

实验报告

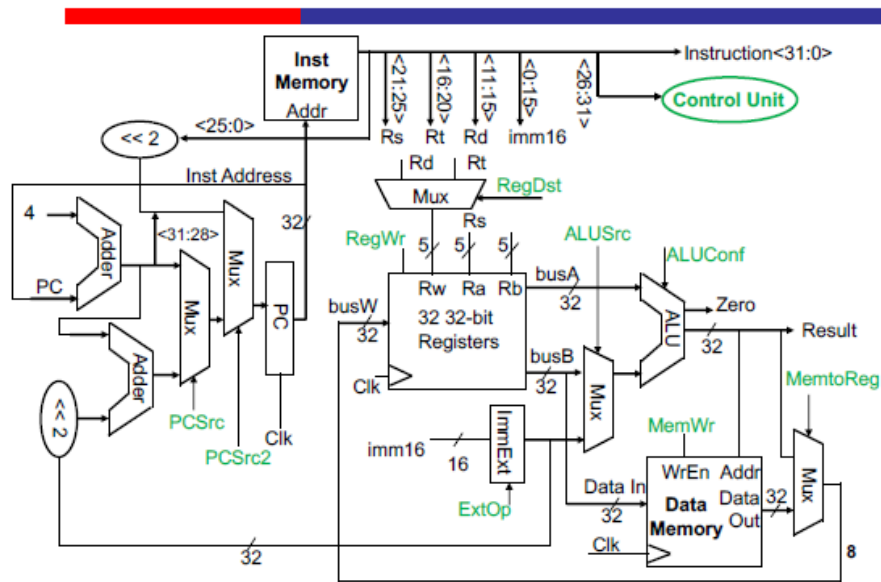
一、实验内容

将理论课处理器大作业中设计的单周期 MIPS 处理器改进为流水线结构，并利用此处理器 完成最短路径算法

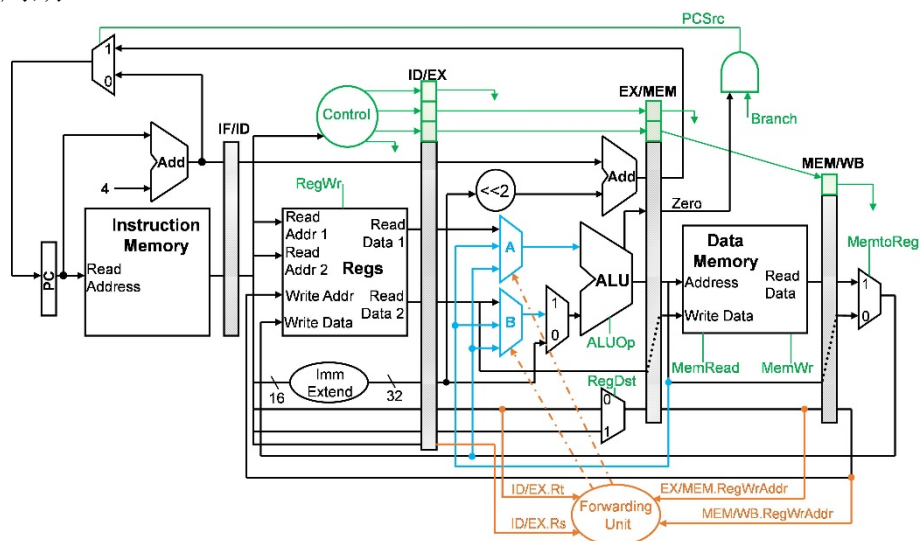
二、设计思路

(1) MIPS 基础框架，单周期：

知识回顾：单周期数据通路



多周期：



(2) 单周期升级流水线

1. 增加流水线寄存器

在单周期处理器的设计中,增加了 4 个流水线寄存器:IF_ID,ID_EX,EX_MEM, MEM_WB。4 个寄存器均为上升沿触发,分别需要保存流水线中的各种中间值和控制信号:

IF_ID:IF 阶段的 PC,取出的指令和时钟、stall 等控制信号。

ID_EX:ID 阶段的 PC,指令的操作码 OpCode,指令解码产生的 Rs、Rd、Rt,位移量 Shamt, R 型指令的 Funct,从寄存器当中读出的数据值,符号扩展单元扩展过的立即数,控制单元产生的所有控制信号(不区分在哪个阶段被使用)。

