

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）

实现的指令有

共实现了 RV32I 中的 24 条必做指令，有 and, addi, add, srl, sw, sra, lui, sll, xori, beq, lw, srli, ori, bne, bge, jal, or, jalr, srli, andi, slli, sub, blt, xor。

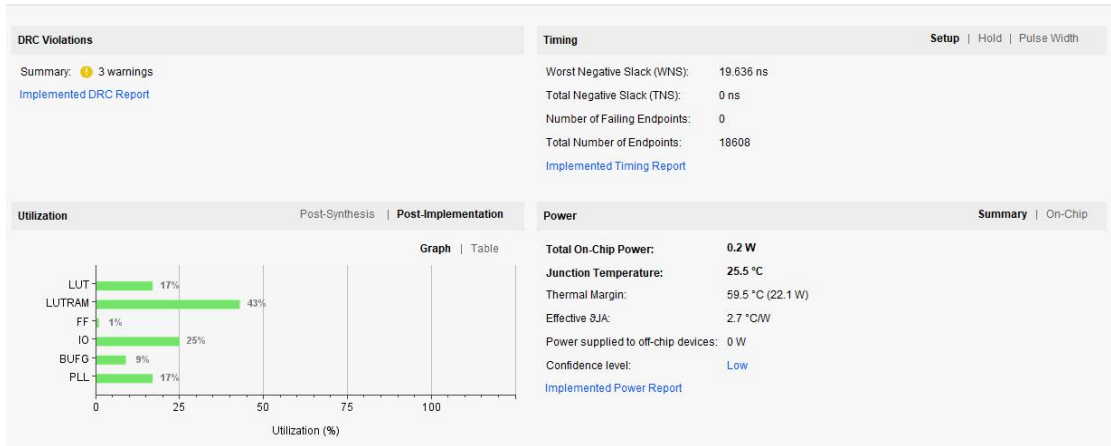
单周期 CPU 频率为 25MHZ，流水线 CPU 频率为 100MHZ。

设计的主要特色（除基本要求以外的设计）

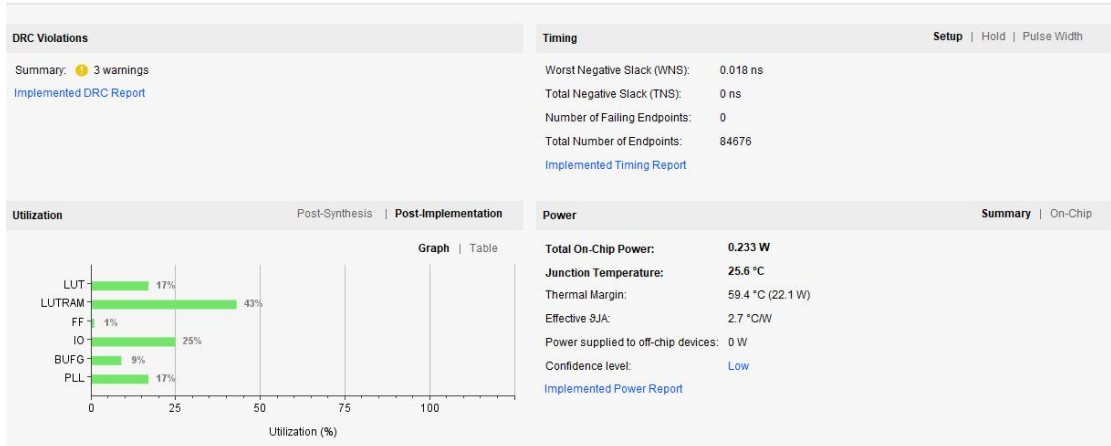
无

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：



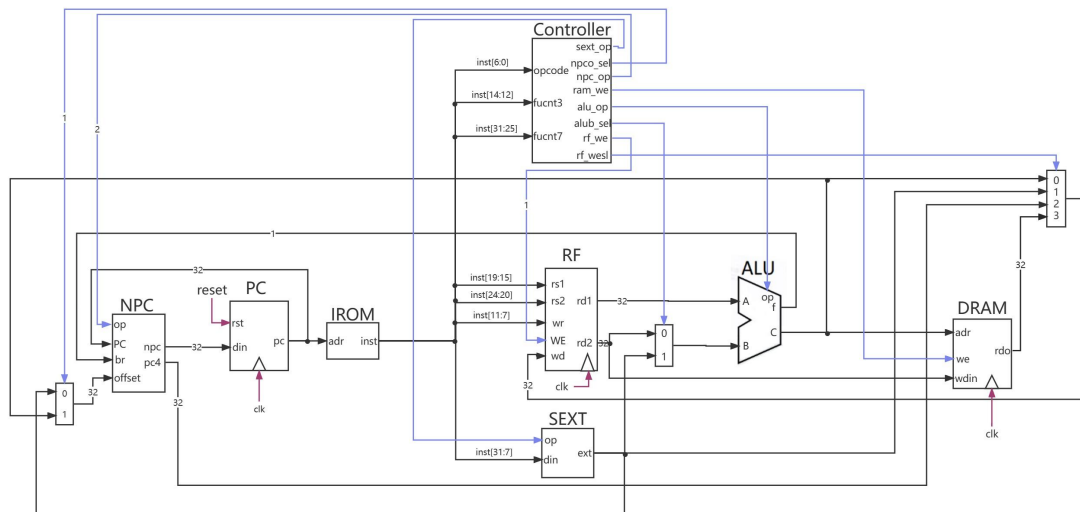
流水线：



# 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。



IF 取指单元：

NPC：生成下一条指令的地址和  $pc+4$  的值

PC：存储着当前指令的地址

IROM：指令存储器，存储指令。根据  $pc$  输出指令内容

ID 译码单元：

RF：寄存器堆，包含 32 个 32bit 寄存器

SEXT：立即数扩展单元，根据指令中立即数的格式将指令中的立即数扩展成 32 位立即数。

EX 执行单元：

ALU：算术逻辑运算单元，负责完成 CPU 中的算术、逻辑、移位和比较等运算，生成运算结果、PC 值计算结果和分支指令的跳转结果等。

MEM 访存单元：

DRAM：数据存储器，根据地址、写使能和时钟，进行数据的读写操作的时序逻辑部件。

WB 写回单元：

wb-mux：选择写回寄存器内容的来源。

CU 控制单元：

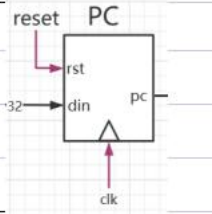
Controller：控制器，根据指令内容生成控制信号。

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:

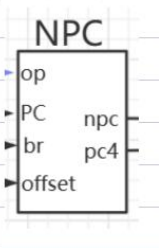
PC	输入	rst	1	复位信号
	输入	clk	1	时钟信号
	输入	din	32	下一条指令地址
	输出	pc	32	当前指令地址



```
always @(posedge clk, negedge rst_n) begin
