

数字逻辑与处理器基础实验 实验报告

综合实验：流水线 MIPS 处理器的设计

无 13 管思源 2021012702

实验目的

- 掌握经典 5 级流水线处理器的结构和工作流程
- 理解流水线处理器中的冒险问题，掌握其解决方法
- 理解处理器与外设的通信方式，掌握数码管 / UART 串口作为外设的实现方式
- 学会使用 Vivado 对处理器进行仿真验证和性能分析

设计方案

总体方案

本次综合实验的工作量较大，我们将任务分拆成四大步骤：

1. 正确实现 MIPS 单周期处理器
 - 在数字逻辑与处理器课程大作业提供的代码基础上，优化代码结构
 - 扩展实现 `bne`、`blez`、`bgtz`、`bltz` 和 `jalr` 指令
 - 设计仿真测试文件，验证处理器运行各指令的正确性
2. 在单周期处理器基础上，添加与外设通信的能力
 - 设计顶层模块、外设控制模块，使其符合单周期处理器的接口
 - 添加与数码管通信的能力，编写 MIPS 汇编代码并在开发板上验证
 - 添加与 UART 串口通信的能力，编写 MIPS 汇编代码并在开发板上验证
3. 正确实现 MIPS 流水线处理器
 - 在单周期处理器基础上，添加流水线结构
 - 通过数据转发、冒险检测等手段解决冒险问题
 - 修改单周期仿真测试文件，验证流水线处理器的运行结果符合预期
4. 整合各部分代码达到实验要求，优化处理器性能
 - 整合流水线处理器和外设控制模块，测试数码管和 UART 串口工作正常
 - 编写排序和输入输出数据的 MIPS 汇编代码，在开发板上验证全流程
 - 利用 Vivado 进行静态时序分析，优化处理器结构，提高频率

在以下部分，我将按步骤顺序说明**硬件代码和仿真测试的设计思路**，MIPS 汇编代码的编写详见“算法指令”部分，在开发板上的验证情况详见“硬件调试”部分，性能优化详见“综合结果与分析”部分。

步骤一：单周期处理器

这一部分主要需要解决的问题是：单周期处理器代码中的控制信号是由我自己编写实现的，无法确保其功能正确性，因此需要设计仿真测试直观地进行验证。

`single_cycle/cpu/testbench/test_cpu.v` 是我设计的仿真测试文件，它的功能有

- 例化处理器，提供时钟和复位信号，等待处理器运行指定的周期数
- 输出处理器每个周期的 PC 值和对应的指令
- 读取同文件夹的 `checkpoints.txt` 文件，验证处理器在指定周期的 PC 值、寄存器值或内存值是否与文件中的预期值一致
- 统计并输出测试结果，包括测试总数、通过数、失败数、通过率等信息

由于处理器的内部运行情况和预期值都与测试文件独立，因此只要处理器的接口不变，这份仿真测试文件可以适用于任意的处理器类型和运行的汇编代码，包括后面的流水线处理器。

同时，这也意味着我们还需要设计测试用的汇编代码和预期结果，它们分别位于 `program/test_instructions.asm|mem` 和 `single_cycle/cpu/testbench/checkpoints.txt`。在这两份文件中，我对所要支持的每条 MIPS 指令依次进行了执行验证，所用到的寄存器也尽量不重复，方便进行调试排错。

其中 `.mem` 文件是 MARS 软件导出的指令文本文件，每行包括一条指令、8 个 16 进制字符。经过改写的 `InstructionMemory.v` 可以通过 `$readmemh` 读取这个文件到程序内存中，更方便我们在不同程序之间的切换。

在编写 `checkpoints.txt` 时需要注意，因为 MARS（或者说 MIPS）的程序内存是从地址 `0x00400000` 开始的，因此在（第一次跳转之后）检查 PC 值需要加上 `0x00400000`，或者像我一样手动修改 `j` 类指令的指令码

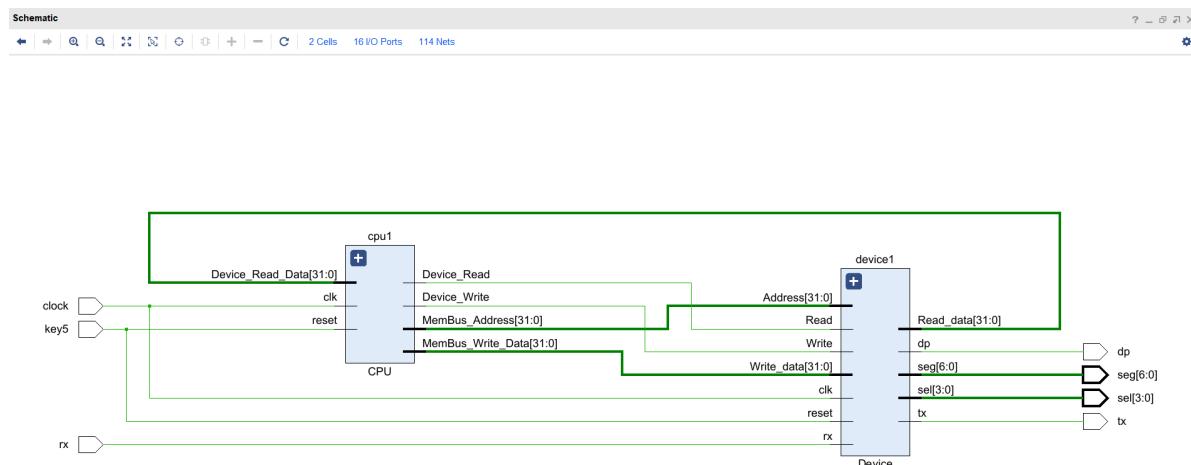
此外，在这个步骤里还需要修改已有的单周期处理器代码来支持 `bne`、`blez`、`bgtz`、`bltz` 和 `jalr` 指令，这主要涉及到修改 `Control` 模块中的控制信号和 `CPU` 模块中 `Branch_target` 的计算。由于只要按照理论课上提供的方法计算真值即可，这里不多赘述。

唯一需要注意的是，`blez`、`bgtz`、`bltz` 这三条指令不能通过类似 `ALU_out <= 0` 的条件判断来实现，因为负数的 `ALU_out` 是由补码表示的，始终满足 `ALU_out >= 0`。相反，我们应该直接通过 `ALU_out` 的最高位（也就是 `ALU_out[31]`）来判断其正负性。

值得一提的是，这里的代码错误是由上面的仿真测试运行结果报错发现的，进一步说明了先设计仿真测试文件的必要性

步骤二：处理器与外设的通信

单周期处理器的代码中实际已经预留了与外设通信的接口，我将其略作修改后形成了如下的系统框图：



其中所有外设被封装到了一个抽象的 `Device` 模块 (`single_cycle/device/Device.v`)，处理器通过地址 `Address`、读使能信号 `Read`、读数据总线 `Read_data`、写使能信号 `write` 和写数据总线 `Write_data` 与 `Device` 模块通信。这样一来，外设与一个 RAM 的模块定义别无二致。

当然，我们需要修改 `CPU` 模块内部访存相关的逻辑来支持访问外设资源，相关代码摘要如下：

```
// CPU.v
assign MemBus_Address = ALU_out;
assign is_Memory = MemBus_Address < 32'h40000000;
assign Memory_Read = MemRead && is_Memory;
assign Memory_Write = MemWrite && is_Memory;
assign Device_Read = MemRead && !is_Memory;
assign Device_Write = MemWrite && !is_Memory;
assign MemBus_Read_Data = is_Memory? Memory_Read_Data: Device_Read_Data;
assign MemBus_Write_Data = Databus2;
```

可以看到，这段代码相当于给访存操作的输入输出信号都加上了一个选择器，其控制信号就是地址是否小于 0x40000000。

在实现完 CPU 模块的外设相关逻辑后，我们要设计落实抽象封装的 Device 模块，其输入输出包括数码管的 sel[3:0]、seg[6:0] 和 dp，以及 UART 串口的 rx 和 tx，要求根据地址和读写使能信号来进行相应的读写操作。

首先是数码管，由于是纯输出设备，因此我们只需要实现它的写操作，相关代码如下：

```
// Device.v
always @(posedge clk or posedge reset) begin
    if (reset) begin
        sel <= 4'd0;
        dp <= 1'd0;
        seg <= 7'd0;
    end
    else begin
        if (write) begin
            case (Address)
                32'h40000010: begin
                    sel <= write_data[11:8];
                    dp <= write_data[7];
                    seg <= write_data[6:0];
                end
            endcase
        end
    end
end
end
```

接下来是 UART 串口。出于模块化设计的考虑，我借鉴实验三的代码设计了 UART 模块（single_cycle/device/UART.v），它通过消息发送端口 TXD、消息接收端口 RXD、和控制信号端口 CON 与上级的 Device 模块通信。

为了实现串口状态读取后的自动清零、以及写入消息后的启动发送，UART 模块还为每个端口配备了一个控制信号（TXD_write、RXD_read 和 CON_read），用于指示该周期对应端口是否被读取或写入。

以下是 Device 模块中 UART 相关的代码摘要：

```
// Device.v
assign UART_RXD_read = Read && (Address == 32'h4000001C);
assign UART_CON_read = Read && (Address == 32'h40000020);
assign UART_TXD_write = Write && (Address == 32'h40000018);

assign Read_data =
    UART_RXD_read? UART_RXD:
    UART_CON_read? UART_CON:
    32'h00000000;

always @(posedge clk or posedge reset) begin
    if (reset) begin
        UART_RXD <= 32'h00000000;
    end
    else begin
        if (write) begin
            case (Address)
                32'h40000018: begin
                    UART_RXD <= write_data;
                end
            endcase
        end
    end
end
```

```

    end
end
end

```

注：这部分代码在硬件调试过程中经历了多次修改，现在看到的是最终正常工作的版本。关于主要难点和改进方向，详见下文“硬件调试”部分。

步骤三：流水线处理器

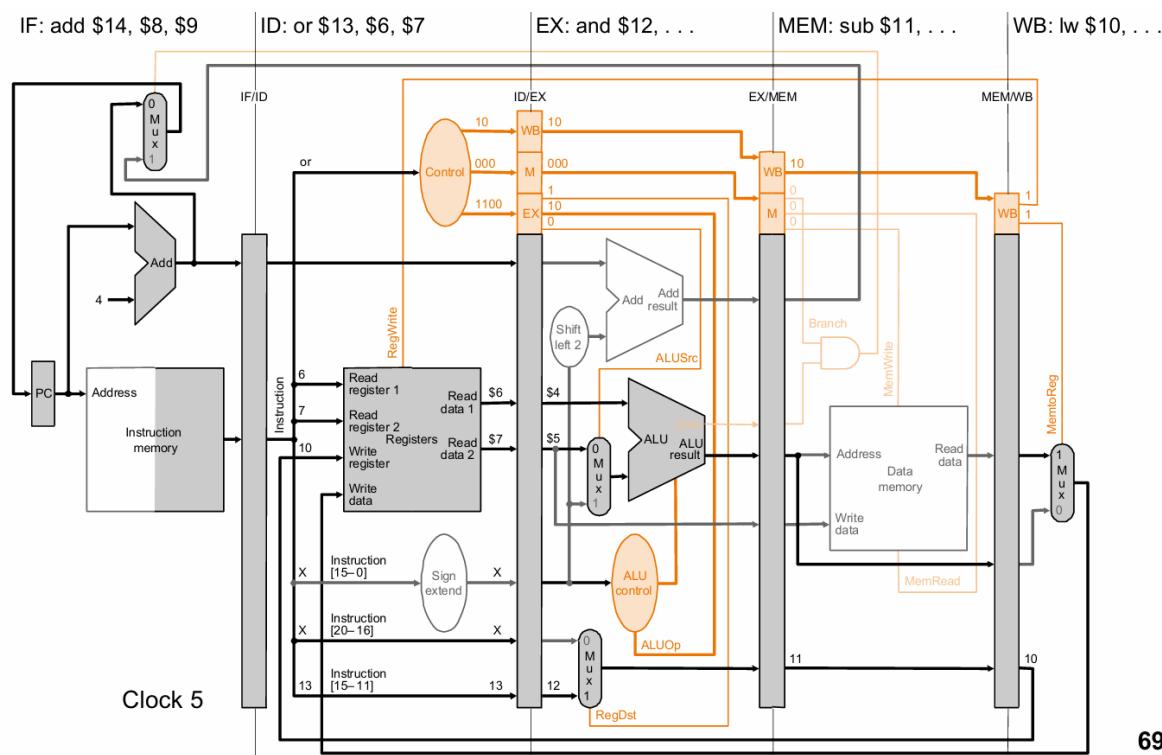
在这一部分，我们迎来了综合实验的重头戏：流水线处理器的实现。

经过分析理论课知识和前面的单周期处理器模块结构后，我发现对于单周期和流水线处理器，诸如计算单元 ALU 模块、控制信号 Control 模块等子模块的功能和工作流程没有变化。我们只需要在单周期处理器代码基础上修改 CPU 模块的内部逻辑，就可以将其改造成流水线结构。

对于单周期处理器中的 CPU 模块，整个模块只有一个寄存器，即 PC。每个时钟周期都是从 PC 开始，经过取指、译码、执行、访存、写回等流程后，再在下一个时钟上升沿更新 PC（也就是在过程块中赋值的过程）。流水线处理器将这个过程分为 5 个独立的阶段，两个阶段之间设有若干个记录过程信息的寄存器，每个时钟周期从上一个阶段末尾的寄存器开始，完成本阶段的任务后在下一个时钟上升沿更新到下一个阶段的寄存器中。因此，在前面代码基础上，我们只需要

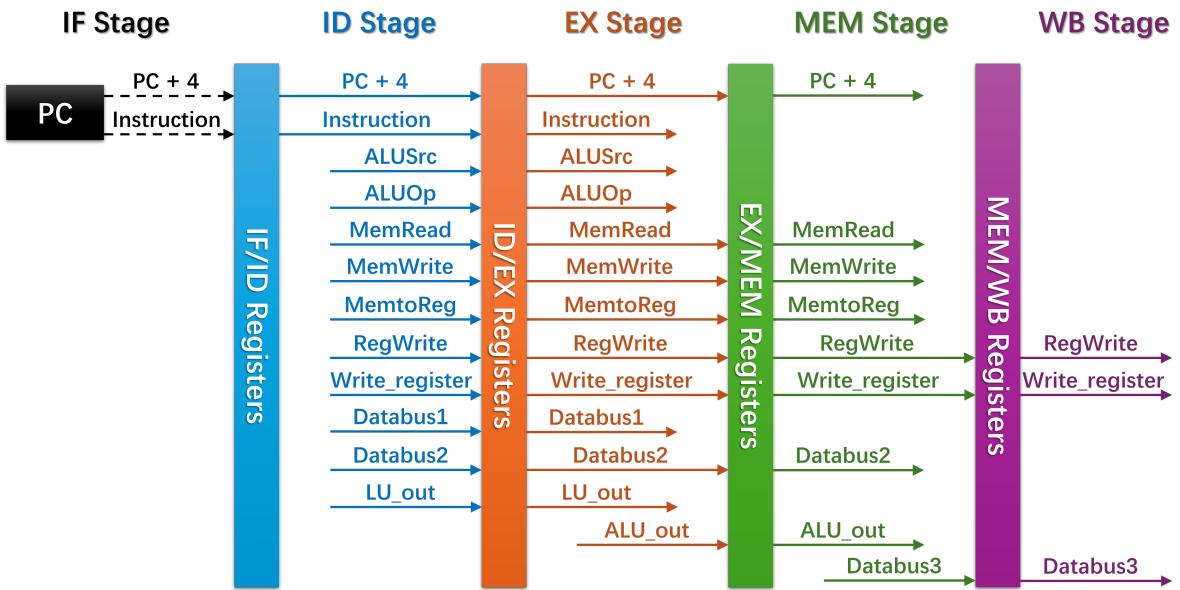
1. 分清每部分代码属于哪个阶段，理出 5 个独立的阶段
2. 在 4 个阶段间隙定义好所需的寄存器，在过程块中完成其同步赋值
3. 修改并检查各个阶段的代码，只使用上阶段传递下来的寄存器或本阶段产生的信号