EX_MEM:EX 阶段的 PC,目标寄存器的地址 RDest,ALU 计算结果,即将写入内存中的数据,跳转目标 BranchTarget, MEM 及其之后阶段使用的控制信号。

MEM_WB:MEM 阶段的 PC,目标寄存器的地址 RDest,内存当中读出的数据,ALU 计算结果,写回时需要的控制和使能。

2. 修改分支和跳转逻辑

按照要求,需要支持新的分支指令,因此原有的 Zero 信号需要扩展。依旧在需要分支时使得 Zero 为 1,在 ALU 中增加了所有新的分支指令的判断条件,使得 Zero 信号仍在需要分支时为 1,Control 模块也要做出相应的修改。

由于 ALU 的这个结构,就完成了分支指令在 EX 阶段判断,而跳转指令在 ID 阶段判断的要求。也正是由于分支和跳转发生的阶段不同,设置 flush 信号,在分支指令发生时,冲刷 IF_ID 和 ID_EX 的控制信号;而跳转只需要冲刷 IF_ID 的控制信号。

此时,还应当修改生成 PC 下一个值的逻辑。下一个 PC 的来源主要有:PC+4,跳转目标,分支目标,中断和异常处理程序的地址,以及 stall 信号拉高时的 PC 本身。

3. 冒险处理

实现 stall 和 flush, Branch 类指令和 J 型指令都在 ID 阶段完成 PC 更新,拿到错误指令时需要 flush。

Flush: beq 类和 j 类在遇到需要 flush 时,将 flush 从 ID 发到 IF_ID,对 ID_EX 阶段的 flush 配合 stall 进行;

Stall: 用于 load_use 和 beq

```
89 | assign flush_IFID = (ID_IsJump | ID_Branch)&(stall_IFID == 0);
90 | assign stall_IFID = (ID_Branch &&
91 |     (( EX_RegWrite && (EX_rd == IF_ID_Instruction[25:21] || EX_rd == IF_ID_Instruction[20:16]) )
92 |     || (MEM_MemRead && (MEM_rd == IF_ID_Instruction[25:21] || MEM_rd == IF_ID_Instruction[20:16])))
93 |     || (EX_MemRead && (EX_rd == IF_ID_Instruction[25:21] || EX_rd == IF_ID_Instruction[20:16])))
94 | )? 1:0;
```

加入 Forwarding unit

- 转发到 ID: Branch 提前需要的数据,来源:EX/MEM
- 转发到 Ex:EX/MEM, MEM/WB
- 转发到 MEM 只会来源于 MEM/WB
- beq 前一条为 R 或是 lw 都需要 stall 然后转发到 ID 阶段的转发与 ID 阶段的判断计算:

```

163 Forwarding_ID BrForwarding(IF_ID_Instruction[25:21], IF_ID_Instruction[20:16],
164                             MEM_rd, MEM_RegWrite,
165                             MEM_MemRead, WB_rd, BrForwardingA, BrForwardingB);
166 assign BrALUDat1 = BrForwardingA == 1 ? MEM_ALU_out :
167               BrForwardingA == 2 ? WB_Databus3 : ID_Databus1;
168 assign BrALUDat2 = BrForwardingB == 1 ? MEM_ALU_out :
169               BrForwardingB == 2 ? WB_Databus3 : ID_Databus2;
170 BranchALU BrAlu(IF_ID_Instruction[31:26], BrALUDat1, BrALUDat2, ID_Branch, Zero);
171

```

到 EX 阶段的转发:

```

225 Forwarding_EX ALUForward(
226     EX_rs, EX_rt,
227     MEM_rd, WB_rd,
228     MEM_RegWrite, WB_RegWrite, MEM_MemRead,
229     ALUForward1, ALUForward2
230 );
231
232 assign ALU_in1 = EX_ALUSrc1? {27'h00000, EX_Shamt}:
233               (ALUForward1==2)? MEM_ALU_out:
234               (ALUForward1==1)? WB_Databus3 : EX_Databus1;
235 assign ALU_in2 = EX_ALUSrc2? EX_LU_out:
236               (ALUForward2==2)? MEM_ALU_out:
237               (ALUForward2==1)? WB_Databus3 : EX_Databus2;
238 ALU alu1(
239     .in1    (ALU_in1),
240     .in2    (ALU_in2),
241     .ALUCtl (EX_ALUCtrl),
242     .Sign   (EX_Sign),
243     .out    (EX_ALU_out),
244     .zero   (nouse_Zero)
245 );