    if(~rst_n) pc <= -4;
    else      pc <= din;
end
NPC
```

NPC

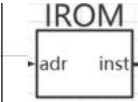
NPC	输入	op	2	控制npc的产生
	输入	pc	32	当前指令地址
	输入	br	1	条件分支是否跳转
	输入	offset	32	跳转的偏移量
	输出	pc4	32	pc+4的值
	输出	npc	32	下一条指令地址



```
assign pc4 = pc + 32'h4;
always @(*) begin
    case (op)
        `NPC_PC4:    npc = pc + 32'h4;
        `NPC_JMP:    npc = pc + offset;
        `NPC_JMPR:   npc = offset;
        `NPC_BEQ:
            begin
                if(br) npc = pc + offset;
                else npc = pc + 32'h4;
            end
        default:     npc = npc;
    endcase
end
```

IROM

IROM	输入	adr	32	指令地址
	输出	inst	32	指令



```
assign inst_addr = if_pc[15:2];
```



```
//写寄存器堆
```

```
always @(posedge clk) begin
```

```
begin
```

```
if(WE && wr!=5'b00000) regfile[wr] <= wd;
```

```
else regfile[wr] <= regfile[wr];
```

```
end
```

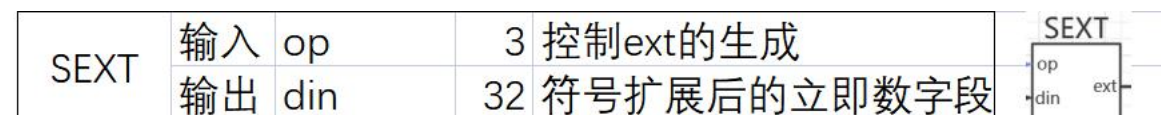
```
regfile[0] <= 0;
```

```
end
```

```
//读寄存器堆
```

```
assign rd1 = regfile[rs1];
```

```
assign rd2 = regfile[rs2];
```



```
case (op)
```

```
`EXT_I:ext = {{20{din[31]}}, din[31:20]};
```

```
`EXT_S:ext = {{20{din[31]}}, din[31:25], din[11:7]};
```