这部分工作其实比起脑力更耗费眼力，因为流水线结构可以直接参照理论课课件（如下图），而后面出错更多的反而是因为有某个变量看漏了、忘记修改了，需要耐心 + 细心的检查。



69

以下是修改完成后各个阶段控制信号和数据的传递流程图，与理论课略有差异（主要是为了代码实现的方便，并不影响实际功能）



在流水线框架搭建完毕后，我们就需要考虑随之而来的冒险问题了。

首先，我们来解决数据冒险问题。根据理论课的知识，这个问题的解决方案包含三个部分：第一是寄存器先写后读、第二是数据转发、第三是对于 load-use 冒险要延迟一周期

对于寄存器先写后读，目前单周期版本的代码并不支持，不过我们很容易就可以通过判断当前周期读的寄存器是否与写的寄存器相同来实现，以下是修改代码：

```
// RegisterFile.v
assign Read_data1 =
  (Read_register1 == 5'b00000)? 32'h00000000:
  (write && Read_register1 == write_register)? write_data:
  RF_data[Read_register1];
assign Read_data2 =
  (Read_register2 == 5'b00000)? 32'h00000000:
  (write && Read_register2 == write_register)? write_data:
  RF_data[Read_register2];
```

对于数据转发，我们可以创建一个 Forwardingunit 模块（pipeline/cpu/ForwardingUnit.v），根据当前寄存器的读写情况来判断是否需要进行数据转发，以及从哪个阶段转发数据。由于原理及实现与理论课相同，这里不多赘述，直接给出代码摘要：

```
always @(*) begin
  // Forward A
  if (EX_MEM_RegWrite && (EX_MEM_Write_Register != 0) && (EX_MEM_Write_Register == ID_EX_RS))
    ForwardA <= 2'b10; // Forward from EX/MEM
  else if (MEM_WB_RegWrite && (MEM_WB_Write_Register != 0) && (MEM_WB_Write_Register == ID_EX_RS))
    ForwardA <= 2'b01; // Forward from MEM/WB
  else
    ForwardA <= 2'b00; // No forwarding
  // Forward B
  // ...
end
```

对于 load-use 冒险，我们最早可以在 load 指令处于 EX 阶段、use 指令处于 ID 阶段判断用到的是不是同一个寄存器。同时为了实现延迟操作，我们创建一个 Hazardunit 模块（pipeline/cpu/HazardUnit.v）来向各个阶段之间的寄存器传递控制信号，相关代码摘要如下：

```

wire Load_use_hazard;
assign Load_use_hazard = ID_EX_MemRead && ((ID_EX_Rt == IF_ID_Rs) || (ID_EX_Rt == IF_ID_Rt));

wire Stall_one;
assign Stall_one = Load_use_hazard;

assign PC_Stall = Stall_one;
assign IF_ID_Stall = Stall_one;
assign ID_EX_Flush = Stall_one;

```

其中 `PC_Stall` 为真时 PC 寄存器不更新，`IF_ID_Stall` 为真时 IF/ID 阶段的寄存器不更新，`ID_EX_Flush` 为真时 ID/EX 阶段的寄存器清零，延迟一周期意味着这三者同时为真。

接下来，在解决完数据冒险问题后，我们来考察分支和跳转问题。

我们需要先确保 PC 值的更新逻辑是正确的。虽然我们与单周期时一样译码得出了是否进行分支和跳转、并且计算得到了分支和跳转的地址，但是它们分别是在 ID 和 EX 阶段，利用本阶段的寄存器值得出来的，是否能采用与单周期时一样的 PC 更新逻辑呢？

答案是否定的，不过我们也只需理顺 PC 值更新的先后逻辑，其余的判断实际上仍与单周期时的类似。对于流水线处理器（分支在 EX 阶段判断、跳转在 ID 阶段判断），下一周期的 PC 值依次有如下几种可能性：

1. 如果三个周期前的指令（当前在 EX 阶段）是分支指令，且分支条件满足，那么 PC 值更新为 EX 阶段计算出的分支地址
2. 如果条件 1 不满足，且两个周期前的指令（当前在 ID 阶段）是跳转指令，那么 PC 值更新为 ID 阶段译码得出的跳转地址（逻辑与单周期时相同）
 - 如果 `PCSsrc == 2'b01`，那么跳转地址为指令码中的目标地址
 - 如果 `PCSsrc == 2'b10`，那么跳转地址为寄存器中的数据
3. 如果上面两个条件都不满足，那么 PC 值更新为当前 PC 值加 4

因此，我们可以写成如下代码：

```

assign PC_next =
  (ID_EX_Branch && Branch_condition)? Branch_target:
  (PCSsrc == 2'b00)? PC_plus_4:
  (PCSsrc == 2'b01)? Jump_target: Databus1;

```

对于流水线处理器，在分支和跳转操作时除了需要更新 PC 值外，还需要撤销已经在 IF 阶段（对于分支操作还有 ID 阶段）的错误执行的指令。为此，我们可以直接更新上面的 `HazardUnit` 模块，具体代码摘要如下：

```

wire Flush_one;
assign Flush_one = Jump;

wire Flush_two;
assign Flush_two = Branch;

assign PC_Stall = Stall_one;
assign IF_ID_Flush = Flush_one || Flush_two;
assign IF_ID_Stall = Stall_one;
assign ID_EX_Flush = Stall_one || Flush_two;

```

与延迟一周期不同的是，分支和跳转操作会清空已经在执行的指令，并且不会阻止更新 PC 寄存器。

最后，我们的流水线处理器理论上已经可以正常工作了，但是还需要仿真测试来确保代码中没有出现错误。正如前面所说，我们可以照搬单周期处理器的仿真测试文件，只需要修改测试程序 `program/test_instructions.asm|mem` 和预期值文件 `pipeline/cpu/testbench/checkpoints.txt`。

对于流水线处理器的仿真测试，主要的改动包括

- 寄存器值较单周期时需要延后 4 个周期检验（单周期需要 1 个周期写回、而流水线在第 5 个周期写回）
- 内存值较单周期时需要延后 3 个周期检验（单周期需要 1 个周期访存、而流水线在第 4 个周期访存）
- 分支和跳转操作时，PC 值会先顺序执行到后 1 / 2 行指令，再跳转到目标地址，需要修改这部分的预期值
- 增加了有关 RAW 和 load-use 冒险问题的测试用例

步骤四：整合各部分代码，优化处理器性能

由于外设部分可以兼容单周期或流水线处理器，因此我们可以照搬单周期处理器的相关代码。因为外设部分在单周期处理器下已经实际验证过、流水线处理器本身功能也通过了仿真验证，因此这部分唯一可能出错的部分就是将排序和输入输出数据功能整合起来的 MIPS 汇编代码。

为了快速修改错误的 MIPS 汇编代码（或单纯地希望切换程序），我还将 `InstructionMemory` 模块进一步改写为 BRAM 格式（同步读写，这在单周期处理器中无法做到），这样可以通过我编写的 TCL 脚本 (`pipeline/update_program.tcl`) 直接更新程序内存，无需重新综合。

有关 MIPS 汇编代码的编写，详见下面的“算法指令”部分；有关优化处理器性能的内容，详见下面的“综合结果与分析”部分。

算法指令

这一部分主要涉及到 6 个 MIPS 汇编代码的编写（均位于 `program` 文件夹），它们分别是

- `test_display.asm`：用于测试数码管外设
- `test_uart.asm`：用于测试 UART 串口外设
- `sort.asm`：排序算法
- `sort_display.asm`：读取内存中数据进行排序，并将排序结果显示在数码管上
- `sort_uart.asm`：通过 UART 串口输入数据，排序后再通过 UART 串口输出
- `sort_uart_display.asm`：通过 UART 串口输入数据，排序后将排序结果显示在数码管上的同时通过 UART 串口输出

还有 `test_instructions.asm` 用于测试处理器功能，前面已经介绍，这里不再赘述

test_display.asm

这个 MIPS 汇编程序需要通过软件方式将数字译为 7 段码，并控制 4 位数码管的动态显示

首先是译码部分，我们单开一个函数 `map`（89 行开始），它以 `$a0` 为参数表示 0 ~ 15 的整数，在 `$v0` 返回译出的 7 段码，以下是部分代码：

```
# test_display.asm
addi $t0, $zero, 0
addi $v0, $zero, 0x3f
beq $a0, $t0, map_return
addi $t0, $zero, 1
addi $v0, $zero, 0x06
beq $a0, $t0, map_return
# ...
map_return: jr $ra
```

与硬件译码（1 周期完成）相比，这段代码平均需要 25 个周期（以单周期计）执行。不过好处在于足够灵活，可以显示实验一代码之外的 a ~ f 等字母。

其次是动态显示部分，我们创建一个 `display` 函数，它接收 `$a0` 为参数表示要显示的数字（使用低 16 位表示 4 位 16 进制数字），同时接收 `$a1` 为参数表示循环显示的次数（以 0.4 秒为单位）。

在函数内部，先调用 `map` 函数将译码得到的 4 位数字的 7 段码分别存入寄存器 `$t1` ~ `$t4` 中，并分别加上 `0x0100 ~ 0x0800` 作为 `sel` 信号。然后遍历 `$t1` ~ `$t4` 存入 `0x40000010` 地址中，每次存入后等待 1 ms，同时总体循环 `$a1` 次。以下是核心代码：

```

# test_display.asm
# Calculate device address offset
lui $t0, 0x4000 # $t0 = 0x40000000

# Set the 1ms count limit
addi $t5, $zero, 25000 # Given CLK_FREQ = 50 MHz

# Loop for $a1 times
addi $t6, $zero, 0
display_loop0:

sw $t1, 0x0010($t0)
addi $t7, $zero, 0
display_loop1:
addi $t7, $t7, 1
bne $t7, $t5, display_loop1

# ... ...

addi $t6, $t6, 1
bne $t6, $a1, display_loop0

```

同样地，与硬件控制动态显示相比，软件实现的控制更灵活，可以根据其他程序改变显示数字或显示时间。

最后，我们在主函数设置 0x01ef、0x23cd、0x45ab、0x6789 这 4 个数字来测试显示效果。

在调试中我发现一开始的汇编代码犯了一个错误：在被调用的 `display` 函数中调用函数 `map` 时，我忘记将返回地址 `$ra` 存到其他寄存器中导致其被刷新，结果数码管什么都显示不出来。由于外设地址超出了 MARS 软件规定的内存，我一开始并没有在 MARS 软件中模拟执行，这使我不得不重复综合、上板观察，排错效率极低。因此在后面的汇编代码编写过程中，我尽量使用 `lui $t0, 0x1001` 将地址先放到内存中排查代码错误，极大提升了开发体验，这也印证了那句“能软件仿真就不要硬件仿真，能仿真就不要烧板”

test_uart.asm

这个 MIPS 汇编程序需要通过软件方式控制 UART 串口收发数据。由于 UART 收发功能是由硬件代码实现的，因此汇编代码只需要 `lw` 和 `sw` 指令读写 UART 对应的地址即可。此外，由于 UART 是逐字节传输的，软件上还需要在 word 和字节之间进行拆分和合并。

对于接收功能，我编写了 `receive` 函数，其核心在于等待 0x40000020 地址的接收状态 (bit 3) 为真（意味着接收到一字节消息），然后读取 0x4000001C 地址的消息。这样重复 4 次，最后将接收到的 4 字节消息合并后返回到 `$v0`。部分代码如下：

```

# test_uart.asm
# Wait for a incoming message
receive_loop0:
lw $t1, 0x0020($t0)
andi $t1, $t1, 0x0008
beq $t1, $zero, receive_loop0

# Read the message
lw $t2, 0x001c($t0)
sll $t2, $t2, 0
addi $v0, $t2, 0

```

发送功能也是类似的，在创建的 `send` 函数中，先将参数 `$a0`（要发送的消息）拆分为 4 字节，然后分别写入 0x40000018 地址，每次写入后等待 0x40000020 地址的发送状态 (bit 2) 为真（意味着发送完毕）。部分代码如下：

```

# test_uart.asm
# Send the message
srl $t2, $a0, 24
sw $t2, 0x0018($t0)

# Wait for the message to be sent
send_loop0:
lw $t1, 0x0020($t0)
andi $t1, $t1, 0x0004
beq $t1, $zero, send_loop0

```

我们注意到这里接收的第 1 字节被储存在了低位，而发送的第 1 字节是参数的高位，中间特意进行了字节翻转来符合输入和输出易读性要求。

最后，我们在主函数中先调用 `receive` 函数，然后将返回值 `$v0` 拷贝到 `$a0` 中，再调用 `send` 函数。这样，我们就可以通过串口助手发送消息，然后观察板子上串口助手是否收到相同的消息（字节翻转），来验证 UART 串口收发功能是否正常。

sort.asm

排序算法使用了选择排序，代码和重点注释如下：

```

# sort.asm
main_entry:
lw $s0, 0($a0) # $s0 = N
sll $s0, $s0, 2
add $s0, $s0, $a0

# for (i = 1; i != N; i++)
addi $t0, $a0, 4 # $t0: i in address
main_loop0:

lw $t2, 0($s0) # $t2: smallest number
addi $t3, $s0, 0 # $t3: corresponding address

# for (j = N; j != i; j--)
addi $t1, $s0, 0 # $t1: j in address
main_loop1:
lw $t4, 0($t1)
# if ($t4 < $t2)
blt $t2, $t4, main_branch1
addi $t2, $t4, 0
addi $t3, $t1, 0
main_branch1:
addi $t1, $t1, -4
bne $t1, $t0, main_loop1

lw $t4, 0($t0)
# if ($t2 < $t4)
blt $t4, $t2, main_branch2
sw $t2, 0($t0)
sw $t4, 0($t3)

main_branch2:
addi $t0, $t0, 4
bne $t0, $s0, main_loop0

```

指导书要求报告这部分使用流水线处理器运行时的 CPI，以下是计算过程：

给定 N 为 20，数据与 pipeline/cpu/ram.mem 中的相同，使用 MARS 仿真得到一共运行了 1228 条指令。

使用之前设计的仿真测试文件，用 Vivado 进行行为级仿真，观察到第 2113 个时钟周期的 PC 值为 0x0054（最后一条指令的地址为 0x0050），说明程序运行结束。然而，由于流水线结构，最后一条程序指令在第 2113 个时钟周期仍在 ID 阶段，我们仍需等待 3 个时钟周期才真正完成所有指令。因此总周期数为 2116 个。

$$CPI = \frac{\# \text{ Clocks}}{\# \text{ Instructions}} = \frac{2116}{1228} = 1.723$$