```

Load-use 冒险: 先 stall 一个周期, 再行转发。根据理论课所学知识, stall 信号产生的条件是: EX 阶段执行 load 指令; ID 阶段的 Rs、Rt 源寄存器与 EX 阶段的 load 指令的 Rd 寄存器相同。stall 信号一旦拉高, PC 寄存器的值保持不变, IF_ID 寄存器不变, 而后续寄存器控制信号全部置零。

三、汇编代码

1. 采用 bellman-ford 算法计算最短路径, 累加得到输出结果
2. 没有除法计算, 故使用累减获得带余除法

```

#16进制转10进制存储
#循环计算余数
addi $t2, $t0, 0
addi $t4, $zero, 0
addi $t3, $zero, 10
div_loop1:
    sub $t2, $t2, $t3          #被除数减除数
    addi $t4, $t4, 1          #统计除法结果
    bgtz $t2, div_loop1       #结果大于等于0, 循环
    beq $t2, $zero, div_loop1
add $s0, $t2, $t3            #余数, 结果小于0则加上出书
addi $t4, $t4, -1            #结果

```

3. 0-9 翻译成 BCD 控制信号

0	00111111	0x3f
1	00000110	0x06
2	01011011	0x5b
3	01001111	0x4f
4	01100110	0x66
5	01101101	0x6d
6	01111101	0x7d
7	00000111	0x07
8	01111111	0x7f
9	01101111	0x6f

4. 软件 BCD 显示控制:

```

print:
addi $t0, $zero, 1          # t0: count time
addi $t8, $zero, 100
print_loop:
    beq $t0, $t8, final_end
    lui $t9, 0x4000
    addi $t9, $t9, 0x0010    # the address for BCD control
    sw $s4, 0($t9)
    jal wait_func
    sw $s5, 0($t9)
    jal wait_func
    sw $s6, 0($t9)
    jal wait_func
    sw $s7, 0($t9)
    jal wait_func
    j print_loop

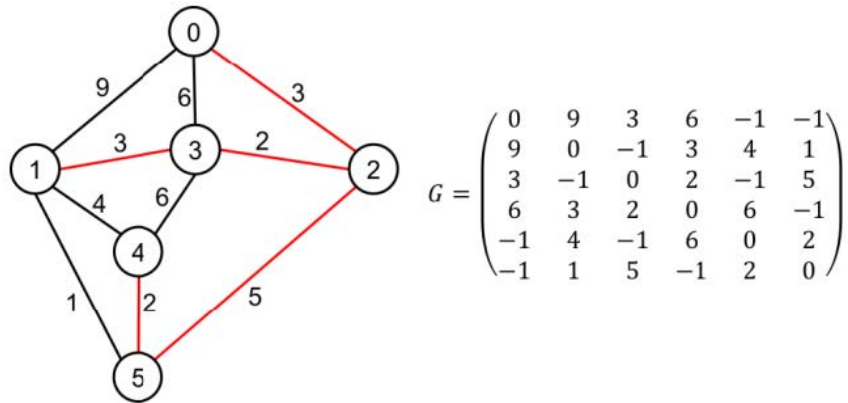
wait_func:
    addi $t1, $zero, 1
    addi $t2, $zero, 10
wait_loop:
    beq $t1, $t2, end_wait_loop
    addi $t1, $t1, 1
    nop
    j wait_loop
end_wait_loop:
    jr $ra

final_end:
    nop

```

四、调试

1. 采用理论课样例测试



答案应为 8 3 5 10 8，即输出 34

2. 调试

(1) 在只有 R-beq 或者 lw-beq 这类时会同时产生 IF_ID 的 flush 和 stall，此时会产生同为 1 的冲突。调试时选择 beq 留在原地，即 flush 失效

(2) 汇编指令与 verilog 设计时要注意单位的不同。lw 指令中的单位是 bytes，而 verilog 中的单位一般是 bit，在设计寄存器大小时要注意大小的换算。

(3) ID 阶段更新 PC 时，计算 PC+4 要用 ID_PC 不能直接用 IF 的 PC，这样 beq 在 IDstall 的时候 PC 也不会跑掉了。实际上这段 PC 更新还是要再考虑考虑，原来的 PC 是直接在 ID 阶段的逻辑里算出来的，问题就是 ID_PC 和现在的 PC 已经不是一个东西了。要分开讨论用哪个进行更新