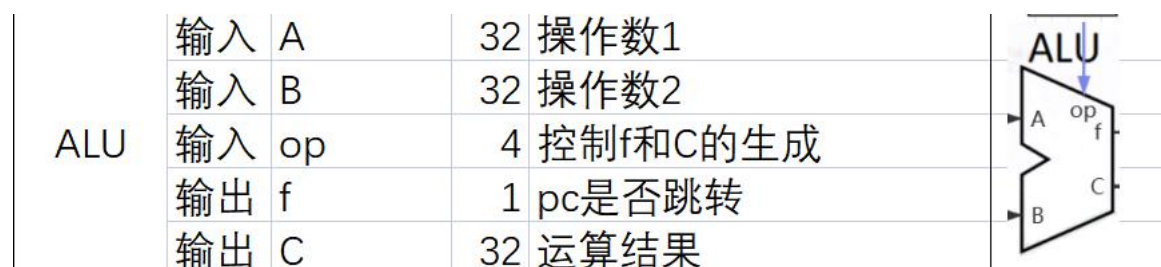
```
`EXT_B:ext = {{19{din[31]}}, din[31], din[7], din[30:25], din[11:8], 1'b0};
```

```
`EXT_U:ext = {din[31:12], 12'b0};
```

```
`EXT_J:ext = {{11{din[31]}}, din[31], din[19:12], din[20], din[30:21], 1'b0};
```

```
default: ext = 32'b0;
```

```
endcase
```



```
case (op)
```

```
`ALU_ADD: C = A + B;
```

```

`ALU_SUB: C = A - B ;
`ALU_AND: C = A & B ;
`ALU_OR : C = A | B ;

`ALU_XOR: C = A ^ B ;
`ALU_SLL: C = A << shamt;
`ALU_SRL: C = A >> shamt ;
`ALU_SRA: C = $signed(A) >>> shamt;

`ALU_BEQ: if(A==B) f=1;else f=0;
`ALU_BNE: if(A!=B) f=1;else f=0;
`ALU_BLT: if($signed(A)<$signed(B)) f=1;else f=0;
`ALU_BGE: if($signed(A)>=$signed(B)) f=1;else f=0;

default: begin C = C; f=f; end

```

DRAM	输入	clk	1	时钟信号
	输入	adr	32	地址
	输入	wdin	32	写数据
	输入	we	1	写使能
	输出	rdo	32	读数据



```

assign Bus_addr=aluC;
assign Bus_wen=dram_we;
assign Bus_wdata=rd2;
assign dram_rdo = Bus_rdata;

```

Control ler	输入	opcode	7	操作码字段
	输入	funct3	3	指令的funct3字段
	输入	funct7	7	指令的funct7字段
	输出	npc_op	2	npc的控制信号
	输出	npco_sel	1	npc的offset的选择信号
	输出	rf_we	1	寄存器堆写使能
	输出	rf_wsel	2	寄存器堆写数据的来源选择
	输出	sext_op	3	符号扩展单元的控制信号
	输出	alu_op	4	算术逻辑单元的控制信号
	输出	alub_sel	1	alu的B输入的选择信号
	输出	dram_we	1	数据存储器的写使能





