要读取 RF 模块 rR1 数据的信号
rf_rf2	1	O	判断是否需要读取 RF 模块 rR2 数据的信号

HazardDetection 模块

HazardDetection			
信号	位宽	I/O	功能描述
clk	1	I	时钟信号
rst	1	I	复位信号
id_opcode	7	I	ID 级指令的 opcode
id_rR1	5	I	ID 级指令的 rR1
id_rR2	5	I	ID 级指令的 rR2
id_rf_rf1	1	I	ID 级指令 rR1 是否需要读取的标志位
id_rf_rf2	1	I	ID 级指令 rR2 是否需要读取的标志位
ex_opcode	7	I	EX 级指令的 opcode
ex_wR	5	I	EX 级指令的 wR
ex_rf_we	1	I	EX 级指令 RF 的写使能信号
ex_rf_wsel	2	I	EX 级指令的向 RF 写数据的选择信号
ex_c	32	I	EX 级指令的 ALU.c
ex_ext	32	I	EX 级指令的的 SEXT.ext
ex_pc4	32	I	EX 级指令的的 NPC.pc4
mem_wR	5	I	MEM 级指令的 wR
mem_rf_we	1	I	MEM 级指令 RF 的写使能信号
mem_rf_wsel	2	I	MEM 级指令的向 RF 写数据的选择信号
mem_c	32	I	MEM 级指令的 ALU.c
mem_rdo	32	I	MEM 级指令的的 DRAM.rdo
mem_ext	32	I	MEM 级指令的的 SEXT.ext
mem_pc4	32	I	MEM 级指令的的 NPC.pc4
wb_wR	5	I	WB 级指令的 wR
wb_rf_we	1	I	WB 级指令 RF 的写使能信号
wb_rf_wsel	2	I	WB 级指令的向 RF 写数据的选择信号
wb_c	32	I	WB 级指令的 ALU.c

wb_rdo	32	I	WB 级指令的的 DRAM.rdo
wb_ext	32	I	WB 级指令的的 SEXT.ext
wb_pc4	32	I	WB 级指令的的 NPC.pc4
pc_update	1	O	NPC 模块是否需要进行跳转指令的判断和更新 npc，若为 0，则 npc=pc+4
rs1_id_data_hazard	1	O	rR1 寄存器发生了除 LOAD 型指令的数据冒险，需要将 rD1 替换为前递数据
rs2_id_data_hazard	1	O	rR2 寄存器发生了除 LOAD 型指令的数据冒险，需要将 rD2 替换为前递数据
forward_rD1	32	O	rD1 的前递数据
forward_rD2	32	O	rD1 的前递数据
if_id_pipeline_stop	1	O	IF/ID 段寄存器暂停信号
id_ex_pipeline_stop	1	O	ID/EX 段寄存器暂停信号
ex_mem_pipeline_stop	1	O	EX/MEM 段寄存器暂停信号
mem_wb_pipeline_stop	1	O	MEM/WB 段寄存器暂停信号

数据冒险的解决方法：

除了 LOAD 类型指令引起的 ID_EX 的数据冒险，对其他类型的数据冒险，采用前递机制，将所需要前递的数据和控制信号先传给 HazardDetection 模块，然后由该模块负责生成前递数据和前递选择信号用于替换 rD1 和 rD2。

核心代码如下

```
always @(*) begin
  if (rs1_id_ex_hazard && (ex_opcode != `OP_LD)) begin
    case (ex_rf_wsel)
      `WB_ALU: forward_rD1 = ex_c;
      `WB_PC4: forward_rD1 = ex_pc4;
      `WB_EXT: forward_rD1 = ex_ext;
      default: forward_rD1 = ex_c;
    endcase
  end else if (rs1_id_mem_hazard) begin
    case (mem_rf_wsel)
      `WB_ALU: forward_rD1 = mem_c;
      `WB_RAM: forward_rD1 = mem_rdo;
      `WB_PC4: forward_rD1 = mem_pc4;
      `WB_EXT: forward_rD1 = mem_ext;
      default: forward_rD1 = mem_c;
    endcase
  end else if (rs1_id_wb_hazard) begin
    case (wb_rf_wsel)
      `WB_ALU: forward_rD1 = wb_c;
      `WB_RAM: forward_rD1 = wb_rdo;
      `WB_PC4: forward_rD1 = wb_pc4;
      `WB_EXT: forward_rD1 = wb_ext;
      default: forward_rD1 = wb_c;
    endcase
  end
end
```

对于 LOAD 类型指令引起的 ID_EX 类型的数据冒险，对 IF/ID 和 ID/EX 模块都暂停一个时钟周期，然后在下一个时钟周期，冒险就变成了 ID_MEM 类型的数据冒险，这时 LOAD 类型的指令也能在 MEM 流水级中读出数据，可以使用前递机制把数据前递过去。

