流水线大作业说明文档

无 17 马林 2021010540

一. 实验目的：

1.进一步熟悉流水线的课程讲授知识，并在此基础上，自己设计完成 MIPS 中其他指令的实现，在此基础上，加深对流水线的理解；

2.将流水线的理论知识应用到实际的工程设计中来，在自己动手实现流水线的过程中体会实际工程的设计思路与设计考量；

3.体会通用处理器的设计思路和方式，理解并掌握 CPU 与外设的连接与通信过程。

二. 设计方案：

本次设计的 CPU 采用 5 级流水线，分别为 IF,ID,EX,MEM,WB。实现的指令除了所要求的指令外，还有 ori，对于数据超出立即数范围的指令，采用 lui 和 ori 代替实现，在后面 CPI 的计算过程中，该指令对应的指令数记作 2。另外，本次并未实现 UART 串口的相关功能。

对于电路中存在的数据冒险，采用 load-use 电路和 forwarding 电路解决，在设计框图中由 A 和 B 体现（详见下面的原理框图），前者还会 stall 一个周期，并清空 ID/EX 阶段的所有级间寄存器（即为寄存器赋 0）。

对于电路中存在的控制冒险，多采用提前判断并清空部分级间寄存器的方式处理。对于分支指令，在 EX 阶段判断，判断需要跳转时，清空 IF/ID、ID/EX 和 EX/MEM 间的所有级间寄存器。对于 J 类指令和 jr、jalr 指令，在 ID 阶段判断，判断需要跳转时，对于前者，清空 IF/ID 阶段的级间寄存器；对于后者，清空 IF/ID 和 ID/EX 阶段的级间寄存器。

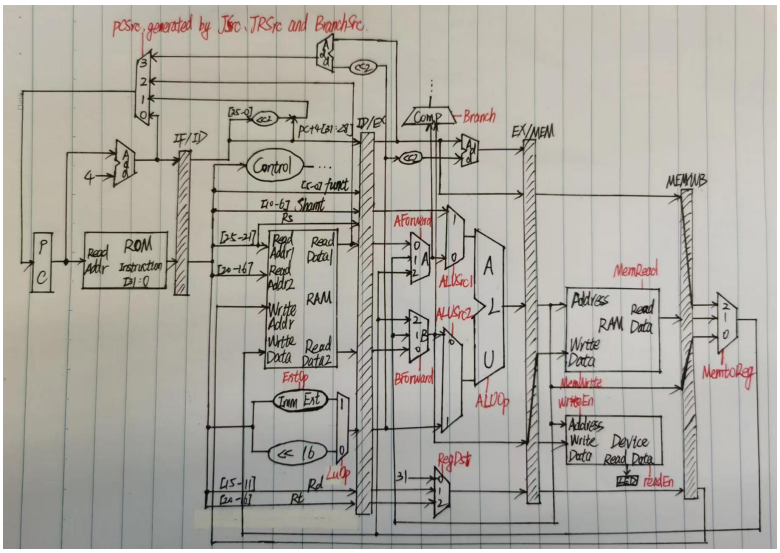
对于 RAM，即数据存储器，以“用多少开多大”的指导，为其开 128 字的空间，每个字 32bits，与此对应的有效地址位数是 7bits，另外，由于要通过软件编写点亮数码管，故对 RAM 的从倒数第二个内存空间开始的连续 16 个内存空间分别初始化为翻译好的初值，初值中不包括 AN 的信息，AN 的控制通过汇编指令实现。对于外设（Device），将从 0x4000_0010 开始的 12bits 内存划分为外设的专有地址，比特与数码管对应位置的对应完全依照指导书，不再赘述。

对于 ROM，即指令存储器，为其开 32768 字的空间，每个字 32bits。在初始化时，先将每个内存空间都初始化为 0，然后利用$readmemh 读取.mem 文件中的指令，本次调试所用的.mem 文件（即 Dijkstra.mem）由 Mars 生成。

对于寄存器，在原定要求之外，实现了先写后读，并且不会延长关键路径。

至于数码管的显示，以 U4 为 reset 按键，R1 为外设的 readEn 控制推钮，当 readEn 为高电平，即推上 R1 时，数码管显示当前计算出的单源路径之和。时钟源即所用板子的 100MHz 时钟源。

本次设计的原理框图如下所示：



注：在本图中，没有体现出部分设计细节，如 flush 等，且 MUX 中的选择序号仅代表数据通路，并不代表其与实际控制信号的对应关系，具体实现以代码为准。

三．关键代码及文件清单：

本次设计中的模块大体沿用“数字逻辑与处理器”课程中关于单周期处理器大作业 CPU 的实现模块，仅有部分改动，如 ALU 中新增了 ori 指令的判别和计算等，在此不作为关键代码列出。