### 1.3 单周期 CPU 仿真及结果分析

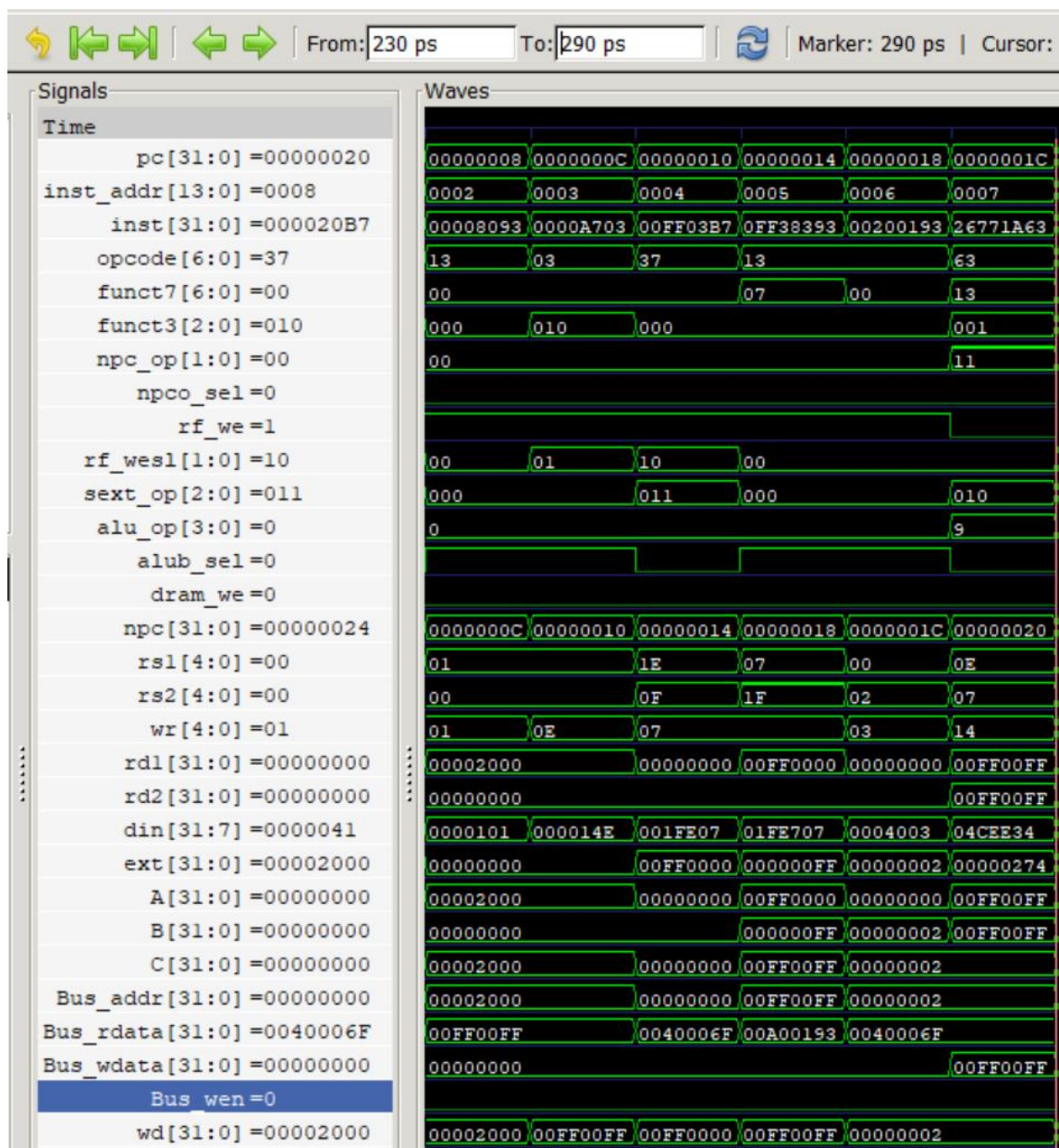
要求：包含逻辑运算、访存、分支跳转三类指令的仿真截图及波形分析；每类指令的截图和分析中，至少包含 1 条具体指令；截图需包含信号名和关键信号。