控制冒险的解决方法：

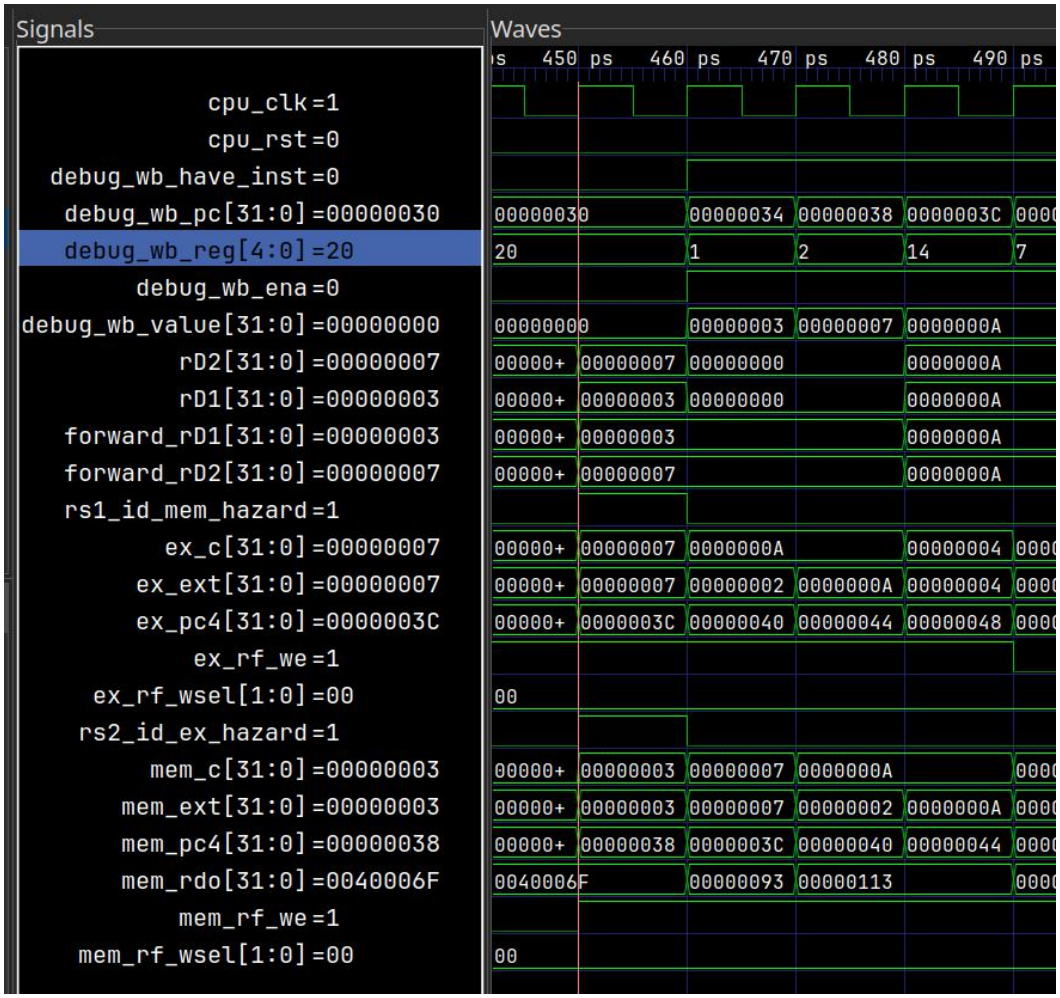
HazardDetection 模块接收 ID 流水级传来的指令信息，对于所有 B 型指令、JAL 指令和 JALR 指令，HazardDetection 将其判断为控制冒险，将 IF/ID 段寄存器暂停 3 个时钟周期，这样控制指令便能通过这 3 个时钟周期实现 EX 模块执行完毕，将 EX 模块执行的结果传给 IF 模块，和更新 pc 这三个操作，实现了指令跳转。

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。

数据冒险：

```
00000034 <test_4>:
  34:   00300093          addi    x1,x0,3
  38:   00700113          addi    x2,x0,7
  3c:   00208733          add x14,x1,x2
```