对于外设，写使能存在并且写地址为 0x4000_0010 时，将数据写入外设中，在读使能存在时输出给数码管：

```
reg [11: 0] dataToShow;

assign Device_Read_Data = readEn ? dataToShow: 0;
always @(posedge clk or posedge reset) begin
    if(reset) begin
        dataToShow <= 0;
    end
    else begin
        if(writeEn && Address == 32'h40000010) begin
            dataToShow <= writeData[11: 0];
        end
    end
end
```

对于 CPU，增添了级间寄存器（以“各级流水线名称_下一级流水线名称”命名），数据转发的控制信号按照所有可能存在的导致转发的情况设计：

```
assign AForward = ((ID_EX_Rs == EX_MEM_WriteAddress) && (EX_MEM_RegWrite && EX_MEM_WriteAddress != 0))? 2'b10:
    ((ID_EX_Rs == MEM_WB_WriteAddress) && MEM_WB_RegWrite && MEM_WB_WriteAddress != 0 &&
    (EX_MEM_WriteAddress != ID_EX_Rs || !EX_MEM_RegWrite))? 2'b01: 2'b00;
assign BForward = ((ID_EX_Rt == EX_MEM_WriteAddress) && (EX_MEM_RegWrite && EX_MEM_WriteAddress != 0))? 2'b10:
    ((ID_EX_Rt == MEM_WB_WriteAddress) && (MEM_WB_RegWrite && MEM_WB_WriteAddress != 0) &&
    (EX_MEM_WriteAddress != ID_EX_Rt || !EX_MEM_RegWrite))? 2'b01: 2'b00;

assign AOut = (AForward == 2'b00)? ID_EX_RsData1: (AForward == 2'b01)?
    RFWriteData: EX_MEM_ALUOut;
assign BOut = (BForward == 2'b00)? ID_EX_RtData2: (BForward == 2'b01)?
    RFWriteData: EX_MEM_ALUOut;
assign ALU_in1 = ID_EX_ALUSrc1? ID_EX_Shamt: AOut;
assign ALU_in2 = ID_EX_ALUSrc2? ID_EX_Imm: BOut;
```

对于在 EX 阶段判断的分支指令：

```
assign BranchSrc = ((ID_EX_Branch == 3'b001 && AOut == BOut) ||
    (ID_EX_Branch == 3'b010 && AOut != BOut) ||
    (ID_EX_Branch == 3'b011 && (AOut[31] || AOut == 32'h00000000)) ||
    (ID_EX_Branch == 3'b100 && !AOut[31] && AOut != 32'h00000000) ||
    (ID_EX_Branch == 3'b101 && AOut[31]))? 1: 0;
```

其中，从上到下的条件分别对应于 beq，bne，blez，bgtz 和 bltz，考虑到参与判断的数字中可能有有符号数的存在，故按照有符号数的方式判别其与 0 的大小关系。显然，当 BranchSrc 为 1 时，判断为分支指令并进行跳转。

对于 PC_next，根据 JSrc，JRSrc 和 BranchSrc 判断其取值：

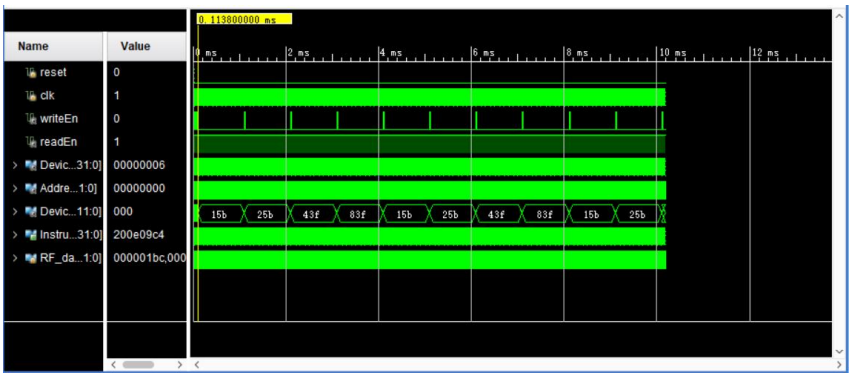
```
assign PC_next = (JSrc && !BranchSrc)? {IF_ID_PC_plus_4[31: 28], IF_ID_Instruction[25: 0], 2'b00}:
    (JRSrc && !BranchSrc)? Rs: BranchSrc? ID_EX_PC_plus_4 + {ID_EX_Imm[29: 0], 2'b00}: PC_plus_4;
```

因为有可能存在分支指令之后紧接着 J 型指令的存在，故分支指令判断的优先级会更高。

在提交的 zip 包中，文件清单由 README.txt 文件给出。

四．仿真结果及分析：

待算法运行完毕后，输出数据 Device_Read_Data 以周期性显示为 0x15b，0x25b，0x43f，0x83f，即周期性点亮数码管为 0x0022，符合设计预期。