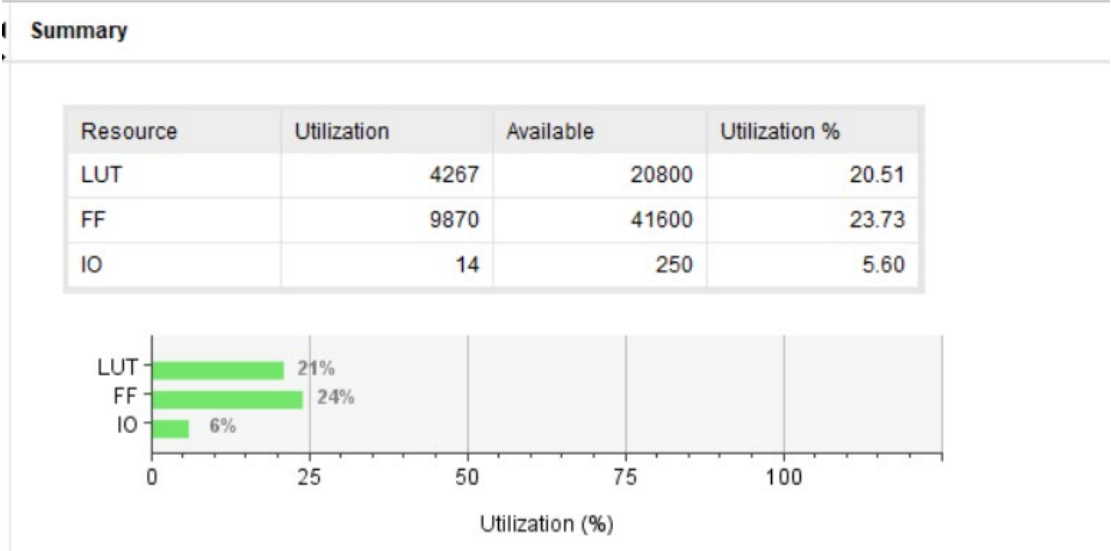
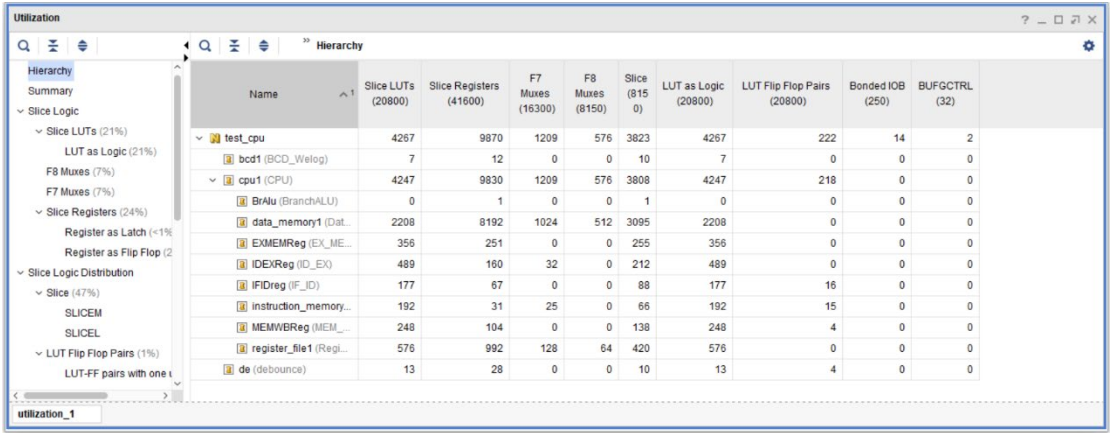
```
182     assign ID_PC_plus_4 = ID_PC + 32'd4;
183     wire [32-1:0] Jump_target;
184     assign Jump_target = {ID_PC_plus_4[31:28], IF_ID_Instruction[25:0], 2'b00};
185     wire [32-1:0] Branch_target;
186     assign Branch_target = (ID_Branch & Zero)? ID_PC_plus_4 + {ID_LU_out[29:0], 2'b00}:
187                               (ID_Branch)? ID_PC_plus_4: PC + 32'h4;
```