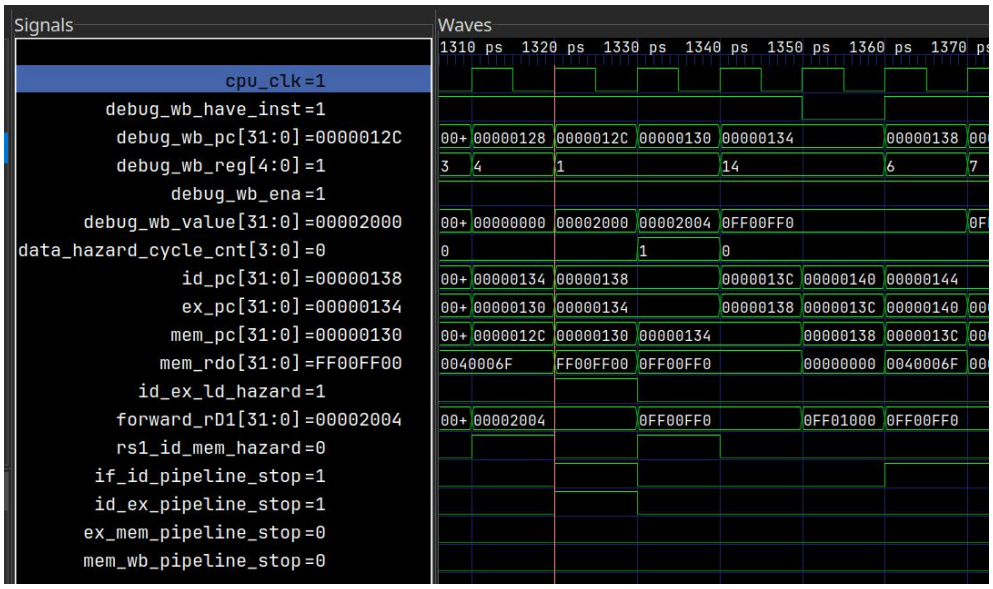


观察上述汇编代码，我们发现 38 和 3c 发生了 ID_EX 类型的数据冒险，34 和 3c 发生了 ID_MEM 类型的数据冒险。

在 450ps 时，ID 流水级正在执行 3c，EX 流水级正在执行 38，MEM 流水级正在执行 34，此时同时发生了 ID_EX 数据冒险和 ID_MEM 数据冒险，于是 rs1_id_mem_hazard 为 1 且 rs2_id_ex_hazard 为 1，HazardDetection 需要产生前递数据 forward_rD1 和 forward_rD2，发现数据正确，于是 3c 的结果正确。

LOAD 类型指令产生的 ID_EX 数据冒险：

```
00000124 <test_12>:
124:   00c00193          addi    x3,x0,12
128:   00000213          addi    x4,x0,0
12c:   000020b7          lui x1,0x2
130:   00408093          addi    x1,x1,4 # 2004 <tdat2>
134:   0040a703          lw  x14,4(x1)
138:   00070313          addi    x6,x14,0
13c:   0ff013b7          lui x7,0xff01
```

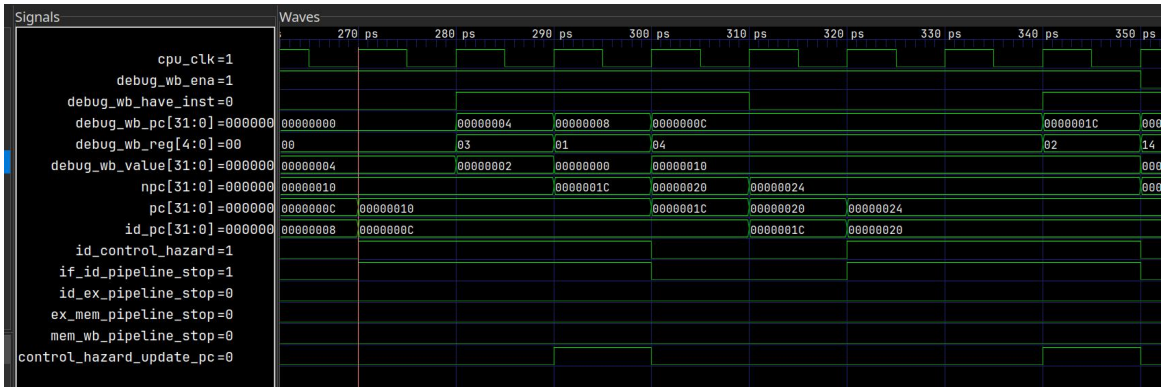


在 1320ps 时，mem_pc 为 130，ex_pc 为 134，id_pc 为 138，id_ex_ld_hazard 为 1，说明发生了 LOAD 类型指令引起的 ID_EX 数据冒险，此时 if_id_pipeline_stop 和 id_ex_pipeline_stop 为 1，暂停 IF/ID 和 ID/EX 段寄存器一个时钟周期。

在 1330ps 时，rs1_id_mem_hazard 为 1，说明发生了 ID_MEM 数据冒险，此时将数据前递回去，看到 forward_rD1 正确，于是 138 指令执行的结果正确。