我们也可以使用 MIPS 仿真结果结合流水线处理器的理论行为来验证 Vivado 仿真得到的周期数：

仿真得到程序一共执行了 418 条分支指令，经过人工统计一共有 81 条分支指令未发生跳转（比较过程 61 条、内层循环 19 条、外层循环 1 条），剩余的 337 条分支指令发生跳转。每条发生跳转的分支指令需要插入 2 条空指令来避免冒险，因此增加了 674 个时钟周期。

同时，第 8、22、31 行的 `lw` 指令触发了 load-use 冒险，每条指令后需要插入 1 条空指令，共计 $1 + 190 + 19 = 210$ 个时钟周期。

综上，运行时增加了 884 个时钟周期，加上程序运行的 1228 个时钟周期，得到总周期数为 2112 个，再加上流水线导致的 4 个首尾空余周期，就是 2116 个，与 Vivado 仿真结果一致。

sort_display.asm

这个程序将 `sort.asm` 和前面设计的 `display` 函数结合在一起，实现了读取内存中数据进行排序，并将排序结果显示在数码管上的功能。

sort_uart.asm

这个程序将 `sort.asm` 和前面设计的 `receive` 和 `send` 函数结合在一起，实现了通过 UART 串口接收数据进行排序，并将排序结果通过 UART 串口发送出去的功能。

值得一提的是，由于 `sort.asm` 和 `test_uart.asm` 都分别进行过实机测试并正常工作，我并没有对串联起来的 `sort_uart.asm` 进行仿真，结果到开发板上就无法正常工作了。后来经过几次排错，发现问题出在 `$t0` 寄存器在排序时发生变化，但在 `send` 时没有重新初始化，而是照搬了 `test_uart.asm`（只在主函数最前面初始化了一次）。这段排错经历说明了遵循 MIPS 寄存器分类的重要性，要在调用函数后默认所有临时寄存器都被修改，需要重新初始化。

sort_uart_display.asm

这是最终验收用到的程序。简单来讲就是在 `sort_uart.asm` 基础上，每个数同时调用 `send` 和 `display` 函数，将结果同时发送到串口和显示在数码管上。

仿真结果与分析

无论是单周期处理器还是流水线处理器，都通过了前面提到的仿真测试，可以支持 `README.md` 中提到的所有 MIPS 指令（包括拓展的 `bne`、`blez`、`bgtz`、`bltz` 和 `jalr` 指令）。

此外，流水线处理器也可以按要求处理各类冒险情形。

阅者可以自行验证，注意需要将 `InstructionMemory.v` 中的 `$readmemh` 文件名修改为 `test_instructions.mem`，并将 `test_instructions.mem` 添加到 Vivado 的设计源码中。

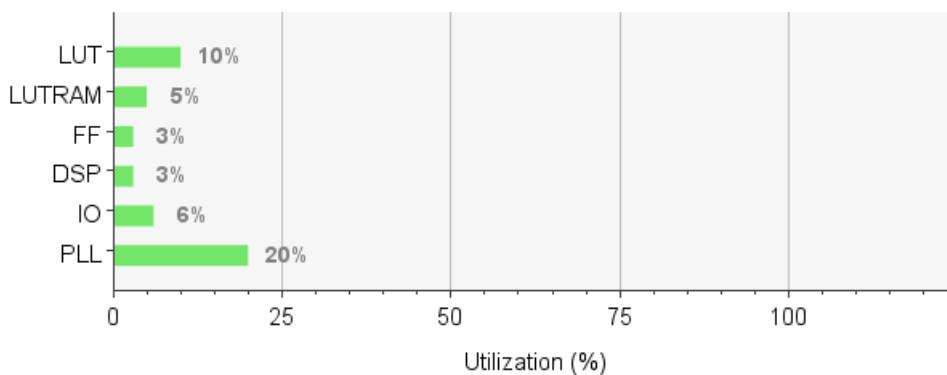
综合结果与分析

硬件资源使用情况

单周期处理器

Summary

| Resource | Utilization | Available | Utilization % |
|----------|-------------|-----------|---------------|
| LUT | 2039 | 20800 | 9.80 |
| LUTRAM | 512 | 9600 | 5.33 |
| FF | 1104 | 41600 | 2.65 |
| DSP | 3 | 90 | 3.33 |
| IO | 16 | 250 | 6.40 |
| PLL | 1 | 5 | 20.00 |



| Name | Slice LUTs (20800) | Slice Registers (41600) | F7 Muxes (16300) | F8 Muxes (8150) | Slice (8150) | LUT as Logic (20800) | LUT as Memory (9600) | DSPs (90) | Bonded IOB (250) | BUFGCTRL (32) | PLLE2_ADV (5) |
|---------------------|--------------------|-------------------------|------------------|-----------------|--------------|----------------------|----------------------|-----------|------------------|---------------|---------------|
| top | 2039 | 1104 | 529 | 200 | 837 | 1527 | 512 | 3 | 16 | 2 | 1 |
| clkwiz (clk_wiz_0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 1 |
| cpu1 (CPU) | 1966 | 1024 | 529 | 200 | 801 | 1454 | 512 | 3 | 0 | 0 | 0 |
| alu1 (ALU) | 15 | 0 | 0 | 0 | 4 | 15 | 0 | 3 | 0 | 0 | 0 |
| data_memory1 (L) | 547 | 0 | 256 | 128 | 161 | 35 | 512 | 0 | 0 | 0 | 0 |
| instruction_memory | 106 | 0 | 16 | 0 | 45 | 106 | 0 | 0 | 0 | 0 | 0 |
| register_file1 (Re) | 1297 | 992 | 257 | 72 | 666 | 1297 | 0 | 0 | 0 | 0 | 0 |
| device1 (Device) | 73 | 80 | 0 | 0 | 44 | 73 | 0 | 0 | 0 | 0 | 0 |
| uart1 (UART) | 73 | 60 | 0 | 0 | 37 | 73 | 0 | 0 | 0 | 0 | 0 |

可以看到系统总计使用了 2039 个 LUT 和 1104 个寄存器，其中用到了 512 个 LUTRAM，这是 FPGA 片上一种使用 LUT 实现高速随机存储的硬件资源，对应于我们数据内存的 1024 个 word。同时，系统还用到了 3 个 DSP 模块，这是 FPGA 用于高速信号处理的硬件资源，对应于 ALU 中的乘法操作。此外，系统还用到了 1 个 PLL，这是因为单周期处理器不能支持 100 MHz 的系统时钟，因此使用了频率合成的 50 MHz 时钟。

从各模块占用的硬件资源比例来看，RegisterFile 模块使用了 1297 个 LUT 和 992 个寄存器，占用了系统资源的绝大部分。其中 992 个寄存器对应着 31 个 32 位寄存器（寄存器 0 恒为 0，不实现为寄存器），而 1297 个 LUT 则实现了这 31 个寄存器在处理器中的各种数据路径。

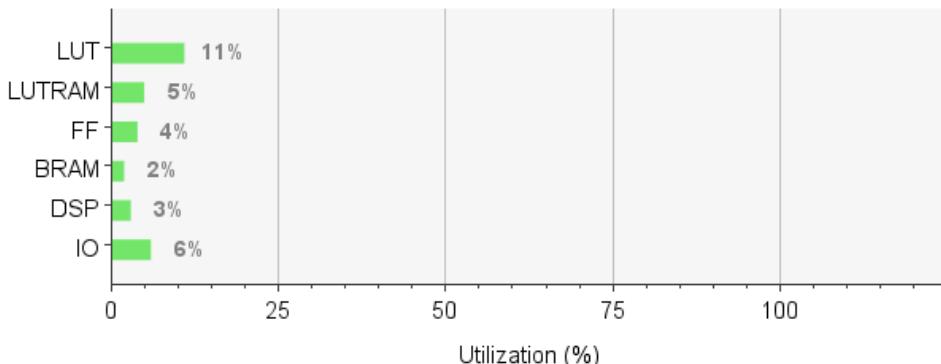
而通过减法，我们可以得出 CPU 模块自身只使用了 1 个 LUT 和 32 个寄存器，这对应着 PC 寄存器及其更新逻辑。这一现象与下面我们分析的流水线处理器形成了鲜明对比。

流水线处理器

对于流水线处理器，虽然我们对其时序性能进行了多次改进，但其硬件资源使用情况没有显著变化。这里我们选用其中具有代表性的一版（v1.2.2）进行分析：

Summary

| Resource | Utilization | Available | Utilization % |
|----------|-------------|-----------|---------------|
| LUT | 2224 | 20800 | 10.69 |
| LUTRAM | 512 | 9600 | 5.33 |
| FF | 1460 | 41600 | 3.51 |
| BRAM | 1 | 50 | 2.00 |
| DSP | 3 | 90 | 3.33 |
| IO | 16 | 250 | 6.40 |



| Hierarchy | | | | | | | | | | | | |
|--------------------------------|--------------------|-------------------------|------------------|-----------------|--------------|----------------------|----------------------|---------------------|-----------|------------------|---------------|--|
| Name | Slice LUTs (20800) | Slice Registers (41600) | F7 Muxes (16300) | F8 Muxes (8150) | Slice (8150) | LUT as Logic (20800) | LUT as Memory (9600) | Block RAM Tile (50) | DSPs (90) | Bonded IOB (250) | BUFGCTRL (32) | |
| N top | 2224 | 1460 | 393 | 192 | 942 | 1712 | 512 | 1 | 3 | 16 | 1 | |
| cpu1 (CPU) | 2148 | 1380 | 393 | 192 | 900 | 1636 | 512 | 1 | 3 | 0 | 0 | |
| alu1 (ALU) | 116 | 0 | 2 | 0 | 66 | 116 | 0 | 0 | 3 | 0 | 0 | |
| mult1 (multiplier) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | |
| data_memory1 | 588 | 0 | 256 | 128 | 182 | 76 | 512 | 0 | 0 | 0 | 0 | |
| instruction_memory1 | 387 | 0 | 0 | 0 | 172 | 387 | 0 | 1 | 0 | 0 | 0 | |
| register_file1 (Register File) | 607 | 992 | 128 | 64 | 475 | 607 | 0 | 0 | 0 | 0 | 0 | |
| device1 (Device) | 76 | 80 | 0 | 0 | 47 | 76 | 0 | 0 | 0 | 0 | 0 | |
| uart1 (UART) | 76 | 60 | 0 | 0 | 40 | 76 | 0 | 0 | 0 | 0 | 0 | |

可以看到系统总计使用了 2224 个 LUT (包括 512 个 LUTRAM)、1460 个寄存器、3 个 DSP 和 1 个 BRAM。

与单周期处理器相较，总体上 LUT 占用个数小幅增加，寄存器个数大幅增加，符合“用面积换频率”的理论预期。此外，由于可以在系统时钟下运行，因此不再需要用到 PLL；而因为将程序内存改写为了 BRAM（详见“设计方案”部分），因此多出了 1 个 BRAM。

当我们进一步查看各模块使用的硬件资源比例时，会发现 RegisterFile 模块使用的 LUT 数量从 1297 个锐减到 607 个，而 CPU 模块自身占用的 LUT 个数从 1 个增加到了 450 个，这是因为在流水线处理器中，RegisterFile 寄存器只负责将数据传递到 CPU 中的 ID_EX 寄存器中，剩余各阶段之间的传递逻辑则由 CPU 模块中的寄存器实现，这也对应了 CPU 模块使用的寄存器数量由 32 个增加到 388 个。

综上，可以认为 LUT 占用数量小幅增加是因为处理器逻辑功能保持不变、同时增加了如数据转发的数据路径；而寄存器数量大幅增加则主要来源于 CPU 模块中各阶段之间的寄存器。

静态时序分析结果

单周期处理器

对于单周期处理器，我们在 100 MHz 系统时钟下进行综合显示超出时间约束约 7 ns，数据路径最大延时在 17 ns 左右。因此，为了满足时序要求，我们选择在 50 MHz 的合成频率下进行综合，以下是静态时序分析结果：

Design Timing Summary

| Setup | Hold | Pulse Width |
|--------------------------------------|----------------------------------|---|
| Worst Negative Slack (WNS): 0.540 ns | Worst Hold Slack (WHS): 0.208 ns | Worst Pulse Width Slack (WPWS): 3.000 ns |
| Total Negative Slack (TNS): 0.000 ns | Total Hold Slack (THS): 0.000 ns | Total Pulse Width Negative Slack (TPWS): 0.000 ns |
| Number of Failing Endpoints: 0 | Number of Failing Endpoints: 0 | Number of Failing Endpoints: 0 |
| Total Number of Endpoints: 7265 | Total Number of Endpoints: 7265 | Total Number of Endpoints: 1622 |

All user specified timing constraints are met.

| Intra-Clock Paths - clk_out1_clk_wiz_0 - Setup | | | | | | | | | |
|--|-----------|--------|-------------|------------------|-------------------|-------------|-------------|-----------|-------------|
| Name | Slack ^ 1 | Levels | High Fanout | From | To | Total Delay | Logic Delay | Net Delay | Requirement |
| Path 1 | 0.540 | 28 | 257 | cpu1/PC_reg[5]/C | cpu1/PC_reg[30]/D | 19.334 | 9.644 | 9.690 | 20.0 |
| Path 2 | 0.572 | 28 | 257 | cpu1/PC_reg[5]/C | cpu1/PC_reg[31]/D | 19.303 | 9.582 | 9.721 | 20.0 |
| Path 3 | 0.609 | 27 | 257 | cpu1/PC_reg[5]/C | cpu1/PC_reg[26]/D | 19.264 | 9.546 | 9.718 | 20.0 |
| Path 4 | 0.644 | 27 | 257 | cpu1/PC_reg[5]/C | cpu1/PC_reg[27]/D | 19.300 | 9.484 | 9.816 | 20.0 |
| Path 5 | 0.662 | 26 | 257 | cpu1/PC_reg[5]/C | cpu1/PC_reg[23]/D | 19.281 | 9.386 | 9.895 | 20.0 |
| Path 6 | 0.689 | 27 | 257 | cpu1/PC_reg[5]/C | cpu1/PC_reg[25]/D | 19.182 | 9.460 | 9.722 | 20.0 |
| Path 7 | 0.705 | 26 | 257 | cpu1/PC_reg[5]/C | cpu1/PC_reg[21]/D | 19.167 | 9.362 | 9.805 | 20.0 |
| Path 8 | 0.714 | 26 | 257 | cpu1/PC_reg[5]/C | cpu1/PC_reg[22]/D | 19.229 | 9.448 | 9.781 | 20.0 |
| Path 9 | 0.718 | 28 | 257 | cpu1/PC_reg[5]/C | cpu1/PC_reg[29]/D | 19.156 | 9.558 | 9.598 | 20.0 |
| Path 10 | 0.757 | 27 | 257 | cpu1/PC_reg[5]/C | cpu1/PC_reg[28]/D | 19.161 | 9.548 | 9.613 | 20.0 |

可以看到，此时建立时间的裕量为 0.54 ns，对应数据路径最大延时为 19.334 ns，对应理论最高时钟频率为

$$f_{max} = \frac{1}{\text{period} - \text{slack}} = \frac{1}{20 \text{ ns} - 0.54 \text{ ns}} = 51.39 \text{ MHz}$$