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+0xf00ff>
18: 00200193     addi x3,x0,2
1c: 26771a63     bne x14,x7,290 <fail>

```



**(1) 逻辑运算指令:****addi x1,x1,0**

230ps:

IF 模块: pc[31:0]=00000008, inst\_addr[13:0]=0002, inst[31:0]=00008093

控制器 opcode=0x13=0001101=OPCODE\_I, funct7=0000000, funct3=000

故而按照普通 I 型指令的方式, 判断为 addi。

控制器: npc\_op[1:0]=00; npc\_co\_sel=0; rf\_we=1; rf\_wesl[1:0]=00;

sext\_op[2:0]=000; alu\_op[3:0]=0; alub\_sel=1; dram\_we=0; 8 个控制信号皆符号预期。

由于 npc\_op[1:0]=00='NPC\_PC4, 故而 npc[31:0]=0000000C

ID 模块: rsl[4:0]=01, rs2[4:0]=00, wr[4:0]=01。

源寄存器 1: x1, 写回寄存器: x1

rdl[31:0]=00002000, rd2[31:0]=00000000。故而[x1]=0x2000

op[2:0]=000, din[31:7]=0000101, ext[31:0]=00000000。立即数扩展单元按照普通 I 型指令的扩展方式 (op=000) 将立即数字段 (din) 扩展为 32 位立即数 0。

EX 模块: A[31:0]=00002000; B[31:0]=00000000; C[31:0]=00002000;

由于 alu\_op[3:0]=0, 故为加运算, 符合预期。

MEM 模块: dram\_we=0, 无操作, 符合预期

WB 模块: rf\_wesl[1:0]=00, 选 aluC 写回寄存器, wd[31:0]=00002000, wr=1, WE=1, 0x00002000 写回寄存器 1, 符合预期。

**(2) 访存指令****lw x14,0(x1)**

IF 模块: pc[31:0]=0000000C, inst\_addr[13:0]=0003, inst[31:0]=0000A703

控制器: 控制器 opcode=0x13=0001101=OPCODE\_I\_LOAD, funct7=0000000, funct3=010

故而按照 load-I 型指令的方式, 判断为 lw。

npc\_op[1:0]=00; npc\_co\_sel=0; rf\_we=1; rf\_wesl[1:0]=01; sext\_op[2:0]=000;

alu\_op[3:0]=0; alub\_sel=1; dram\_we=0; 8 个控制信号皆符号预期。

由于 npc\_op[1:0]=00='NPC\_PC4, 故而 npc[31:0]=00000010

ID 模块: rsl[4:0]=01, rs2[4:0]=00, wr[4:0]=0E。

源寄存器 1: x1, 写回寄存器: x14

rdl[31:0]=00002000, rd2[31:0]=00000000。故而[x1]=0x2000