控制冒险：

```
c:   0100026f          jal x4,1c <target_2>
```



在 270ps 时，id_pc 为 c，此时 id_control_hazard 为 1，因此 if_id_pipeline_stop 为 1，在下一个时钟沿开始时暂停 3 个时钟周期，第一个周期用于 ID 流水级，第二个周期用于 EX 流水级得出跳转的结果和更新 npc，第三个周期用于 pc 的更新。

在 290ps 时，即第二个周期，update_pc 为 1，说明 npc 需要更新，npc 更新成功为 1c。

在 300pc 时，pc 成功更新为 1c。

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。

在 CPU 下板时，我发现板子的数码管不停地闪烁，起初我以为是因为自己写的数码管刷新频率有问题，但是后面检查过后发现没有问题。后来经过反复的思考和检查，发现是由于我对总线的理解不到位。数码管显示的数据，不应该是直接接到总线上，这样的话，总线的数据变化了，数码管显示也会发生变化，举个例子，CPU 向 RAM 写入数据时需要占用总线，而此时数码管要显示的数据是直接接到总线上的，这时数码管显示的数据就会变成 CPU 向 RAM 写入的数据，而由于总线是经常被占用的，总线上的数据也经常变化，因此数码管也就不停闪烁。解决方法是，将数码管连上总线的写使能信号，这样，只有总线对数码管进行写入时，数码管要显示的数据才需要进行更新。

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。

个人收获：

学完本课程，我不仅亲自动手实现了单周期的 CPU，也更进一步，实现了流水线的 CPU，并实现了前递机制和暂停机制，对 CPU 的架构和流水线有了更深入的认识，对计算机的底层有了进一步的认识。

在上计组课时，我听着老师上课讲的 CPU 数据通路和流水线，如同听天书。老师课上讲数据通路从零开始讲起，从最基本的 add 指令讲起，遇到一个新指令就加一个硬件，最后即使老师只讲了一小部分指令，老师的数据通路图也看起来复杂无比，控制信号也非常多。在老师讲完数据通路图后，我仍然不是很清楚为什么要有这些控制信号和为什么数据通路图上要有这个硬件，这实际上是因为我对指令到硬件的实现不够熟悉，没有深入地去思考每条指令需要哪些硬件，需要哪些信号。

而在这门课程，在最开始设计那个信号表的时候，我才认真地去思考每条指令需要哪些硬件和哪些信号，才发现很多指令在硬件实现上基本都是类似的，才发现原来 CPU 实现不是原来想象中的那么难，这全归功于这门课将具体的指令简化为抽象的指令（通过数据通路表和控制信号取值表发现不同指令对硬件使用的相似性），然后基于抽象指令并使用模块化的思路来实现相应的硬件电路，通过这样，大大简化了设计，同时也实现了具体指令与硬件电路的联系，我发现单周期的 CPU 的代码量少的惊人，完全不是我原来所预想的那么困难，也是这样我更觉得通过抽象简化设计这一思想的伟大之处。

不过流水线 CPU 的实现确实相当困难，信号量相比单周期 CPU 多了数倍不止，并且需要深入思考每条指令的暂停机制和前递机制，当然，利用抽象的方法，其实流水线也就需要处理数据冒险和控制冒险的几种情况，不过主要还是需要对时序理解非常深刻。

建议：

CPU 下板的指导弱了一点，指导书可以在这方面写的更详细一点，我在单周期

CPU 通过 trace 之后，面对 CPU 下板有点一头雾水的感觉，课程组提供了 CPU 下板的代码框架和总线的框架是好的，但是指导书关于这方面写的不够详细。并且对于 CPU 下板，这次实验应该是只需要实现数码管和拨码开关就行了，LED 和按键开关既没有用到也没有考查，实验步骤需要更新。

实验步骤

（1）根据给出的数据通路表的示例，完成数据通路表，画出数据通路图;

（2）根据给出的控制信号表的示例，完成控制信号表，在数据通路图上添加控制信号，完成单周期CPU设计结构图;

（3）从指导书首页的实验包下载链接中下载Vivado模板工程（`proj_miniRV.zip`或`proj_miniLA.zip`）；划分好子模块，新建`.v`源文件并使用Verilog HDL实现各模块 (芯片型号：**XC7A100TFGG484-1**);

（4）编写仿真程序和使用Trace验证方法，测试设计的正确性;

（5）设计总线桥和I/O接口（必做外设接口：数码管、拨码开关、按键开关、LED）；

（6）在汇编器中导出汇编实验的机器码，参照3.2.1 定义程序ROM，将机器码拷贝到新建的`.coe`后缀的文本文件中，再将`.coe`文件导入到存储器IP核;

（7）添加约束文件，运行综合、实现、生成比特流，并完成下板运行及设计优化。