我们可以进一步查看延时最大的这条路径，如下图所示：

| Path 1 - top_timing_summary_routed | | | | | | | |
|--|-----------|-----------|----------|--------------------|-----------|-----------------------------------|--|
| Path | Type | Incr (ns) | Path ... | Location | Cell Pin | Cell | Netlist Resources |
| FDCE (Prop_fdce_C_Q) | (r) 0.433 | -1.438 | | Site: SLICE_X50Y56 | Q | PC_reg[5] (FDCE) | cpu1/PC_reg[5]/Q |
| net (fo=76, routed) | | 1.088 | -0.350 | | | | cpu1/instruction_memory1/Q[3] |
| LUT6 (Prop_lut6_I1_O) | (r) 0.105 | -0.245 | | Site: SLICE_X49Y54 | O | PC[23]_i_5 (LUT6) | cpu1/instruction_memory1/PC[23]_i_5/O |
| net (fo=1, routed) | | 0.000 | -0.245 | | | | cpu1/instruction_memory1/PC[23]_i_5_n_0 |
| MUXF7 (Prop_muxf7_I1_O) | (r) 0.182 | -0.063 | | Site: SLICE_X49Y54 | O | PC_reg[23]_i_3 (MUXF7) | cpu1/instruction_memory1/PC_reg[23]_i_3/O |
| net (fo=1, routed) | | 0.344 | 0.281 | | | | cpu1/instruction_memory1/PC_reg[23]_i_3_n_0 |
| LUT6 (Prop_lut6_I4_O) | (r) 0.252 | 0.533 | | Site: SLICE_X49Y54 | O | PC[23]_i_2 (LUT6) | cpu1/instruction_memory1/PC[23]_i_2/O |
| net (fo=257, routed) | | 1.229 | 1.762 | | | | cpu1/register_file1/PC_Reg[23] |
| LUT6 (Prop_lut6_I4_O) | (r) 0.105 | 1.867 | | Site: SLICE_X57Y72 | O | out0_0_i_113 (LUT6) | cpu1/register_file1/out0_0_i_113/O |
| net (fo=1, routed) | | 0.000 | 1.867 | | | | cpu1/register_file1/out0_0_i_113_n_0 |
| MUXF7 (Prop_muxf7_I1_O) | (r) 0.182 | 2.049 | | Site: SLICE_X57Y72 | O | out0_0_i_41 (MUXF7) | cpu1/register_file1/out0_0_i_41/O |
| net (fo=1, routed) | | 0.639 | 2.688 | | | | cpu1/register_file1/out0_0_i_41_n_0 |
| LUT6 (Prop_lut6_I1_O) | (r) 0.252 | 2.940 | | Site: SLICE_X58Y70 | O | out0_0_i_18 (LUT6) | cpu1/register_file1/out0_0_i_18/O |
| net (fo=4, routed) | | 0.953 | 3.894 | | | | cpu1/register_file1/DataBus[16] |
| LUT2 (Prop_lut2_I0_O) | (r) 0.125 | 4.019 | | Site: SLICE_X60Y62 | O | out0_0_i_1 (LUT2) | cpu1/register_file1/out0_0_i_1/O |
| net (fo=8, routed) | | 0.742 | 4.760 | | | | cpu1/alu1/ALU_in1[16] |
| DSP48E1 (Prop_dsp48e1_A[16]_PCOUT[47]) | (r) 3.556 | 8.316 | | Site: DSP48_X1Y23 | PCOUT[47] | out0_0 (DSP48E1) | cpu1/alu1/out0_0/PCOUT[47] |
| net (fo=1, routed) | | 0.002 | 8.318 | | | | cpu1/alu1/out0_0_n_106 |
| DSP48E1 (Prop_dsp48e1_PCIN[47]_P[0]) | (r) 1.271 | 9.589 | | Site: DSP48_X1Y24 | P[0] | out0_1 (DSP48E1) | cpu1/alu1/out0_1/P[0] |
| net (fo=2, routed) | | 0.590 | 10.179 | | | | cpu1/alu1/out0_1_n_105 |
| LUT2 (Prop_lut2_I0_O) | (r) 0.105 | 10.284 | | Site: SLICE_X55Y60 | O | RF_data[31][19]_i_21 (LUT2) | cpu1/alu1/RF_data[31][19]_i_21/O |
| net (fo=1, routed) | | 0.000 | 10.284 | | | | cpu1/alu1/RF_data[31][19]_i_21_n_0 |
| CARRY4 (Prop_carry4_S[1]_CO[3]) | (r) 0.457 | 10.741 | | Site: SLICE_X55Y60 | CO[3] | RF_data_reg[31][19]_i_16 (CARRY4) | cpu1/alu1/RF_data_reg[31][19]_i_16/CO[3] |
| net (fo=1, routed) | | 0.000 | 10.741 | | | | cpu1/alu1/RF_data_reg[31][19]_i_16_n_0 |
| CARRY4 (Prop_carry4_CI_CO[0]) | (f) 0.180 | 10.921 | | Site: SLICE_X55Y61 | O[0] | RF_data_reg[31][23]_i_13 (CARRY4) | cpu1/alu1/RF_data_reg[31][23]_i_13/O[0] |
| net (fo=1, routed) | | 0.657 | 11.579 | | | | cpu1/register_file1/RF_data[31][23]_i_16_n_0 |
| LUT5 (Prop_lut5_I0_O) | (f) 0.249 | 11.828 | | Site: SLICE_X61Y61 | O | RF_data[31][20]_i_11 (LUT5) | cpu1/register_file1/RF_data[31][20]_i_11/O |
| net (fo=1, routed) | | 0.568 | 12.396 | | | | cpu1/register_file1/RF_data[31][20]_i_11_n_0 |
| LUT6 (Prop_lut6_I5_O) | (f) 0.105 | 12.501 | | Site: SLICE_X61Y67 | O | RF_data[31][20]_i_7 (LUT6) | cpu1/register_file1/RF_data[31][20]_i_7/O |
| net (fo=1, routed) | | 0.113 | 12.613 | | | | cpu1/register_file1/RF_data[31][20]_i_7_n_0 |
| LUT6 (Prop_lut6_I5_O) | (f) 0.105 | 12.718 | | Site: SLICE_X61Y67 | O | RF_data[31][20]_i_3 (LUT6) | cpu1/register_file1/RF_data[31][20]_i_3/O |
| net (fo=3, routed) | | 0.581 | 13.300 | | | | cpu1/register_file1/MemBus_Address[20] |
| LUT4 (Prop_lut4_I3_O) | (r) 0.105 | 13.405 | | Site: SLICE_X59Y68 | O | PC[31]_i_19 (LUT4) | cpu1/register_file1/PC[31]_i_19/O |
| net (fo=1, routed) | | 0.837 | 14.242 | | | | cpu1/register_file1/PC[31]_i_19_n_0 |
| LUT6 (Prop_lut6_I4_O) | (r) 0.105 | 14.347 | | Site: SLICE_X60Y63 | O | PC[31]_i_14 (LUT6) | cpu1/register_file1/PC[31]_i_14/O |
| net (fo=1, routed) | | 0.361 | 14.707 | | | | cpu1/register_file1/PC[31]_i_14_n_0 |
| LUT6 (Prop_lut6_I2_O) | (r) 0.105 | 14.812 | | Site: SLICE_X60Y63 | O | PC[31]_i_10 (LUT6) | cpu1/register_file1/PC[31]_i_10/O |
| net (fo=30, routed) | | 0.667 | 15.479 | | | | cpu1/register_file1/PC[31]_i_10_n_0 |
| LUT5 (Prop_lut5_I2_O) | (r) 0.105 | 15.584 | | Site: SLICE_X57Y59 | O | PC[4]_i_6 (LUT5) | cpu1/register_file1/PC[4]_i_6/O |
| net (fo=1, routed) | | 0.000 | 15.584 | | | | cpu1/register_file1/PC[4]_i_6_n_0 |
| CARRY4 (Prop_carry4_S[1]_CO[3]) | (r) 0.457 | 16.041 | | Site: SLICE_X57Y59 | CO[3] | PC_reg[4]_i_2 (CARRY4) | cpu1/register_file1/PC_reg[4]_i_2/CO[3] |
| net (fo=1, routed) | | 0.000 | 16.041 | | | | cpu1/register_file1/PC_reg[4]_i_2_n_0 |
| CARRY4 (Prop_carry4_CI_CO[3]) | (r) 0.098 | 16.139 | | Site: SLICE_X57Y60 | CO[3] | PC_reg[8]_i_2 (CARRY4) | cpu1/register_file1/PC_reg[8]_i_2/CO[3] |
| net (fo=1, routed) | | 0.000 | 16.139 | | | | cpu1/register_file1/PC_reg[8]_i_2_n_0 |
| CARRY4 (Prop_carry4_CI_CO[3]) | (r) 0.098 | 16.237 | | Site: SLICE_X57Y61 | CO[3] | PC_reg[12]_i_2 (CARRY4) | cpu1/register_file1/PC_reg[12]_i_2/CO[3] |
| net (fo=1, routed) | | 0.000 | 16.237 | | | | cpu1/register_file1/PC_Reg[12]_i_2_n_0 |
| CARRY4 (Prop_carry4_CI_CO[3]) | (r) 0.098 | 16.335 | | Site: SLICE_X57Y62 | CO[3] | PC_reg[16]_i_2 (CARRY4) | cpu1/register_file1/PC_Reg[16]_i_2/CO[3] |
| net (fo=1, routed) | | 0.000 | 16.335 | | | | cpu1/register_file1/PC_Reg[16]_i_2_n_0 |
| CARRY4 (Prop_carry4_CI_CO[3]) | (r) 0.098 | 16.433 | | Site: SLICE_X57Y63 | CO[3] | PC_reg[20]_i_2 (CARRY4) | cpu1/register_file1/PC_Reg[20]_i_2/CO[3] |
| net (fo=1, routed) | | 0.000 | 16.433 | | | | cpu1/register_file1/PC_Reg[20]_i_2_n_0 |
| CARRY4 (Prop_carry4_CI_CO[3]) | (r) 0.098 | 16.531 | | Site: SLICE_X57Y64 | CO[3] | PC_reg[24]_i_2 (CARRY4) | cpu1/register_file1/PC_Reg[24]_i_2/CO[3] |
| net (fo=1, routed) | | 0.000 | 16.531 | | | | cpu1/register_file1/PC_Reg[24]_i_2_n_0 |
| CARRY4 (Prop_carry4_CI_CO[3]) | (r) 0.098 | 16.629 | | Site: SLICE_X57Y65 | CO[3] | PC_reg[28]_i_2 (CARRY4) | cpu1/register_file1/PC_Reg[28]_i_2/CO[3] |
| net (fo=1, routed) | | 0.000 | 16.629 | | | | cpu1/register_file1/PC_Reg[28]_i_2_n_0 |
| CARRY4 (Prop_carry4_CI_O[1]) | (r) 0.265 | 16.894 | | Site: SLICE_X57Y66 | O[1] | PC_reg[31]_i_4 (CARRY4) | cpu1/register_file1/PC_Reg[31]_i_4/O[1] |
| net (fo=1, routed) | | 0.319 | 17.213 | | | | cpu1/register_file1/Branch_target[30] |
| LUT5 (Prop_lut5_I2_O) | (r) 0.250 | 17.463 | | Site: SLICE_X57Y67 | O | PC[30]_i_1 (LUT5) | cpu1/register_file1/PC[30]_i_1/O |
| net (fo=1, routed) | | 0.000 | 17.463 | | | | cpu1/p_0_in_[30] |
| FDCE | | | | Site: SLICE_X57Y67 | D | PC_reg[30] (FDCE) | cpu1/PC_reg[30]/D |
| Arrival Time | | | | | | | |
| | | 17.463 | | | | | |

可以看到路径从 PC 寄存器出来，经过取指、译码、执行等流程后回到 PC 寄存器，符合单周期处理器的特征。从倒数第 4 行的 Branch_target 可以看出，这是与分支跳转有关的一条路径。然而，由于在单周期处理器中所有指令——无论多复杂——都需要在一个周期内执行完，所以我们对此也没有什么优化的空间。

由于在单周期处理器中我们没有将程序内存实现为 BRAM，Vivado 会将其实现为数量可变的 LUT（指令越多、数量越多）。我们发现这样一来，不同的汇编代码综合得到的硬件资源占用和时序性能之间会有差异。这里给出的结果是基于我所编写的 program 文件夹中的几份汇编代码进行综合的结果，更长或更复杂的汇编代码可能会有更差的时序性能。

流水线处理器

对于流水线处理器，我们先将频率目标设定在系统时钟的 100 MHz。这，是我们刚实现完处理器所有功能时得到的时序性能：

Version: v1.0

Commit Hash: 8d2615f

Design Timing Summary

| Setup | Hold | Pulse Width |
|---|---|---|
| Worst Negative Slack (WNS): -4.053 ns | Worst Hold Slack (WHS): 0.078 ns | Worst Pulse Width Slack (WPWS): 3.870 ns |
| Total Negative Slack (TNS): -1195.748 ns | Total Hold Slack (THS): 0.000 ns | Total Pulse Width Negative Slack (TPWS): 0.000 ns |
| Number of Failing Endpoints: 359 | Number of Failing Endpoints: 0 | Number of Failing Endpoints: 0 |
| Total Number of Endpoints: 7774 | Total Number of Endpoints: 7774 | Total Number of Endpoints: 2023 |

Timing constraints are not met.

| Intra-Clock Paths - clock - Setup | | | | | | | | | |
|-----------------------------------|---------------|--------|-------------|-------------------------------|-----------------------------------|-------------|-------------|-----------|-------------|
| Name | Slack | Levels | High Fanout | From | To | Total Delay | Logic Delay | Net Delay | Requirement |
| Path 1 | -4.053 | 18 | 104 | cpu1/MEM_WB_R...reg_replica/C | cpu1/IF_ID_Instr...eg[17]_rep_0/D | 13.858 | 7.425 | 6.433 | 10.0 |
| Path 2 | -4.014 | 18 | 104 | cpu1/MEM_WB_R...reg_replica/C | cpu1/IF_ID_Instr...eg[17]_rep/CE | 13.690 | 7.425 | 6.265 | 10.0 |
| Path 3 | -4.014 | 18 | 104 | cpu1/MEM_WB_R...reg_replica/C | cpu1/IF_ID_Instruction_reg[20]/CE | 13.690 | 7.425 | 6.265 | 10.0 |
| Path 4 | -4.014 | 18 | 104 | cpu1/MEM_WB_R...reg_replica/C | cpu1/IF_ID_Instruction_reg[7]/CE | 13.690 | 7.425 | 6.265 | 10.0 |
| Path 5 | -4.014 | 18 | 104 | cpu1/MEM_WB_R...reg_replica/C | cpu1/IF_ID_Instruction_reg[8]/CE | 13.690 | 7.425 | 6.265 | 10.0 |
| Path 6 | -4.014 | 18 | 104 | cpu1/MEM_WB_R...reg_replica/C | cpu1/IF_ID_PC_plus_4_reg[2]/CE | 13.690 | 7.425 | 6.265 | 10.0 |
| Path 7 | -4.014 | 18 | 104 | cpu1/MEM_WB_R...reg_replica/C | cpu1/IF_ID_PC_plus_4_reg[5]/CE | 13.690 | 7.425 | 6.265 | 10.0 |
| Path 8 | -4.014 | 18 | 104 | cpu1/MEM_WB_R...reg_replica/C | cpu1/IF_ID_PC_plus_4_reg[7]/CE | 13.690 | 7.425 | 6.265 | 10.0 |
| Path 9 | -4.014 | 18 | 104 | cpu1/MEM_WB_R...reg_replica/C | cpu1/IF_ID_PC_plus_4_reg[9]/CE | 13.690 | 7.425 | 6.265 | 10.0 |
| Path 10 | -3.993 | 18 | 104 | cpu1/MEM_WB_R...reg_replica/C | cpu1/IF_ID_PC_plus_4_reg[23]/D | 13.793 | 7.425 | 6.368 | 10.0 |