(4) bgtz 和 bltz 要注意不能直接使用 >, < 号比较，这是无符号比较，要 check 符号位和是否是 0。

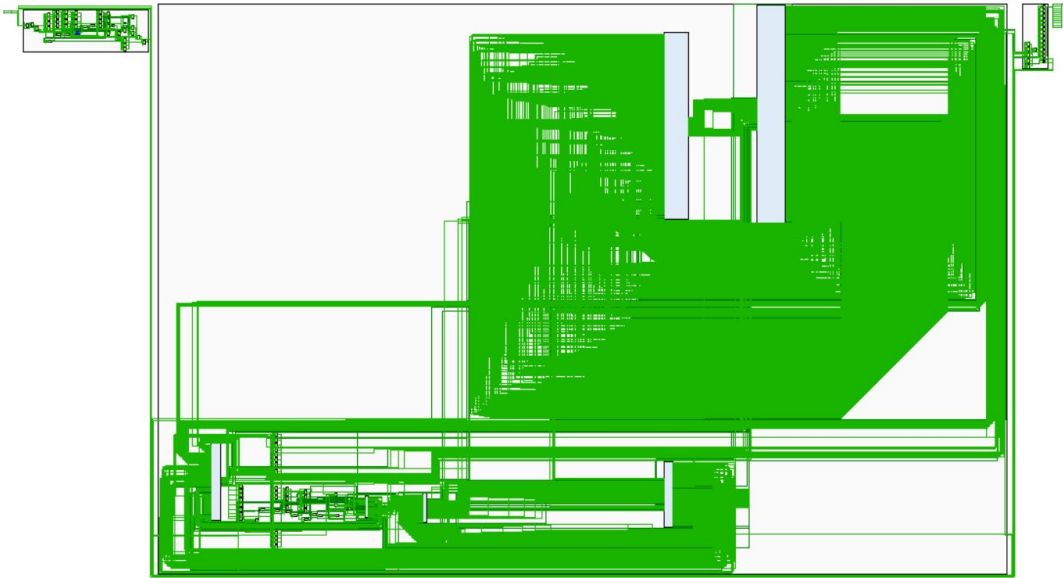
```
else if(OpCode == 6'h07) begin // bgtz
    Zero <= ((data1[31] == 0) & ~(data1 == 0));
end
else if(OpCode == 6'h01) begin // bltz
    Zero <= (data1[31] == 1 );
end
```