由于 op[2:0]=000, din[31:7]=000014E, ext[31:0]=00000000。立即数扩展单元按照普通 I 型指令的扩展方式 (op=000) 将立即数字段 (din) 扩展为 32 位立即数 0。

EX 模块: A[31:0]=00002000; B[31:0]=00000000; C[31:0]=00002000;

由于 alu\_op[3:0]=0, 故为加运算, 符合预期。

MEM 模块: dram\_we=0, 不写存储器, Bus\_addr[31:0]=00002000, Bus\_rdata[31:0]=00FF00FF, Bus\_wdata[31:0]=00000000, Bus\_wen=0 从地址 0x00002000 读出数据 0x00FF00FF。

WB 模块: rf\_wesl[1:0]=01, 选 DRAM.rdo 写回寄存器, wd[31:0]=0x00FF00FF, wr=0E, WE=1, 0x00FF00FF 写回寄存器 14, 符合预期。

**(3) 分支跳转指令****bne x14,x7,290**

IF 模块: pc[31:0]=0000001C, inst\_addr[13:0]=0007, inst[31:0]=26771A63

控制器: 控制器 opcode=0x63=0110011=OPCODE\_B, funct7=0x13=0010011=FUNCT7\_BNE, funct3=001。

故而按照 B 型指令的方式, 判断为 bne。

npc\_op[1:0]=11; npc\_sel=0; rf\_we=0; rf\_wesl[1:0]=00; sext\_op[2:0]=010;  
alu\_op[3:0]=1001; alub\_sel=0; dram\_we=0; 8 个控制信号皆符号预期。

ID 模块: rsl[4:0]=0E, rs2[4:0]=07, wr[4:0]=14。

源寄存器 1: x14, 源寄存器 2: x7

rdl[31:0]=00FF00FF, rd2[31:0]=00FF00FF。

由于 op[2:0]=010, din[31:7]=04CEE34, ext[31:0]=00000274。立即数扩展单元按照 B 型指令的扩展方式 (op=010) 将立即数字段 (din) 扩展为 32 位立即数。

EX 模块: A[31:0]=00FF00FF; B[31:0]=00FF00FF; C[31:0]=00000002;

由于 alu\_op[3:0]=1001, 故比较 AB 大小, 等于, f=0, 不发生跳转, 符合预期。

由于 npc\_op[1:0]=11='NPC\_BEQ 且 aluf=0, 故而不发生跳转, pc+4, npc[31:0]=00000020 符合预期

MEM 模块: dram\_we=0 无操作

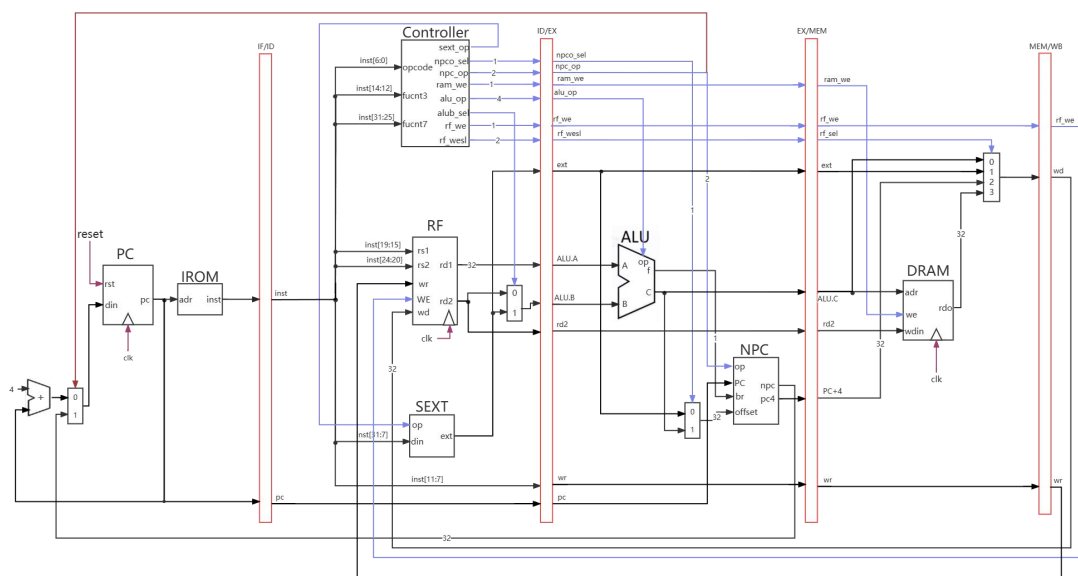
WB 模块: 由于 we=0, 不写回寄存器, 符合预期。



## 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。



IF 取指阶段：

PC：存储着当前指令的地址

IROM：指令存储器，存储指令。根据 pc 输出指令内容

ID 译码阶段：

RF：寄存器堆，包含 32 个 32bit 寄存器