可以看到建立时间超出时间约束 4.053 ns，最大延时为 13.858 ns，距离目标还有较大差距。

让我们来看看是哪条路径拖了后腿？

| Path 1 - top_timing_summary_routed | | | | | | | | |
|--------------------------------------|-----------|-----------|--------------------|-----------|---|------------------------------------|--|--|
| Data Path | | Incr (ns) | Path ... | Location | Cell Pin | Cell | Netlist Resources | |
| FDCE (Prop_fdce_C_Q) | (r) 0.379 | 4.951 | Site: SLICE_X49Y37 | Q | MEM_WB_RegWrite_reg_replica (FDCE) | cpu1/MEM_WB_RegWrite_reg_replica/Q | cpu1/alu1/MEM_WB_RegWrite_repln_alias | |
| net (fo=1, routed) | 0.674 | 5.624 | | | | | cpu1/alu1/out0_i_37/O | |
| LUT6 (Prop_lut6_I0_Q) | (r) 0.105 | 5.729 | Site: SLICE_X48Y35 | O | out0_i_37 (LUT6) | | cpu1/alu1/forwarding_unit1/p_7_in | |
| net (fo=2, routed) | 0.333 | 6.062 | | | | | cpu1/alu1/EX_MEMORY_Database2[31]_I_2/O | |
| LUT6 (Prop_lut6_I0_Q) | (r) 0.105 | 6.167 | Site: SLICE_X53Y35 | O | EX_MEMORY_Database2[31]_I_2 (LUT6) | | cpu1/alu1/EX_MEMORY_Database2[31]_I_2/O | |
| net (fo=32, routed) | 0.606 | 6.774 | | | | | cpu1/alu1/forwarding_unit1/ForwardB0 | |
| LUT5 (Prop_lut5_I2_Q) | (r) 0.105 | 6.879 | Site: SLICE_X52Y37 | O | EX_MEMORY_Database2[10]_I_1 (LUT5) | | cpu1/alu1/EX_MEMORY_Database2[10]_I_1/O | |
| net (fo=2, routed) | 0.464 | 7.343 | | | | | cpu1/alu1/ID_EX_Database2_reg[31][10] | |
| LUT3 (Prop_lut3_I2_Q) | (r) 0.105 | 7.448 | Site: SLICE_X53Y37 | O | out0_i_22 (LUT3) | | cpu1/alu1/out0_i_22/O | |
| net (fo=17, routed) | 0.556 | 8.004 | | | | | cpu1/alu1/ALU_in2[10] | |
| DSP48E1 (Prop_dsp_B[10]_PCOUT[47]) | (r) 3.244 | 11.248 | Site: DSP48_X1Y17 | PCOUT[47] | out0_0 (DSP48E1) | | cpu1/alu1/out0_0/PCOUT[47] | |
| net (fo=1, routed) | 0.002 | 11.250 | | | | | cpu1/alu1/out0_0_n_106 | |
| DSP48E1 (Prop_dsp48e1_PCIN[47]_P[0]) | (r) 1.271 | 12.521 | Site: DSP48_X1Y18 | P[0] | out0_1 (DSP48E1) | | cpu1/alu1/out0_1/P[0] | |
| net (fo=2, routed) | 0.505 | 13.026 | | | | | cpu1/alu1/out0_1_n_105 | |
| LUT2 (Prop_lut2_I0_Q) | (r) 0.105 | 13.131 | Site: SLICE_X54Y44 | O | EX_MEMORY_ALU_out[19]_I_19 (LUT2) | | cpu1/alu1/EX_MEMORY_ALU_out[19]_I_19/O | |
| net (fo=1, routed) | 0.000 | 13.131 | | | | | cpu1/alu1/EX_MEMORY_ALU_out[19]_I_19_n_0 | |
| CARRY4 (Prop_carry4_S[1]_CO[3]) | (r) 0.444 | 13.575 | Site: SLICE_X54Y44 | CO[3] | EX_MEMORY_ALU_out_reg[19]_I_14 (CARRY4) | | cpu1/alu1/EX_MEMORY_ALU_out_reg[19]_I_14/CO[3] | |
| net (fo=1, routed) | 0.000 | 13.575 | | | | | cpu1/alu1/EX_MEMORY_ALU_out_reg[19]_I_14_n_0 | |
| CARRY4 (Prop_carry4_CI_CO[3]) | (r) 0.100 | 13.675 | Site: SLICE_X54Y45 | CO[3] | EX_MEMORY_ALU_out_reg[23]_I_11 (CARRY4) | | cpu1/alu1/EX_MEMORY_ALU_out_reg[23]_I_11/CO[3] | |
| net (fo=1, routed) | 0.000 | 13.675 | | | | | cpu1/alu1/EX_MEMORY_ALU_out_reg[23]_I_11_n_0 | |
| CARRY4 (Prop_carry4_CI_CO[3]) | (r) 0.100 | 13.775 | Site: SLICE_X54Y46 | CO[3] | EX_MEMORY_ALU_out_reg[27]_I_12 (CARRY4) | | cpu1/alu1/EX_MEMORY_ALU_out_reg[27]_I_12/CO[3] | |
| net (fo=1, routed) | 0.000 | 13.775 | | | | | cpu1/alu1/EX_MEMORY_ALU_out_reg[27]_I_12_n_0 | |
| CARRY4 (Prop_carry4_CI_O[3]) | (f) 0.262 | 14.037 | Site: SLICE_X54Y47 | O[3] | EX_MEMORY_ALU_out_reg[31]_I_11 (CARRY4) | | cpu1/alu1/EX_MEMORY_ALU_out_reg[31]_I_11/O[3] | |
| net (fo=1, routed) | 0.360 | 14.397 | | | | | cpu1/alu1/EX_MEMORY_ALU_out_reg[31]_I_11_n_4 | |
| LUT5 (Prop_lut5_I0_Q) | (f) 0.250 | 14.647 | Site: SLICE_X53Y47 | O | EX_MEMORY_ALU_out[31]_I_7 (LUT5) | | cpu1/alu1/EX_MEMORY_ALU_out[31]_I_7/O | |
| net (fo=1, routed) | 0.229 | 14.876 | | | | | cpu1/alu1/EX_MEMORY_ALU_out[31]_I_7_n_0 | |
| LUT6 (Prop_lut6_I5_Q) | (f) 0.105 | 14.981 | Site: SLICE_X53Y46 | O | EX_MEMORY_ALU_out[31]_I_3 (LUT6) | | cpu1/alu1/EX_MEMORY_ALU_out[31]_I_3/O | |
| net (fo=1, routed) | 0.000 | 14.981 | | | | | cpu1/alu1/EX_MEMORY_ALU_out[31]_I_3_n_0 | |
| MUXF7 (Prop_muxf_I0_Q) | (f) 0.178 | 15.159 | Site: SLICE_X63Y46 | O | EX_MEMORY_ALU_out_reg[31]_I_1 (MUXF7) | | cpu1/alu1/EX_MEMORY_ALU_out_reg[31]_I_1/O | |
| net (fo=5, routed) | 0.521 | 15.680 | | | | | cpu1/alu1/ALU_out[31] | |
| LUT2 (Prop_lut2_I1_Q) | (r) 0.252 | 15.932 | Site: SLICE_X49Y43 | O | ID_EX_Instruction[31]_I_4 (LUT2) | | cpu1/alu1/ID_EX_Instruction[31]_I_4/O | |
| net (fo=3, routed) | 0.495 | 16.427 | | | | | cpu1/alu1/ID_EX_Instruction[31]_I_4_n_0 | |
| LUT4 (Prop_lut4_I3_Q) | (r) 0.105 | 16.532 | Site: SLICE_X48Y45 | O | ID_EX_Instruction[31]_I_2_comp_1 (LUT4) | | cpu1/alu1/ID_EX_Instruction[31]_I_2_comp_1/O | |
| net (fo=1, routed) | 0.313 | 16.845 | | | | | cpu1/alu1/ID_EX_Instruction[31]_I_2_n_0_repn | |
| LUT6 (Prop_lut6_I4_Q) | (r) 0.105 | 16.950 | Site: SLICE_X49Y45 | O | IF_ID_Instruction[31]_I_3_comp_1 (LUT6) | | cpu1/alu1/IF_ID_Instruction[31]_I_3_comp_1/O | |
| net (fo=104, routed) | 0.652 | 17.602 | | | | | cpu1/alu1/ID_EX_Instruction[28] | |
| LUT5 (Prop_lut5_I2_Q) | (r) 0.105 | 17.707 | Site: SLICE_X46Y45 | O | IF_ID_Instruction[17]_rep_0_i_1 (LUT5) | | cpu1/alu1/IF_ID_Instruction[17]_rep_0_i_1/O | |
| net (fo=1, routed) | 0.722 | 18.429 | | | | | cpu1/alu1_n_344 | |
| FDCE | | | Site: SLICE_X48Y50 | D | IF_ID_Instruction[17]_rep_0 (FDCE) | | cpu1/IF_ID_Instruction[17]_rep_0/D | |
| Arrival Time | | | | | | 18.429 | | |

乍一看有点迷茫。可以看出来路径先从数据转发开始，接着进入了 ALU 进行计算，可到后面是怎么与

`ID_EX_Instruction` 和 `IF_ID_Instruction` 扯上关系的呢？这就要回到源代码了。我们经过查找发现

`ID_EX_Instruction[31]` 只在判断分支条件的时候被用到了：

```
// CPU.v
wire Branch_condition;
assign Branch_condition =
  (ID_EX_Instruction[31:26] == 6'h04 && Zero) ||
  (ID_EX_Instruction[31:26] == 6'h05 && !Zero) ||
  (ID_EX_Instruction[31:26] == 6'h06 && (ALU_out[31] == 1'b1 || Zero)) ||
  (ID_EX_Instruction[31:26] == 6'h07 && (ALU_out[31] == 1'b0 && !zero)) ||
  (ID_EX_Instruction[31:26] == 6'h01 && (ALU_out[31] == 1'b1));
```

而既要用到 `Branch_condition`、又与 `IF_ID_Instruction` 有关的，就只有 `HazardUnit` 模块中 `IF_ID_Flush` 的判断和执行了。

因此，我们可以拼凑出来这条路径就是先判断是否需要数据转发、然后经过 ALU 计算、再判断分支条件、最后撤销 IF/ID 阶段寄存器的值，属于 EX 阶段的数据通路。

那么这条路径是否有可以优化的地方呢？对比 EX 阶段常规的数据路径，我们发现后两部分延长了数据通路，是额外增加的计算量。然而，分支条件的判断实际上计算相对简单，实际上并不需要引入 ALU 的减法操作以及相应的计算延时。因此，我想到可以将分支条件的判断和 ALU 计算在硬件上并行，用多出的电路面积换时间。

修改后的分支判断代码如下：

```
// CPU.v
wire Branch_condition;
assign Branch_condition =
  (ID_EX_Instruction[31:26] == 6'h04 && Databus1_forwarded == Databus2_forwarded) ||
  (ID_EX_Instruction[31:26] == 6'h05 && Databus1_forwarded != Databus2_forwarded) ||
  (ID_EX_Instruction[31:26] == 6'h06 && (Databus1_forwarded[31] == 1'b1 ||
  Databus1_forwarded == 0)) ||
  (ID_EX_Instruction[31:26] == 6'h07 && (Databus1_forwarded[31] == 1'b0 &&
  Databus1_forwarded != 0)) ||
  (ID_EX_Instruction[31:26] == 6'h01 && Databus1_forwarded[31] == 1'b1);
```

除此之外，这一版实际还取消了 `Branch` 控制信号，因为只要 `Branch_condition` 为真，`Branch` 一定为真。因此可以用 `Branch_condition` 替代所有的 `Branch && Branch_condition`

经过修改后的代码综合后时序性能如下：

Version: v1.1

Commit Hash: 51d3344

Design Timing Summary

| Setup | | Hold | | Pulse Width | |
|------------------------------|------------|------------------------------|----------|---|----------|
| Worst Negative Slack (WNS): | -0.919 ns | Worst Hold Slack (WHS): | 0.091 ns | Worst Pulse Width Slack (WPWS): | 3.870 ns |
| Total Negative Slack (TNS): | -11.336 ns | Total Hold Slack (THS): | 0.000 ns | Total Pulse Width Negative Slack (TPWNS): | 0.000 ns |
| Number of Failing Endpoints: | 28 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 7752 | Total Number of Endpoints: | 7752 | Total Number of Endpoints: | 2009 |

Timing constraints are not met.

| Intra-Clock Paths - clock - Setup | | | | | | | |
|-----------------------------------|-----------|--------|-------------|-------------------------------|-------------------------------|-------------|-------------|
| Name | Slack ^ 1 | Levels | High Fanout | From | To | Total Delay | Logic Delay |
| Path 1 | -0.919 | 13 | 76 | cpu1/MEM_WB_R...reg_replica/C | cpu1/PC_reg[31]/D | 10.894 | 6.819 |
| Path 2 | -0.696 | 11 | 76 | cpu1/MEM_WB_R...reg_replica/C | cpu1/EX_MEM_ALU_out_reg[23]/D | 10.689 | 6.619 |
| Path 3 | -0.693 | 12 | 76 | cpu1/MEM_WB_R...reg_replica/C | cpu1/EX_MEM_ALU_out_reg[26]/D | 10.650 | 6.650 |
| Path 4 | -0.621 | 11 | 76 | cpu1/MEM_WB_R...reg_replica/C | cpu1/EX_MEM_ALU_out_reg[24]/D | 10.576 | 6.518 |
| Path 5 | -0.608 | 11 | 76 | cpu1/MEM_WB_R...reg_replica/C | cpu1/PC_reg[25]/D | 10.625 | 6.604 |
| Path 6 | -0.598 | 11 | 76 | cpu1/MEM_WB_R...reg_replica/C | cpu1/EX_MEM_ALU_out_reg[25]/D | 10.571 | 6.604 |
| Path 7 | -0.542 | 13 | 76 | cpu1/MEM_WB_R...reg_replica/C | cpu1/EX_MEM_ALU_out_reg[31]/D | 10.546 | 6.892 |
| Path 8 | -0.535 | 11 | 76 | cpu1/MEM_WB_R...reg_replica/C | cpu1/PC_reg[24]/D | 10.491 | 6.518 |
| Path 9 | -0.534 | 11 | 76 | cpu1/MEM_WB_R...reg_replica/C | cpu1/EX_MEM_ALU_out_reg[21]/D | 10.535 | 6.609 |
| Path 10 | -0.523 | 13 | 76 | cpu1/MEM_WB_R...reg_replica/C | cpu1/EX_MEM_ALU_out_reg[29]/D | 10.482 | 6.809 |