五、资源与时序性能使用

资源使用情况

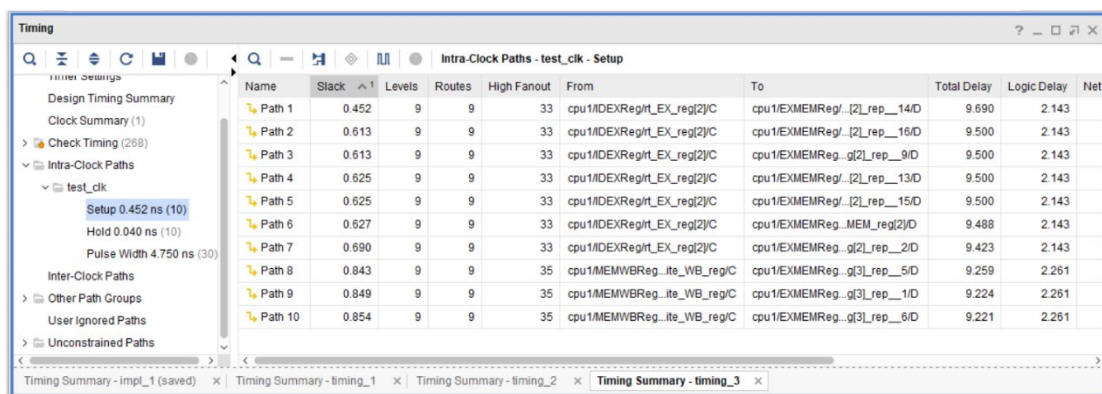
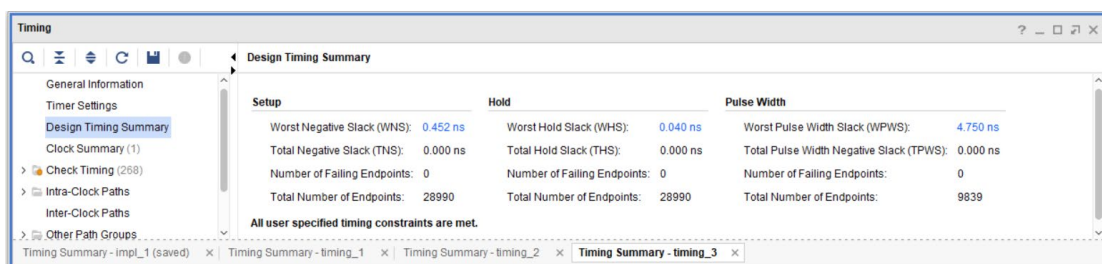
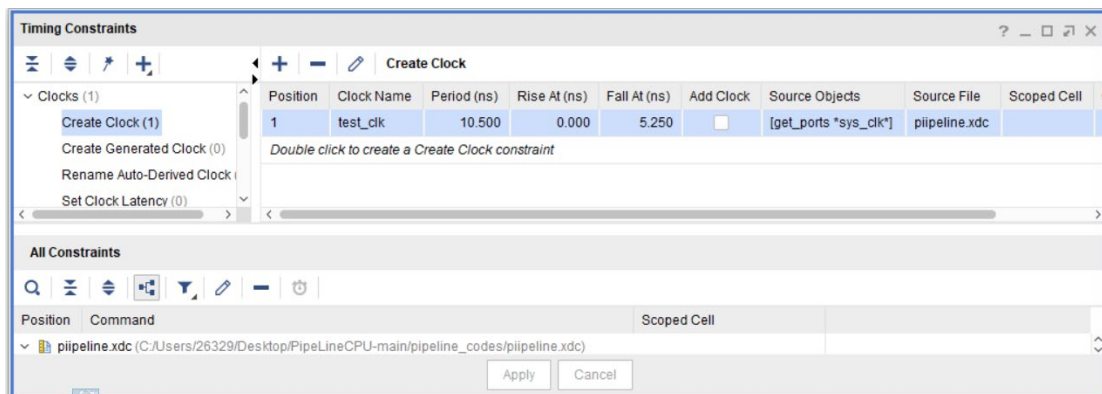


Schematic



时序情况

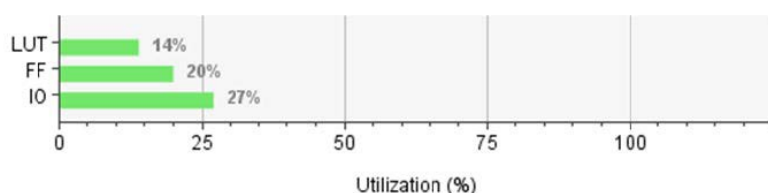
设置时钟约束为 10.5ns（测试时初始设置 10ns，此时 WNS 和 TNS 为负数，效果并不好。调试后选择了 10.5ns），实际使用从管脚接来的 100M 信号

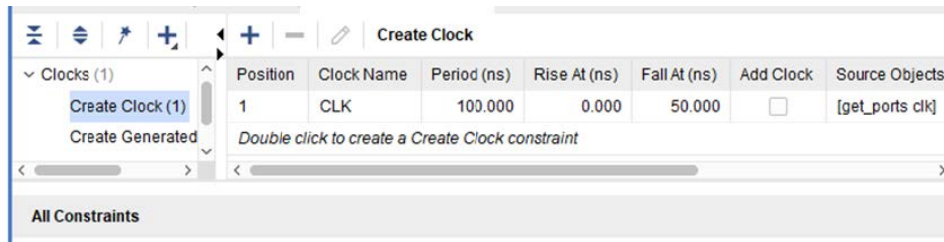


最短周期大约在 9.690ns 左右，最大时钟频率 $F = \frac{1}{9.690ns} = 103.2MHz$

与单周期处理器对比

Resource	Utilization	Available	Utilization %
LUT	2928	20800	14.08
FF	8425	41600	20.25
IO	68	250	27.20





$$\text{最大时钟频率约为 } F_m = \frac{1}{(clk-WNS)} = \frac{1}{(100-87.965)} = 99\text{MHz}$$

两者对比，可以发现流水线处理器用到了更多的 LUT 和 FF，其中 LUT 增长比较明显。因为虽然流水线中确实有了更多的级间寄存器，但是实际消耗主要是存储器实现，FF 消耗增加较少；LUT 则是由于增加了各种控制信号、转发处理等消耗量增大。