SEXT：立即数扩展单元，根据指令中立即数的格式将指令中的立即数扩展成 32 位立即数。

Controller：控制器，根据指令内容生成控制信号。

EX 执行阶段：

ALU：算术逻辑运算单元，负责完成 CPU 中的算术、逻辑、移位和比较等运算，生成运算结果、PC 值计算结果和分支指令的跳转结果等。

NPC：生成下一条指令的地址和 pc+4 的值

MEM 访存阶段：

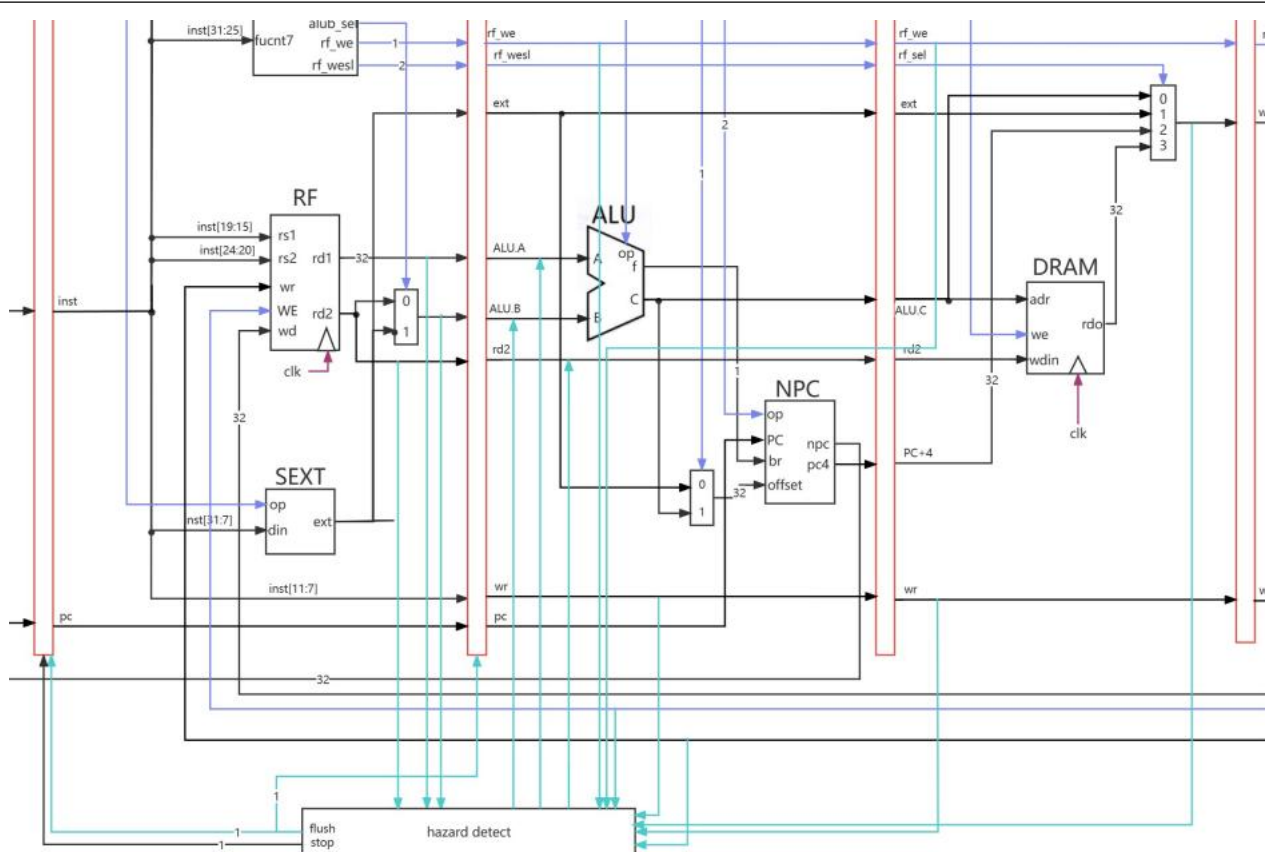
DRAM：数据存储器，根据地址、写使能和时钟，进行数据的读写操作的时序逻辑部件。

WB 写回阶段：

完成寄存器堆的写操作

## 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。



数据冒险：前递+停顿的方式解决。对于能用前递解决的冒险，全部用前递解决。不能用前递完全解决的，采用停顿的方式来解决冒险。

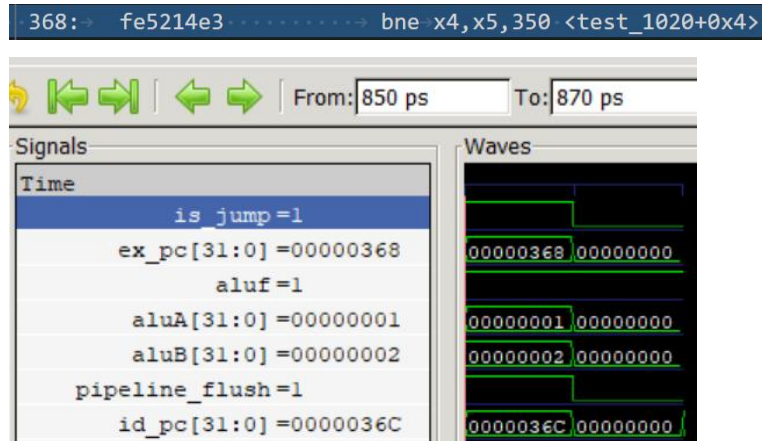
控制冒险：采用静态分支预测的方式，预测每条指令都不发生跳转，如果发生跳转，则拉高 pipeline\_flush，清空相关流水线寄存器等。

若指令 A 与指令 B 之间可能存在冒险，则冒险检测在指令 B 的 ID 阶段进行检测。取 B.rs1、B.rs2 与 A.wr 相比对，结合 we、alub\_sel 等信号判断，若确实存在冒险，则进行前递，将真正的对应的 wd 赋值给 final\_rd1 和 final\_rd2，前递到 B 指令的 EX 阶段，替代原本的 ALU.A 和 ALU.B，以及 rd2。根据冒险检测结果，输出流水线停顿信号 stop 和流水线清空信号 flush。

## 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。

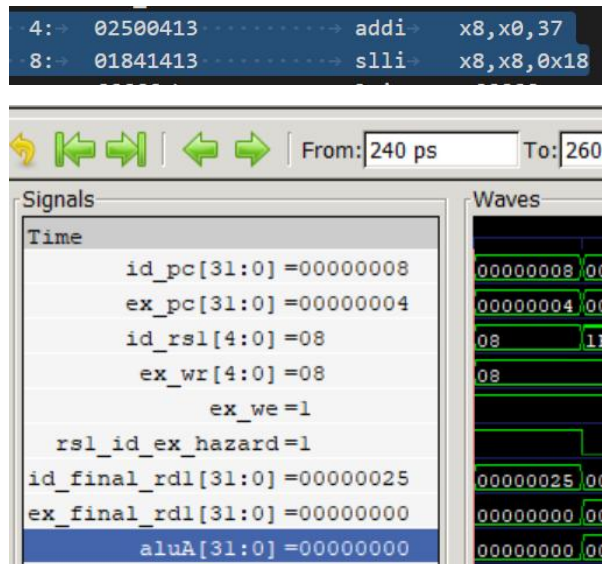
控制冒险



850-860ps ( $ex\_pc=00000368$ ) :  $ex\_pc=00000368$ ,  $bne$  指令正处于 EX 阶段，且经比较后  $aluA=00000001$  不等于  $aluB=00000002$ 。故会发生跳转  $is\_jump=1$ ，符合预期。

$pipeline\_flush=1$ ，清空 IF/ID 和 ID/EX 的流水线寄存器中的内容，使之相关内容归零。如下一个周期，860-870ps 时  $id\_pc=00000000$ ，符合预期。

数据冒险：A 情况（相邻指令发生冒险）



$addi$  指令与  $slli$  指令之间的数据冒险：

240-250ps ( $id\_pc=00000008$ ) 时：  $id\_pc=00000008$ ,  $ex\_pc=00000004$ ,  $id\_rs1=08$ ,  $ex\_wr=08$ ,  $ex\_we=1$ 。存在 A 情况数据冒险， $rs1\_id\_ex\_hazard=1$ ，符合预期。

采取前递策略： $id\_final\_rd1=ex\_wd=25000000$ 。经过 ID/EX 流水线寄存器传到  $slli$  指令的 EX 阶段，作为  $aluA$  参与运算。250-260ps,  $aluA=ex\_final\_rd1=25000000$ ，符合预期。

数据冒险：B 情况（间隔一条指令发生冒险）