可以看到延时普遍减少了 3 ns 左右，建立时间的超时最大只剩 0.919 ns，成效显著。

下面，我们重复刚才的方法来分析延时最大的路径，如下图所示：

| Path 1 - top_timing_summary_routed | | | | | | | |
|--------------------------------------|-----------|-----------|--------------------|-----------|----------|--------------------------------------|--|
| Data Path | | Incr (ns) | Path ... | Location | Cell Pin | Cell | Netlist Resources |
| FDCE (Prop_fdce_C_Q) | (r) 0.379 | 4.942 | Site: SLICE_X53Y29 | Q | | MEM_WB_RegWrite_reg_replica (FDCE) | cpu1/MEM_WB_RegWrite_reg_replica/Q |
| net (fo=1, routed) | 0.658 | 5.599 | | | | | cpu1/alu1/MEM_WB_RegWrite_repN_alias |
| LUT6 (Prop_lut6_10_Q) | (r) 0.105 | 5.704 | Site: SLICE_X51Y29 | O | | out0_0_i_27 (LUT6) | cpu1/alu1/out0_0_i_27/O |
| net (fo=2, routed) | 0.472 | 6.177 | | | | | cpu1/alu1/forwarding_unit/p_z_in |
| LUT6 (Prop_lut6_10_Q) | (r) 0.105 | 6.282 | Site: SLICE_X51Y27 | O | | out0_0_i_19 (LUT6) | cpu1/alu1/out0_0_i_19/O |
| net (fo=68, routed) | 0.589 | 6.870 | | | | | cpu1/alu1/ForwardA0 |
| LUT5 (Prop_lut5_11_Q) | (r) 0.105 | 6.975 | Site: SLICE_X53Y24 | O | | out0_0_i_24 (LUT5) | cpu1/alu1/out0_0_i_24/O |
| net (fo=6, routed) | 0.277 | 7.253 | | | | | cpu1/alu1/ID_EX_Databus1_reg[0] |
| LUT3 (Prop_lut3_2_Q) | (r) 0.105 | 7.358 | Site: SLICE_X52Y25 | O | | out0_0_i_17 (LUT3) | cpu1/alu1/out0_0_i_17/O |
| net (fo=76, routed) | 0.380 | 7.737 | | | | | cpu1/alu1/ALU_in1[0] |
| DSP48E1 (Prop_dsp..._A[0]_PCOUT[47]) | (r) 3.397 | 11.134 | Site: DSP48_X1Y10 | PCOUT[47] | | out0_0 (DSP48E1) | cpu1/alu1/out0_0_ip/PCOUT[47] |
| net (fo=1, routed) | 0.002 | 11.136 | | | | | cpu1/alu1/out0_0_n_106 |
| DSP48E1 (Prop_dsp48e1_PCIN[47]_P[0]) | (r) 1.271 | 12.407 | Site: DSP48_X1Y11 | P[0] | | out0_1 (DSP48E1) | cpu1/alu1/out0_1_P[0] |
| net (fo=2, routed) | 0.640 | 13.047 | | | | | cpu1/alu1/out0_1_n_105 |
| CARRY4 (Prop_carry4_DL[1]_CO[3]) | (r) 0.430 | 13.477 | Site: SLICE_X54Y27 | CO[3] | | EX_MEM_ALU_out_reg[19]_i_10 (CARRY4) | cpu1/alu1/EX_MEM_ALU_out_reg[19]_i_10/CO[3] |
| net (fo=1, routed) | 0.000 | 13.477 | | | | | cpu1/alu1/EX_MEM_ALU_out_reg[19]_i_10_n_0 |
| CARRY4 (Prop_carry4_CL_CO[3]) | (r) 0.100 | 13.577 | Site: SLICE_X54Y28 | CO[3] | | EX_MEM_ALU_out_reg[23]_i_14 (CARRY4) | cpu1/alu1/EX_MEM_ALU_out_reg[23]_i_14/CO[3] |
| net (fo=1, routed) | 0.000 | 13.577 | | | | | cpu1/alu1/EX_MEM_ALU_out_reg[23]_i_14_n_0 |
| CARRY4 (Prop_carry4_CL_CO[3]) | (r) 0.100 | 13.677 | Site: SLICE_X54Y29 | CO[3] | | EX_MEM_ALU_out_reg[27]_i_12 (CARRY4) | cpu1/alu1/EX_MEM_ALU_out_reg[27]_i_12/CO[3] |
| net (fo=1, routed) | 0.000 | 13.677 | | | | | cpu1/alu1/EX_MEM_ALU_out_reg[27]_i_12_n_0 |
| CARRY4 (Prop_carry4_CL_O[3]) | (r) 0.262 | 13.939 | Site: SLICE_X54Y30 | O[3] | | EX_MEM_ALU_out_reg[31]_i_11 (CARRY4) | cpu1/alu1/EX_MEM_ALU_out_reg[31]_i_11/O[3] |
| net (fo=1, routed) | 0.524 | 14.463 | | | | | cpu1/alu1/EX_MEM_ALU_out_reg[31]_i_11_n_4 |
| LUT5 (Prop_lut5_10_Q) | (r) 0.250 | 14.713 | Site: SLICE_X51Y34 | O | | EX_MEM_ALU_out[31]_i_7 (LUT5) | cpu1/alu1/EX_MEM_ALU_out[31]_i_7/O |
| net (fo=1, routed) | 0.113 | 14.825 | | | | | cpu1/alu1/EX_MEM_ALU_out[31]_i_7_n_0 |
| LUT6 (Prop_lut6_15_Q) | (r) 0.105 | 14.930 | Site: SLICE_X51Y34 | O | | EX_MEM_ALU_out[31]_i_3 (LUT6) | cpu1/alu1/EX_MEM_ALU_out[31]_i_3/O |
| net (fo=2, routed) | 0.422 | 15.352 | | | | | cpu1/data_memory1/ID_EX_ALUOp_reg[1]_0_alias |
| LUT5 (Prop_lut5_13_Q) | (r) 0.105 | 15.457 | Site: SLICE_X51Y36 | O | | PC[31]_i_2_comp (LUT5) | cpu1/data_memory1/PC[31]_i_2_compi/O |
| net (fo=1, routed) | 0.000 | 15.457 | | | | | cpu1/data_memory1_n_0 |
| FDCE | | | Site: SLICE_X51Y36 | D | | PC_reg[31] (FDCE) | cpu1/PC_reg[31]/D |
| Arrival Time | | 15.457 | | | | | |

可以看到数据路径短多了，也很清晰。这又是一条 EX 阶段的数据通路，从数据转发开始，到 ALU 计算，最后存到 PC 寄存器中。等等，为什么会存到 PC 寄存器中？查找源代码我们发现这是由于跳转指令的数据转发造成的：当 `jr` 或 `jalr` 指令指定的跳转寄存器在其上一条指令被写入，就需要从 ALU 的输出转发。

前面也是转发、后面也是转发，如果 ALU 计算那么费时，我们能不能把转发任务移到其他阶段，平均一下延时呢？我发现是可以的，请观察下面的代码：

```
// CPU.v
wire [32 - 1: 0] Databus1_forwarded;
assign Databus1_forwarded =
  (ForwardA == 2'b10)? EX_MEM_ALU_out:
  (ForwardA == 2'b01)? MEM_WB_Databus3: ID_EX_Databus1;

assign ALU_in1 = ID_EX_ALUSrc1? {27'h000000, ID_EX_Instruction[10:6]}:
Databus1_forwarded;
```

发现了什么？数据都是从阶段之间的寄存器转发过来的，这意味着这些数据在上一周期就已经被计算出来了，因此，我们完全可以把这部分代码挪到 ID 阶段的末尾，并与跳转指令的转发代码合并。以下是修改后的代码摘要：

```
// CPU.v
wire [32 - 1: 0] Databus1_forwarded;
assign Databus1_forwarded =
  (ForwardA == 2'b10)? ALU_out:
  (ForwardA == 2'b01)? Databus3: Databus1;

always @(`posedge clk or posedge reset) begin
  if (reset || ID_EX_Flush) begin
    ID_EX_Databus1 <= 32'h00000000;
  end
  else begin
    ID_EX_Databus1 <= Databus1_forwarded;
  end
end
```

```

end

assign ALU_in1 = ID_EX_ALUSrc1? {27'h00000, ID_EX_Instruction[10:6]}: ID_EX_Databus1;

```

```

// ForwardingUnit.v
always @(*) begin
    // Forward A
    if (ID_EX_RegWrite && (ID_EX_Write_register != 0) && (ID_EX_Write_register == IF_ID_RS))
        ForwardA <= 2'b10; // Forward from EX Stage
    else if (EX_MEM_RegWrite && (EX_MEM_Write_register != 0) && (EX_MEM_Write_register == IF_ID_RS))
        ForwardA <= 2'b01; // Forward from MEM Stage
    else
        ForwardA <= 2'b00; // No forwarding
    // ...
end

```

其中 ForwardingUnit 判断的控制信号均提前一个周期。

经过修改后的代码综合后时序性能如下：

Version: v1.2

Commit Hash: af8779c

Design Timing Summary

| Setup | | Hold | | Pulse Width | | |
|------------------------------|----------|------------------------------|----------|--|----------|--|
| Worst Negative Slack (WNS): | 0.103 ns | Worst Hold Slack (WHS): | 0.085 ns | Worst Pulse Width Slack (WPWS): | 3.870 ns | |
| Total Negative Slack (TNS): | 0.000 ns | Total Hold Slack (THS): | 0.000 ns | Total Pulse Width Negative Slack (TPWS): | 0.000 ns | |
| Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | |
| Total Number of Endpoints: | 7757 | Total Number of Endpoints: | 7757 | Total Number of Endpoints: | 2010 | |

All user specified timing constraints are met.

| Intra-Clock Paths - clock - Setup | | | | | | | | | |
|-----------------------------------|-----------|--------|-------------|-------------------------------|-------------------------------|-------------|-------------|-----------|-------------|
| Name | Slack ^ 1 | Levels | High Fanout | From | To | Total Delay | Logic Delay | Net Delay | Requirement |
| Path 1 | 0.103 | 10 | 15 | cpu1/ID_EX_Databus2_reg[14]/C | cpu1/EX_MEM_ALU_out_reg[31]/D | 9.759 | 6.784 | 2.975 | 10.0 |
| Path 2 | 0.182 | 11 | 15 | cpu1/ID_EX_Databus2_reg[14]/C | cpu1/PC_reg[31]/D | 9.717 | 6.889 | 2.928 | 10.0 |
| Path 3 | 0.209 | 11 | 15 | cpu1/ID_EX_Databus2_reg[14]/C | cpu1/ID_EX_Databus1_reg[28]/D | 9.732 | 6.785 | 2.947 | 10.0 |
| Path 4 | 0.230 | 10 | 15 | cpu1/ID_EX_Databus2_reg[14]/C | cpu1/PC_reg[24]/D | 9.668 | 6.716 | 2.952 | 10.0 |
| Path 5 | 0.244 | 9 | 15 | cpu1/ID_EX_Databus2_reg[14]/C | cpu1/EX_MEM_ALU_out_reg[24]/D | 9.595 | 6.611 | 2.984 | 10.0 |
| Path 6 | 0.262 | 10 | 15 | cpu1/ID_EX_Databus2_reg[14]/C | cpu1/ID_EX_Databus1_reg[24]/D | 9.678 | 6.716 | 2.962 | 10.0 |
| Path 7 | 0.291 | 10 | 15 | cpu1/ID_EX_Databus2_reg[14]/C | cpu1/ID_EX_Databus2_reg[31]/D | 9.607 | 6.784 | 2.823 | 10.0 |
| Path 8 | 0.296 | 10 | 15 | cpu1/ID_EX_Databus2_reg[14]/C | cpu1/ID_EX_Databus2_reg[24]/D | 9.645 | 6.716 | 2.929 | 10.0 |
| Path 9 | 0.332 | 10 | 15 | cpu1/ID_EX_Databus2_reg[14]/C | cpu1/PC_reg[20]/D | 9.566 | 6.597 | 2.969 | 10.0 |
| Path 10 | 0.340 | 10 | 15 | cpu1/ID_EX_Databus2_reg[14]/C | cpu1/ID_EX_Databus1_reg[25]/D | 9.600 | 6.808 | 2.792 | 10.0 |

我们惊喜地发现延时迈入了 10 ns 大关，同时 100 MHz 下的时序性能也得以满足。

是否还有进一步优化的空间呢？我们再来看看延时最大的数据路径：

| Path 1 - top_timing_summary_routed | | | | | | | |
|---------------------------------------|-----------|-----------|--------------------|-----------|---------------------------------------|--|--|
| Data Path | | Incr (ns) | Path ... | Location | Cell Pin | Cell | Netlist Resources |
| FDCE (Prop_fdce_C_Q) | (r) 0.379 | 4.934 | Site: SLICE_X36Y93 | Q | ID_EX_Databus2_reg[14] (FDCE) | cpu1/alu1/ID_EX_Databus2_reg[14]/Q | |
| net (fo=6, routed) | 0.573 | 5.507 | | | | | cpu1/alu1/out0_1_1[14] |
| LUT3 (Prop_lut3_I2_Q) | (r) 0.114 | 5.621 | Site: SLICE_X36Y93 | O | out0_i_18 (LUT3) | cpu1/alu1/out0_i_18/O | |
| net (fo=15, routed) | 0.801 | 6.422 | | | | | cpu1/alu1/ALU_in2[14] |
| DSP48E1 (Prop_DSP..._B[14]_PCOUT[47]) | (r) 3.413 | 9.835 | Site: DSP48_X0Y37 | PCOUT[47] | out0_0 (DSP48E1) | cpu1/alu1/out0_0/PCOUT[47] | |
| net (fo=1, routed) | 0.002 | 9.837 | | | | | cpu1/alu1/out0_0_n_106 |
| DSP48E1 (Prop_DSP48e1_PCIN[47]_P[4]) | (r) 1.271 | 11.108 | Site: DSP48_X0Y38 | P[4] | out0_1 (DSP48E1) | cpu1/alu1/out0_1_P[4] | |
| net (fo=2, routed) | 0.684 | 11.791 | | | | | cpu1/alu1/out0_1_n_101 |
| LUT2 (Prop_lut2_I0_Q) | (r) 0.105 | 11.896 | Site: SLICE_X11Y96 | O | EX_MEM_ALU_out[23]_i_20 (LUT2) | cpu1/alu1/EX_MEM_ALU_out[23]_i_20/O | |
| net (fo=1, routed) | 0.000 | 11.896 | | | | | cpu1/alu1/EX_MEM_ALU_out[23]_i_20_n_0 |
| CARRY4 (Prop_carry4_S[1]_CO[3]) | (r) 0.457 | 12.353 | Site: SLICE_X11Y96 | CO[3] | EX_MEM_ALU_out[reg][23]_i_15 (CARRY4) | cpu1/alu1/EX_MEM_ALU_out[reg][23]_i_15/CO[3] | |
| net (fo=1, routed) | 0.000 | 12.353 | | | | | cpu1/alu1/EX_MEM_ALU_out[reg][23]_i_15_n_0 |
| CARRY4 (Prop_carry4_Cl_CO[3]) | (r) 0.098 | 12.451 | Site: SLICE_X11Y97 | CO[3] | EX_MEM_ALU_out[reg][27]_i_13 (CARRY4) | cpu1/alu1/EX_MEM_ALU_out[reg][27]_i_13/CO[3] | |
| net (fo=1, routed) | 0.000 | 12.451 | | | | | cpu1/alu1/EX_MEM_ALU_out[reg][27]_i_13_n_0 |
| CARRY4 (Prop_carry4_Cl_O[3]) | (r) 0.260 | 12.711 | Site: SLICE_X11Y98 | O[3] | EX_MEM_ALU_out[reg][31]_i_26 (CARRY4) | cpu1/alu1/EX_MEM_ALU_out[reg][31]_i_26/O[3] | |
| net (fo=1, routed) | 0.392 | 13.103 | | | | | cpu1/alu1/EX_MEM_ALU_out[reg][31]_i_26_n_4 |
| LUT5 (Prop_lut5_I0_Q) | (r) 0.257 | 13.360 | Site: SLICE_X13Y98 | O | EX_MEM_ALU_out[31]_i_15 (LUT5) | cpu1/alu1/EX_MEM_ALU_out[31]_i_15/O | |
| net (fo=1, routed) | 0.000 | 13.360 | | | | | cpu1/alu1/EX_MEM_ALU_out[31]_i_15_n_0 |
| MUX7 (Prop_mux7_I0_Q) | (r) 0.178 | 13.538 | Site: SLICE_X13Y98 | O | EX_MEM_ALU_out[reg][31]_i_7 (MUX7) | cpu1/alu1/EX_MEM_ALU_out[reg][31]_i_7/O | |
| net (fo=3, routed) | 0.251 | 13.789 | | | | | cpu1/alu1/EX_MEM_ALU_out[reg][31]_i_7_n_0 |
| LUT6 (Prop_lut6_I5_Q) | (r) 0.252 | 14.041 | Site: SLICE_X13Y97 | O | EX_MEM_ALU_out[31]_i_1 (LUT6) | cpu1/alu1/EX_MEM_ALU_out[31]_i_1/O | |
| net (fo=2, routed) | 0.273 | 14.315 | | | | | cpu1/ALU_out[31] |
| FDCE | | | Site: SLICE_X12Y99 | D | EX_MEM_ALU_out[reg][31] (FDCE) | cpu1/EX_MEM_ALU_out[reg][31]/D | |
| Arrival Time | | 14.315 | | | | | |