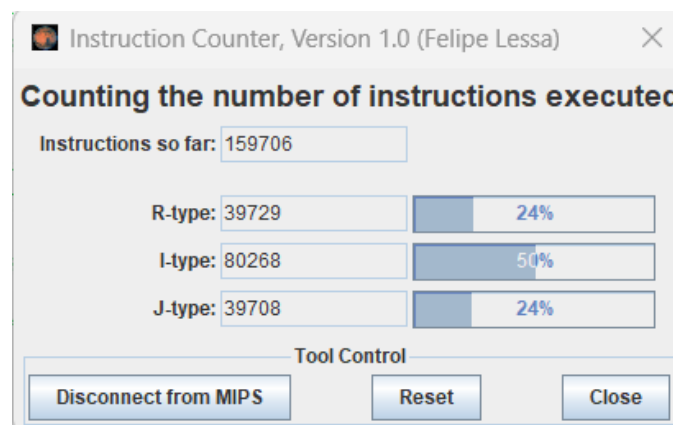
但是优化结果显示频率增加不明显，一方面可能是单周期实现功能有所不同，逻辑更简单，以及理论课测试的并没有很精确。另一方面也有可能是 vivado 综合实现过程中设计的问题，单周期综合实现所需时间远高于 pipeline 的综合实现时间。

六、仿真结果

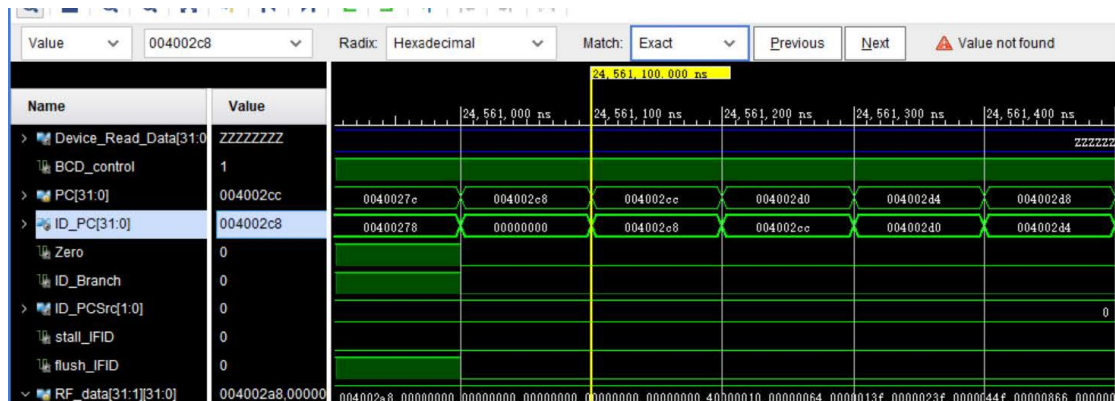
仿真结果：16-19 寄存器中存储的是每位上的十进制数字，20-23 存储译码结果，与测试答案 34 符合