```
290: 00000093 ..... addi x1,x0,0
294: 00000113 ..... addi x2,x0,0
298: 00208733 ..... add x14,x1,x2
```

rs1_id_mem_hazard=1	
id_pc[31:0] = 00000298	00000298 0000029C
mem_pc[31:0] = 00000290	00000290 00000294
id_rs1[4:0] = 01	01 00
mem_wr[4:0] = 01	01 02
mem_we = 1	
mem_wd[31:0] = 00000000	00000000
id_final_rd1[31:0] = 00000000	00000000
aluA[31:0] = 00000000	00000000

addi x1,x0,0 指令与 add x14,x1,x2 指令之间的数据冒险：

320-330ps (id\_pc=00000298) 时: id\_pc=00000298, mem\_pc=00000290, id\_rs1=01, mem\_wr=01, mem\_we=1。存在 B 情况数据冒险, rs1\_id\_mem\_hazard=1, 符合预期。

采取前递策略: id\_final\_rd1=mem\_wd=00000000。经过 ID/EX 流水线寄存器传到 add 指令的 EX 阶段, 作为 aluA 参与运算。330-340ps, aluA=ex\_final\_rd1=00000000, 符合预期。

数据冒险：C 情况（间隔两条指令发生冒险）

```
594: f0038393 ..... addi x7,x7,-256 # -256
598: 000011b7 ..... lui x3,0x1
59c: bc418193 ..... addi x3,x3,-1084 # -1084
5a0: cc7718e3 ..... bne x14,x7,270 <fail>
```

rs2_id_wb_hazard=1	
id_pc[31:0] = 000005A0	000005A0 000005A4
wb_pc[31:0] = 00000594	00000594 00000598
id_rs2[4:0] = 07	07 00
wb_wr[4:0] = 07	07 03
wb_we = 1	
wb_wd[31:0] = 0F000F00	0F000F00 00001000
id_final_rd2[31:0] = 0F000F00	0F000F00 00000000
ex_final_rd2[31:0] = FFFFFFFBC4	FFFFFFBC4 0F000F00
aluB[31:0] = FFFFFFFBC4	FFFFFFBC4 0F000F00

addi x7,x7,-256 指令与 bne x14,x7,270 指令之间的数据冒险：

3210-32200ps (id\_pc=000005A0) 时: id\_pc=000005A0, wb\_pc=00000594, id\_rs2=07, wb\_wr=07, wb\_we=1。存在 C 情况数据冒险, rs2\_id\_wb\_hazard=1, 符合预期。

采取前递策略: id\_final\_rd2=wb\_wd=0F000F00。经过 ID/EX 流水线寄存器传到 bne 指令的 EX 阶段, 作为 aluB 参与运算。3220-3230ps, aluB=ex\_final\_rd2=0F000F00, 符合预期。

### 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。

问题: PC 对应异常, 第一条指令即出错。

原因: trace 比对需要从 0 就已经开始比对了, 需要使 PC 复位后的地址不是开始地址 0。

解决方法: 将 PC 复位后的地址改为-4, 而不是 0。

问题: B 型指令跳转出错。

原因: 通过仿真分析, 发现是因为对于两个操作数的比较出错, 如-1>1。经排查发现是 cpu 将有符号数自动按照无符号数进行比较了,

解决办法:  $\$signed(A)$  可以将无符号数 A 化为有符号数, 故而采用  $\$signed(A) < \$signed(B)$  得以解决问题。

问题: 虽然流水线停顿了, 可是 trace 比对的 wb\_pc 不同。

原因: 经仿真分析和代码排查, 发现是因为尽管后续的 pc 在停顿, 可是 next\_pc 异常, 分析发现, 不仅需要流水线寄存器中的 pc 停顿, pc 本身 (if\_pc) 也需要停顿, 当 stop 拉高时, pc 寄存器需保证 pc 不变。这样才能使得 pc 在停顿前后的变化正常。

问题: 数据冒险, addi 与 sw 之间的冒险, 写数据异常。

原因: 排查后发现写数据和写地址均出错。经仿真分析发现, 由于 sw 的 rd2 并非 ALU.B, 故而前递后的数据并不能真正刷新 rd2, 反而是干扰了 ALU 的运算。

解决办法: 增加判断逻辑, 将前递的数据正确的传输到 rd2, 而不是 ALU.B。



## 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：在《计算机设计与实践》这门综合实践课程中，我实现了单周期和流水线型 CPU，是以 RISC-V 处理器为核心的片上系统。从使用 RISC-V 汇编语言编写计算器，到亲自设计并实现单周期/流水线 CPU。从中我不仅熟悉了 RARS、Logisim、Vivado 等工具的使用，还熟悉了 miniRV 指令集、RISC-V 指令系统，理解了单周期和流水线 CPU 的工作过程。不仅假设了对数字逻辑设计、计算机组成原理中的相关知识的掌握，还深入理解了计算机硬件系统的工作原理和机制，提高了解决问题的能力。

建议：增加一些计算机系统等相关理论科普，以丰富课程内容。