更短了！这仍旧是一条 EX 阶段的数据通路，不过经过我们的上一次修改，通路上已经看不到转发了，纯粹地就是从 ALU 输入到输出。

我们注意到路径中经过了 2 个 DSP 模块，这主要是用以支持乘法计算的。我查询资料发现乘法还可以通过 Vivado 提供的 Multiplier IP 核来实现，这样是否能减少延时呢？以下是尝试的修改代码：

```
// ALU.v
wire [32 - 1: 0] mult_out;
mult_gen_0 mult1(
    .A (in1),
    .B (in2),
    .P (mult_out)
);

always @(*) begin
    case (ALUct1)
        4'd2: out <= mult_out;
    endcase
end
```

注意 Multiplier IP 核自定义时需要选择“速度优化”、实现的硬件资源需要选择“Use Mults”、流水线阶段数需要选择为 0，来支持一个周期内的异步读取。

经过修改后的代码综合后时序性能如下：

Version: v1.2.1

Commit Hash: 2728fc6

Design Timing Summary

| Setup | Hold | Pulse Width |
|--------------------------------------|----------------------------------|---|
| Worst Negative Slack (WNS): 0.227 ns | Worst Hold Slack (WHS): 0.080 ns | Worst Pulse Width Slack (WPWS): 3.870 ns |
| Total Negative Slack (TNS): 0.000 ns | Total Hold Slack (THS): 0.000 ns | Total Pulse Width Negative Slack (TPWS): 0.000 ns |
| Number of Failing Endpoints: 0 | Number of Failing Endpoints: 0 | Number of Failing Endpoints: 0 |
| Total Number of Endpoints: 7753 | Total Number of Endpoints: 7753 | Total Number of Endpoints: 2006 |

All user specified timing constraints are met.

| Name | Slack ^ 1 | Levels | High Fanout | From | To | Total Delay | Logic Delay | Net Delay | Requirement |
|---------|-----------|--------|-------------|--------------------------|-------------------------------|-------------|-------------|-----------|-------------|
| Path 1 | 0.227 | 10 | 19 | cpu1/ID_EX_ALUSrc1_reg/C | cpu1/PC_reg[30]/D | 9.763 | 7.762 | 2.001 | 10.0 |
| Path 2 | 0.292 | 10 | 19 | cpu1/ID_EX_ALUSrc1_reg/C | cpu1/PC_reg[31]/D | 9.698 | 7.697 | 2.001 | 10.0 |
| Path 3 | 0.311 | 10 | 19 | cpu1/ID_EX_ALUSrc1_reg/C | cpu1/PC_reg[29]/D | 9.679 | 7.678 | 2.001 | 10.0 |
| Path 4 | 0.326 | 9 | 19 | cpu1/ID_EX_ALUSrc1_reg/C | cpu1/PC_reg[26]/D | 9.665 | 7.664 | 2.001 | 10.0 |
| Path 5 | 0.331 | 9 | 19 | cpu1/ID_EX_ALUSrc1_reg/C | cpu1/PC_reg[28]/D | 9.660 | 7.659 | 2.001 | 10.0 |
| Path 6 | 0.391 | 9 | 19 | cpu1/ID_EX_ALUSrc1_reg/C | cpu1/PC_reg[27]/D | 9.600 | 7.599 | 2.001 | 10.0 |
| Path 7 | 0.394 | 7 | 19 | cpu1/ID_EX_ALUSrc1_reg/C | cpu1/ID_EX_Databus1_reg[20]/D | 9.546 | 6.959 | 2.587 | 10.0 |
| Path 8 | 0.409 | 6 | 19 | cpu1/ID_EX_ALUSrc1_reg/C | cpu1/EX_MEM_ALU_out_reg[18]/D | 9.510 | 6.854 | 2.656 | 10.0 |
| Path 9 | 0.410 | 9 | 19 | cpu1/ID_EX_ALUSrc1_reg/C | cpu1/PC_reg[25]/D | 9.581 | 7.580 | 2.001 | 10.0 |
| Path 10 | 0.437 | 7 | 19 | cpu1/ID_EX_ALUSrc1_reg/C | cpu1/ID_EX_Databus2_reg[18]/D | 9.572 | 6.959 | 2.613 | 10.0 |

可以看到有小幅提升，但并不明显。

我还尝试了将程序内存改写为 BRAM（上面已经介绍过），综合后时序性能如下：

Version: v1.2.2

Commit Hash: eb9f018

Design Timing Summary

| Setup | | Hold | | Pulse Width | | | |
|------------------------------|--|----------|--|------------------------------|--|----------|--|
| Worst Negative Slack (WNS): | | 0.333 ns | | Worst Hold Slack (WHS): | | 0.100 ns | |
| Total Negative Slack (TNS): | | 0.000 ns | | Total Hold Slack (THS): | | 0.000 ns | |
| Number of Failing Endpoints: | | 0 | | Number of Failing Endpoints: | | 0 | |
| Total Number of Endpoints: | | 7697 | | Total Number of Endpoints: | | 7697 | |

All user specified timing constraints are met.

| Intra-Clock Paths - clock - Setup | | | | | | | | | |
|-----------------------------------|-----------|--------|-------------|------------------------------|-------------------|-------------|-------------|-----------|-------------|
| Name | Slack ^ 1 | Levels | High Fanout | From | To | Total Delay | Logic Delay | Net Delay | Requirement |
| Path 1 | 0.333 | 11 | 12 | cpu1/ID_EX_Databus2_reg[3]/C | cpu1/PC_reg[29]/D | 9.658 | 7.599 | 2.059 | 10.0 |
| Path 2 | 0.338 | 11 | 12 | cpu1/ID_EX_Databus2_reg[3]/C | cpu1/PC_reg[31]/D | 9.653 | 7.594 | 2.059 | 10.0 |
| Path 3 | 0.398 | 11 | 12 | cpu1/ID_EX_Databus2_reg[3]/C | cpu1/PC_reg[30]/D | 9.593 | 7.534 | 2.059 | 10.0 |
| Path 4 | 0.417 | 11 | 12 | cpu1/ID_EX_Databus2_reg[3]/C | cpu1/PC_reg[28]/D | 9.574 | 7.515 | 2.059 | 10.0 |
| Path 5 | 0.432 | 10 | 12 | cpu1/ID_EX_Databus2_reg[3]/C | cpu1/PC_reg[25]/D | 9.560 | 7.501 | 2.059 | 10.0 |
| Path 6 | 0.437 | 10 | 12 | cpu1/ID_EX_Databus2_reg[3]/C | cpu1/PC_reg[27]/D | 9.555 | 7.496 | 2.059 | 10.0 |
| Path 7 | 0.497 | 10 | 12 | cpu1/ID_EX_Databus2_reg[3]/C | cpu1/PC_reg[26]/D | 9.495 | 7.436 | 2.059 | 10.0 |
| Path 8 | 0.516 | 10 | 12 | cpu1/ID_EX_Databus2_reg[3]/C | cpu1/PC_reg[24]/D | 9.476 | 7.417 | 2.059 | 10.0 |
| Path 9 | 0.530 | 9 | 12 | cpu1/ID_EX_Databus2_reg[3]/C | cpu1/PC_reg[21]/D | 9.462 | 7.403 | 2.059 | 10.0 |
| Path 10 | 0.535 | 9 | 12 | cpu1/ID_EX_Databus2_reg[3]/C | cpu1/PC_reg[23]/D | 9.457 | 7.398 | 2.059 | 10.0 |

同样也是有小幅提升，但并不明显。

至此，建立时间的最大裕量为 0.333 ns，对应理论最高时钟频率为

$$f_{max} = \frac{1}{\text{period} - \text{slack}} = \frac{1}{10 \text{ ns} - 0.333 \text{ ns}} = 103.34 \text{ MHz}$$

正当我准备结束优化时，我突然想到：既然延时最大的是 EX 阶段的 ALU 计算，而拖慢它的是乘法运算，我是不可以直接不支持乘法，反正排序算法用不上呢？

说干就干，下面是删除乘法后的时序性能：

Version: v2.0

Commit Hash: 17f3269

Design Timing Summary

| Setup | | Hold | | Pulse Width | | | |
|------------------------------|--|-----------|--|------------------------------|--|----------|--|
| Worst Negative Slack (WNS): | | -0.034 ns | | Worst Hold Slack (WHS): | | 0.074 ns | |
| Total Negative Slack (TNS): | | -0.034 ns | | Total Hold Slack (THS): | | 0.000 ns | |
| Number of Failing Endpoints: | | 1 | | Number of Failing Endpoints: | | 0 | |
| Total Number of Endpoints: | | 7746 | | Total Number of Endpoints: | | 7746 | |

Timing constraints are not met.

| Name | Slack ^ 1 | Levels | High Fanout | From | To | Total Delay | Logic Delay | Net Delay | Requirement |
|---------|-----------|--------|-------------|----------------------------------|--------------------------------|-------------|-------------|-----------|-------------|
| Path 1 | -0.034 | 14 | 232 | cpu1/IF_ID_Instruction_reg[21]/C | cpu1/PC_reg[30]/D | 6.595 | 2.691 | 3.904 | 6.7 |
| Path 2 | 0.031 | 14 | 232 | cpu1/IF_ID_Instruction_reg[21]/C | cpu1/PC_reg[31]/D | 6.530 | 2.626 | 3.904 | 6.7 |
| Path 3 | 0.050 | 14 | 232 | cpu1/IF_ID_Instruction_reg[21]/C | cpu1/PC_reg[29]/D | 6.511 | 2.607 | 3.904 | 6.7 |
| Path 4 | 0.063 | 13 | 232 | cpu1/IF_ID_Instruction_reg[21]/C | cpu1/PC_reg[26]/D | 6.497 | 2.593 | 3.904 | 6.7 |
| Path 5 | 0.068 | 13 | 232 | cpu1/IF_ID_Instruction_reg[21]/C | cpu1/PC_reg[28]/D | 6.492 | 2.588 | 3.904 | 6.7 |
| Path 6 | 0.094 | 6 | 105 | cpu1/ID_EX_Database1_reg[10]/C | cpu1/ID_EX_Database1_reg[29]/D | 6.503 | 1.376 | 5.127 | 6.7 |
| Path 7 | 0.128 | 13 | 232 | cpu1/IF_ID_Instruction_reg[21]/C | cpu1/PC_reg[27]/D | 6.432 | 2.528 | 3.904 | 6.7 |
| Path 8 | 0.144 | 6 | 105 | cpu1/ID_EX_Database1_reg[10]/C | cpu1/ID_EX_Database1_reg[28]/D | 6.452 | 1.193 | 5.259 | 6.7 |
| Path 9 | 0.147 | 13 | 232 | cpu1/IF_ID_Instruction_reg[21]/C | cpu1/PC_reg[25]/D | 6.413 | 2.509 | 3.904 | 6.7 |
| Path 10 | 0.159 | 12 | 232 | cpu1/IF_ID_Instruction_reg[21]/C | cpu1/PC_reg[22]/D | 6.399 | 2.495 | 3.904 | 6.7 |

这里我为了使 Vivado 尽可能地根据时钟约束进行布线布局优化，将频率提升到了 150 MHz。可以看到延时大幅降低到了 6.595 ns，并且线延迟首次大于了逻辑延迟，说明仍有优化的空间。