Name	Value	00000000	00000001	00000010	00000011	00000100	00000101	00000110	00000111	00001000	00001001	00001010	00001011	00001100	00001101	00001110	00001111	00010000	00010001	00010010	00010011	00010100	00010101	00010110	00010111	00011000	00011001	00011010	00011011	00011100	00011101	00011110	00011111	00100000	00100001	00100010	00100011	00100100	00100101	00100110	00100111	00101000	00101001	00101010	00101011	00101100	00101101	00101110	00101111	00110000	00110001	00110010	00110011	00110100	00110101	00110110	00110111	00111000	00111001	00111010	00111011	00111100	00111101	00111110	00111111	01000000	01000001	01000010	01000011	01000100	01000101	01000110	01000111	01001000	01001001	01001010	01001011	01001100	01001101	01001110	01001111	01010000	01010001	01010010	01010011	01010100	01010101	01010110	01010111	01011000	01011001	01011010	01011011	01011100	01011101	01011110	01011111	01100000	01100001	01100010	01100011	01100100	01100101	01100110	01100111	01101000	01101001	01101010	01101011	01101100	01101101	01101110	01101111	01110000	01110001	01110010	01110011	01110100	01110101	01110110	01110111	01111000	01111001	01111010	01111011	01111100	01111101	01111110	01111111	10000000	10000001	10000010	10000011	10000100	10000101	10000110	10000111	10001000	10001001	10001010	10001011	10001100	10001101	10001110	10001111	10010000	10010001	10010010	10010011	10010100	10010101	10010110	10010111	10011000	10011001	10011010	10011011	10011100	10011101	10011110	10011111	10100000	10100001	10100010	10100011	10100100	10100101	10100110	10100111	10101000	10101001	10101010	10101011	10101100	10101101	10101110	10101111	10110000	10110001	10110010	10110011	10110100	10110101	10110110	10110111	10111000	10111001	10111010	10111011	10111100	10111101	10111110	10111111	11000000	11000001	11000010	11000011	11000100	11000101	11000110	11000111	11001000	11001001	11001010	11001011	11001100	11001101	11001110	11001111	11010000	11010001	11010010	11010011	11010100	11010101	11010110	11010111	11011000	11011001	11011010	11011011	11011100	11011101	11011110	11011111	11100000	11100001	11100010	11100011	11100100	11100101	11100110	11100111	11101000	11101001	11101010	11101011	11101100	11101101	11101110	11101111	11110000	11110001	11110010	11110011	11110100	11110101	11110110	11110111	11111000	11111001	11111010	11111011	11111100	11111101	11111110	11111111
------	-------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------

CPI

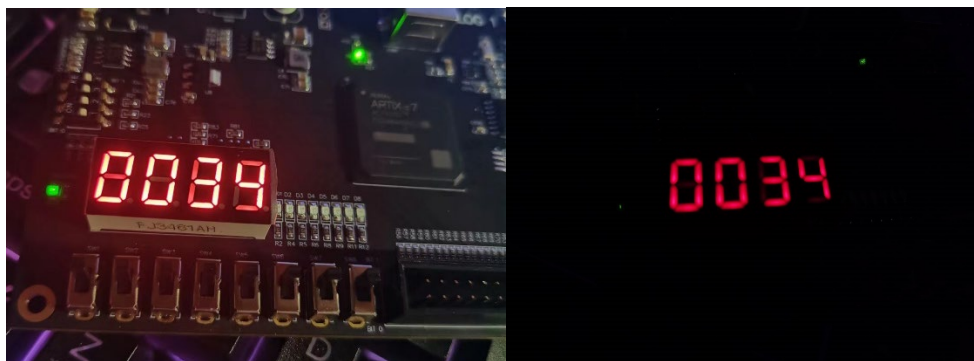


循环执行原汇编程序，得到指令数 159706 个。仿真得到流水线使用时间为 24561400ns，即 245614 个 cycle



$$CPI = \frac{245614}{159706} = 1.5379134519$$

硬件运行：



运行正确。

七、总结

本次实验系统比较复杂，需要的知识全面，工作量较大，设计方案需要从总体角度考虑周全。

在进行流水线的设计时，首先考虑的是添加流水线寄存器。在这一过程中，一点点添加每个流水线寄存器需要保存的数据和控制，中间寄存器的宽度逐渐增大，代码行数也逐渐膨胀。到了最后，反倒要重新整理一遍这些信号，把它们在 CPU 顶层文件当中写清。

编写汇编代码同样也是很困难的一部分，对这一部分的工作量预估不够也导致未能如期完成实验。所幸从头开始复习了一遍汇编之后，写出的代码能够在处理器上运行。这深刻地说明，硬件和软件的设计是密切关联的。

八、文件清单

- pipeline 约束和设计文件
 - ALU.v
 - ALUControl.v

- BCD_Welog.v
- BranchALU.v
- clk_gen.v
- Control.v
- CPU.v
- DataMemory.v
- debounce.v
- Device.v
- EX_MEM.v
- Forwarding_EX.v
- Forwarding_ID.v
- ID_EX.v
- IF_ID.v
- InstructionMemory.v
- MEM_WB.v
- piipeline.xdc 约束文件
- RegisterFile.v
- sim_pip.v 仿真文件
- test_cpu.v 最终框架设计
- asm
 - bellman_4pipeline.asm 原始的汇编程序
 - bellman_4pipeline_4CPIcalculate.asm 有限循环次
 - machinecode.txt 汇编译码后导出的文件
 - mq.txt 上一个文件提出来的机器指令
 - instruction.txt 机器码指令代码
- 实验报告