我们同样可以查看延时最大的数据路径：

| Path 1 - top_timing_summary_routed | | | | | | |
|------------------------------------|-----------|-----------|--------------------|----------|----------------------------------|--|
| Data Path | | Incr (ns) | Path ... | Location | Cell Pin | Cell |
| FDCE (Prop_fdce_C_Q) | (r) 0.398 | -1.486 | Site: SLICE_X34Y24 | Q | IF_ID_Instruction_reg[21] (FDCE) | cpu1/IF_ID_Instruction_reg[21]/Q |
| net (fo=232, routed) | 0.927 | -0.558 | | | | cpu1/IF_ID_Instruction_reg_n_0[21] |
| LUT3 (Prop_lut3_2_O) | (r) 0.253 | -0.305 | Site: SLICE_X36Y19 | O | ID_EX_Database1[31]_i_20 (LUT3) | cpu1/ID_EX_Database1[31]_i_20/O |
| net (fo=32, routed) | 0.755 | 0.449 | | | | cpu1/register_file1/ID_EX_Database1[1]_i_3_1 |
| LUT6 (Prop_lut6_4_O) | (r) 0.267 | 0.716 | Site: SLICE_X36Y13 | O | ID_EX_Database1[3]_i_5 (LUT6) | cpu1/register_file1/ID_EX_Database1[3]_i_5/O |
| net (fo=1, routed) | 0.468 | 1.184 | | | | cpu1/register_file1/ID_EX_Database1[3]_i_5_n_0 |
| LUT6 (Prop_lut6_1_O) | (r) 0.105 | 1.289 | Site: SLICE_X38Y13 | O | ID_EX_Database1[3]_i_3 (LUT6) | cpu1/register_file1/ID_EX_Database1[3]_i_3/O |
| net (fo=2, routed) | 0.666 | 1.956 | | | | cpu1/register_file1/RF_data[3] |
| LUT5 (Prop_lut5_4_O) | (r) 0.110 | 2.066 | Site: SLICE_X39Y21 | O | PC[4]_i_25 (LUT5) | cpu1/register_file1/PC[4]_i_25/O |
| net (fo=1, routed) | 0.704 | 2.770 | | | | cpu1/register_file1/PC[4]_i_25_n_0 |
| LUT6 (Prop_lut6_0_O) | (r) 0.268 | 3.038 | Site: SLICE_X40Y23 | O | PC[4]_i_12 (LUT6) | cpu1/register_file1/PC[4]_i_12/O |
| net (fo=1, routed) | 0.375 | 3.413 | | | | cpu1/register_file1/PC[4]_i_12_n_0 |
| LUT6 (Prop_lut6_1_0) | (r) 0.105 | 3.518 | Site: SLICE_X37Y23 | O | PC[4]_i_6 (LUT6) | cpu1/register_file1/PC[4]_i_6/O |
| net (fo=1, routed) | 0.000 | 3.518 | | | | cpu1/register_file1/PC[4]_i_6_n_0 |
| CARRY4 (Prop_carry4_S[2]_CO[3]) | (r) 0.332 | 3.850 | Site: SLICE_X37Y23 | CO[3] | PC_reg[4]_i_1 (CARRY4) | cpu1/register_file1/PC_reg[4]_i_1/CO[3] |
| net (fo=1, routed) | 0.000 | 3.850 | | | | cpu1/register_file1/PC_reg[4]_i_1_n_0 |
| CARRY4 (Prop_carry4_CI_CO[3]) | (r) 0.098 | 3.948 | Site: SLICE_X37Y24 | CO[3] | PC_reg[8]_i_1 (CARRY4) | cpu1/register_file1/PC_reg[8]_i_1/CO[3] |
| net (fo=1, routed) | 0.008 | 3.956 | | | | cpu1/register_file1/PC_reg[8]_i_1_n_0 |
| CARRY4 (Prop_carry4_CI_CO[3]) | (r) 0.098 | 4.054 | Site: SLICE_X37Y25 | CO[3] | PC_reg[12]_i_1 (CARRY4) | cpu1/register_file1/PC_reg[12]_i_1/CO[3] |
| net (fo=1, routed) | 0.000 | 4.054 | | | | cpu1/register_file1/PC_reg[12]_i_1_n_0 |
| CARRY4 (Prop_carry4_CI_CO[3]) | (r) 0.098 | 4.152 | Site: SLICE_X37Y26 | CO[3] | PC_reg[16]_i_1 (CARRY4) | cpu1/register_file1/PC_reg[16]_i_1/CO[3] |
| net (fo=1, routed) | 0.000 | 4.152 | | | | cpu1/register_file1/PC_reg[16]_i_1_n_0 |
| CARRY4 (Prop_carry4_CI_CO[3]) | (r) 0.098 | 4.250 | Site: SLICE_X37Y27 | CO[3] | PC_reg[20]_i_1 (CARRY4) | cpu1/register_file1/PC_reg[20]_i_1/CO[3] |
| net (fo=1, routed) | 0.000 | 4.250 | | | | cpu1/register_file1/PC_reg[20]_i_1_n_0 |
| CARRY4 (Prop_carry4_CI_CO[3]) | (r) 0.098 | 4.348 | Site: SLICE_X37Y28 | CO[3] | PC_reg[24]_i_1 (CARRY4) | cpu1/register_file1/PC_reg[24]_i_1/CO[3] |
| net (fo=1, routed) | 0.000 | 4.348 | | | | cpu1/register_file1/PC_reg[24]_i_1_n_0 |
| CARRY4 (Prop_carry4_CI_CO[3]) | (r) 0.098 | 4.446 | Site: SLICE_X37Y29 | CO[3] | PC_reg[28]_i_1 (CARRY4) | cpu1/register_file1/PC_reg[28]_i_1/CO[3] |
| net (fo=1, routed) | 0.000 | 4.446 | | | | cpu1/register_file1/PC_reg[28]_i_1_n_0 |
| CARRY4 (Prop_carry4_CI_O[1]) | (r) 0.265 | 4.711 | Site: SLICE_X37Y30 | O[1] | PC_reg[31]_i_2 (CARRY4) | cpu1/register_file1/PC_reg[31]_i_2/O[1] |
| net (fo=1, routed) | 0.000 | 4.711 | | | | cpu1/register_file1_n_66 |
| FDCE | | | Site: SLICE_X37Y30 | D | PC_reg[30] (FDCE) | cpu1/PC_reg[30]/D |
| Arrival Time | | 4.711 | | | | |

首先可以观察到这次的路径终于不是 EX 阶段的数据通路了。经过与代码的对照，我们可以确认这是 ID 阶段的数据通路，且用于实现 `jr`、`jalr` 指令更新 PC 值。

在这里，跳转指令与之前的分支条件的判断有些类似，同样是额外延长的路径。因此，我想到可以将跳转指令的判断延后到 EX 阶段，这样跳转指令的数据依赖也从 ID 阶段刚计算出的控制信号和 `RegisterFile` 中刚取出的数据，变为了 ID_EX 阶段的寄存器。这样一来，跳转指令的相关逻辑可以与分支指令、ALU 计算在硬件上并行，有效减少延时。当然了，这样做的坏处是跳转时会多清空一个周期，增加 CPI。不过鉴于跳转指令在令中通常占比不高，这一影响应当可以被延时降低带来的更高频率所抵消。

经过修改后的代码综合后时序性能如下：

Version: v3.0

Commit Hash: cfaa343

Design Timing Summary

| Setup | | Hold | | Pulse Width | | |
|------------------------------|-----------|------------------------------|----------|--|----------|--|
| Worst Negative Slack (WNS): | -0.292 ns | Worst Hold Slack (WHS): | 0.079 ns | Worst Pulse Width Slack (WPWS): | 1.370 ns | |
| Total Negative Slack (TNS): | -5.936 ns | Total Hold Slack (THS): | 0.000 ns | Total Pulse Width Negative Slack (TPWS): | 0.000 ns | |
| Number of Failing Endpoints: | 63 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | |
| Total Number of Endpoints: | 7758 | Total Number of Endpoints: | 7758 | Total Number of Endpoints: | 2020 | |

Timing constraints are not met.

| Intra-Clock Paths - clk_out1_clk_wiz_0 - Setup | | | | | | | | | |
|--|--------|--------|-------------|---------------------------------|-------------------------------|-------------|-------------|-----------|-------------|
| Name | Slack | Levels | High Fanout | From | To | Total Delay | Logic Delay | Net Delay | Requirement |
| Path 1 | -0.292 | 10 | 115 | cpu1/ID_EX_Databus1_reg[4]/C | cpu1/ID_EX_Databus1_reg[25]/D | 5.257 | 2.037 | 3.220 | 5.0 |
| Path 2 | -0.263 | 6 | 116 | cpu1/ID_EX_Instruction_reg[9]/C | cpu1/ID_EX_Databus2_reg[15]/D | 5.186 | 1.009 | 4.177 | 5.0 |
| Path 3 | -0.256 | 9 | 141 | cpu1/ID_EX_ALUSrc2_reg/C | cpu1/ID_EX_Databus1_reg[0]/D | 5.219 | 1.537 | 3.682 | 5.0 |
| Path 4 | -0.241 | 6 | 116 | cpu1/ID_EX_Instruction_reg[9]/C | cpu1/ID_EX_Databus1_reg[15]/D | 5.222 | 1.009 | 4.213 | 5.0 |
| Path 5 | -0.226 | 6 | 109 | cpu1/ID_EX_Instruction_reg[8]/C | cpu1/ID_EX_Databus2_reg[20]/D | 5.145 | 1.009 | 4.136 | 5.0 |
| Path 6 | -0.210 | 9 | 141 | cpu1/ID_EX_ALUSrc2_reg/C | cpu1/ID_EX_Databus2_reg[0]/D | 5.146 | 1.537 | 3.609 | 5.0 |
| Path 7 | -0.197 | 8 | 175 | cpu1/ID_EX_Databus2_reg[2]/C | cpu1/ID_EX_Databus1_reg[23]/D | 5.123 | 1.788 | 3.335 | 5.0 |
| Path 8 | -0.192 | 6 | 109 | cpu1/ID_EX_Instruction_reg[8]/C | cpu1/ID_EX_Databus1_reg[20]/D | 5.113 | 1.009 | 4.104 | 5.0 |
| Path 9 | -0.182 | 6 | 158 | cpu1/EX_MEM_AL_g[7]_rep_0/C | cpu1/ID_EX_Databus2_reg[2]/D | 5.100 | 1.009 | 4.091 | 5.0 |
| Path 10 | -0.178 | 8 | 115 | cpu1/ID_EX_Databus1_reg[4]/C | cpu1/ID_EX_Databus1_reg[17]/D | 5.168 | 1.841 | 3.327 | 5.0 |

这里我为了使 Vivado 尽可能地根据时钟约束进行布线布局优化，将频率提升到了 200 MHz。此时，建立时间的最大裕量为 -0.292 ns，对应理论最高时钟频率为

$$f_{max} = \frac{1}{\text{period} - \text{slack}} = \frac{1}{5 \text{ ns} + 0.292 \text{ ns}} = 188.96 \text{ MHz}$$

在测试中，我发现有时即便违反了时间约束（如上面的例子），程序烧写到开发板上后仍能正常工作。我认为这一方面可能是因为 Vivado 为了确保硬件正常工作、将延时估计得略大于实际值，另一方面是因为我编写的 MIPS 程序并没有用到延时最大的数据路径，这些路径在实际程序的使用中触及概率很小。

因此，我就想测测看，如果不管综合得出的理论延时和理论最高频率，单以开发板上能够正常运行为标准，频率最高是多少？

经过多次尝试，我最终将最高频率确定到 270 ~ 275 MHz。其中 270 MHz 频率下程序能够完全正常地运行，而 275 MHz 频率下开发板可以通过 UART 串口正常地接收和发送消息，但已经不能在数码管上显示了（乱码）。

硬件调试

除了“算法指令”部分提到的汇编代码出错导致的调试情况（其实应该在仿真中解决），硬件调试主要就是在解决与外设通信的问题，而其中最耗费时间当属 UART 通信的调试。

由于硬件调试没有合适的断点、输出或变量窗口，我一开始难以定位到出问题的代码。后来经过搜索网络资料，我尝试使用 Vivado 提供的 ILA IP 核进行排错，操作流程如下：

1. 在要查看的寄存器前标注 `(* mark_debug = "true" *)`
2. 在 Vivado 的 Synthesis 完成后，打开 Design 并选择 Set Up Debug
3. 根据向导指引自定义 ILA IP 核
4. 按原有流程运行 Implementation 然后生成比特流文件
5. 在烧写开发板时，窗口中的 debug core 选择对应的 `*.ltx` 文件
6. 在开发板运行程序的过程中，我们可以注意到右上侧显示了与仿真测试类似的波形窗口
7. 在调试窗口设置断点触发条件，点击监听等待条件被触发
8. 触发后可以在波形窗口中查看断点前后的寄存器波形，方便调试排错

以我调试 UART 通信为例。在串口调试助手发送消息后、开发板没有任何反应的情况下，我先是将 `UART` 模块内的寄存器变量全部加上了监听，结果发现 `UART` 模块实际可以正常接收到串口调试助手发送的消息。然后，我转而怀疑 `CPU` 模块没有正常将访问到的 `UART` 消息存到寄存器中，并对访存相关的控制信号都加上了监听，结果一切正常，然而读到的消息却是全 0。两边排除下来唯一的嫌疑落到了 `Device` 模块上，我将 `UART_RXD_read` 等寄存器加上监听后终于解开了错误的面纱：一开始 `UART_RXD_read` 被设置为寄存器，是在过程块中被赋值的，也就意味着它只会在访存的下一个周期变为高电平、在访存的那个周期仍为低电平；而读取数据的总线是根据 `UART_RXD_read` 控制信号来判断读取到的值的，因此自然读到的是全 0。相同的事情发生在了 `UART_TXD_write` 及控制信号的判断上，导致系统始终没能正常发送。

这样的时序错误难以由人工检查代码或硬件调试工具排查出来。我认为在这种情况下 Vivado 提供的 ILA IP 核发挥了重要作用。当然，这个案例也再次体现了线性变量和寄存器变量的不同、assign 异步赋值和过程块同步赋值的差异。

修复这个问题后，系统已经可以正常地通过 UART 串口进行通信了。但是，我在多次尝试后发现仍有一些时候系统无法接收到正确的消息。经过 ILA IP 核的调试，我发现系统内的各个控制信号均没有问题。因此，我推测问题有可能是 rx 在真实物理环境中收到了干扰，使得 UART 模块检测到了低电平并错误地开始了接收（由于波特率远低于系统时钟，调试核的位深不足以记录到 rx 的实际波形，所以只能是推测）。基于这个推测，我修改了 UART 模块代码，使其在开始时至少要检测到 0.5 位长度的低电平才能开始接收，增加接收的可靠性。幸运的是，在修改之后，上述偶发的接受错误的确没有再发生，验证了我的推测。

如果说上面的时序错误还可以被仿真测试检查出来，那么这种偶发性的、与实际物理环境有关的错误几乎只能通过硬件调试来发现。也正是因为在设计的每个步骤都上板验证功能，所以在后续的步骤里才没有出现之前设计的模块的问题、或耦合形成的问题。这一点对我以后进行硬件系统相关的工作很有启发。

总结

将功能相同的单周期处理器与流水线处理器进行比较。

根据静态时序分析的结果，单周期处理器的最高频率为 51.39 MHz，流水线处理器的最高频率 103.34 MHz（乘法操作删除后不计）。而“算法指令”部分计算出排序程序在流水线处理器上运行的 CPI 为 1.723。

以此为基础进行推算，单周期处理器每秒可处理 51.39 M 条指令，而流水线处理器每秒可处理 $103.34 / 1.723 = 59.98$ M 条指令，提升了 16.7%，似乎没有预期地那么多。

提升幅度不大的原因有二：一是因为乘法需要的计算延时太大、使 EX 阶段耗时显著大于其他阶段，流水线的优势并不明显；二是因为我编写的 MIPS 程序中分支指令较多、load-use 冒险也很多，导致 CPI 偏高。

面向未来，针对乘法耗时的问题，可以突破经典 5 级流水线结构，将乘法运算拉长至多个周期，阶段之间任务分配更均匀；针对 CPI 偏高的问题，则可以优化 MIPS 汇编代码（编译器的工作之一），或者增加分支预测来减少分支操作需要清空指令的概率。

关键代码及文件清单

- `single_cycle/`：单周期处理器的相关代码
- `pipeline/`：流水线处理器的相关代码
- `program/`：MIPS 汇编代码
- `img/`：实验报告配图
- `README.md`：支持的 MIPS 指令列表
- `uart_data.txt`：可供串口调试助手直接发送和比对的输入输出数据

在 `single_cycle/` 和 `pipeline/` 内，文件结构如下：

- `cpu`：处理器相关代码
 - `testbench`：仿真测试代码
- `device`：外设相关代码
- `img`：仿真和综合结果截图
- `constraint.xdc`：硬件约束文件
- `top.v`：顶层模块
- `top.bit`：比特流文件，可直接烧写到开发板
- `update_program.tcl`：更新比特流文件的 TCL 脚本，详见前文介绍