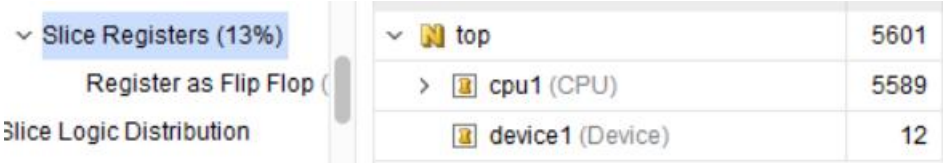
五．综合情况：

资源占用：

查找表占用情况如下：

可见，共使用 3147 个查找表。

寄存器占用情况如下：



可见，共使用 5601 个寄存器。

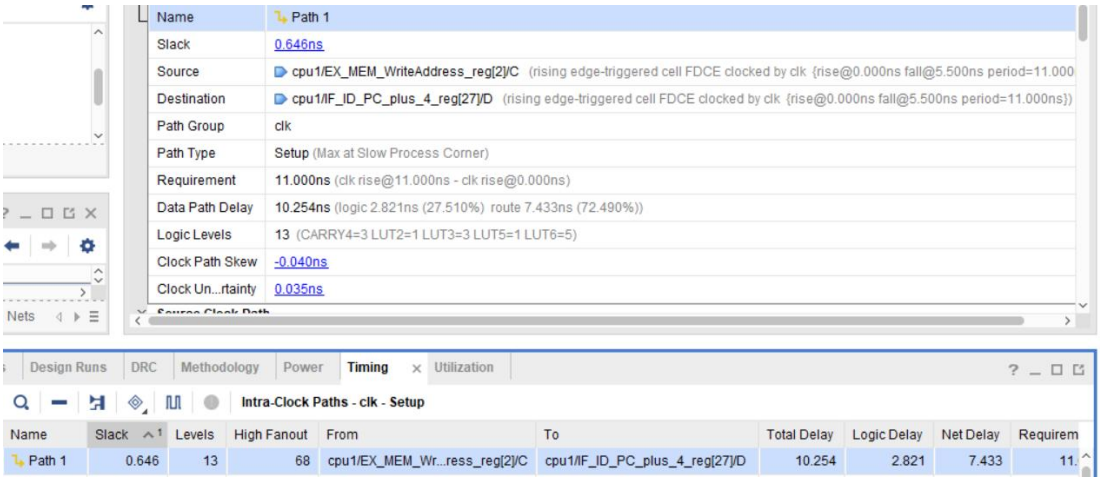
时序性能：

时序性能如下：

据此，可以计算出流水线的主频为：

$$f = \frac{1}{11 - 0.646} \times 10^9 Hz = 96.58 MHz$$

与之对应的关键路径为：

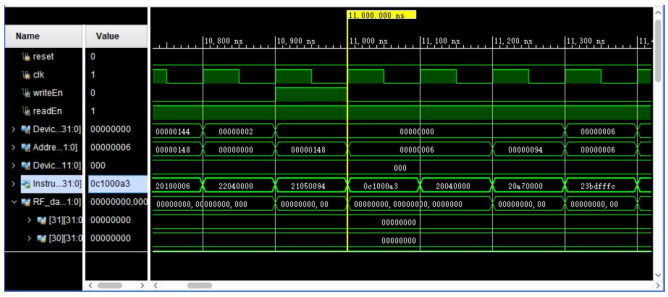


可见，关键路径是分支指令在 EX 阶段并存在 Forwarding 时的通路，该路径有 13 级组合逻辑，扇出为 68。

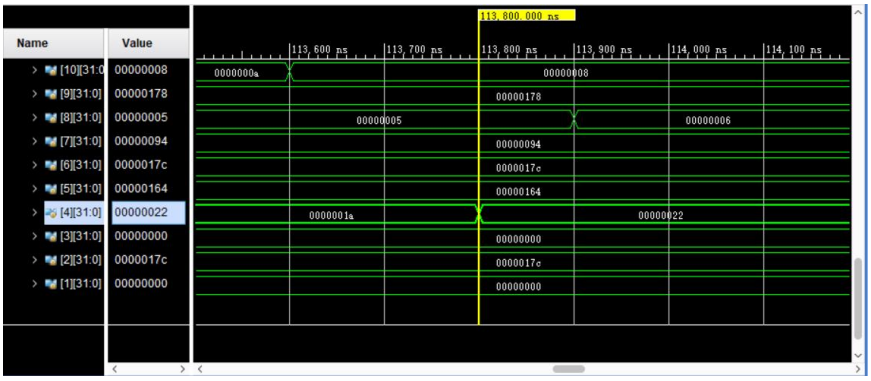
不考虑数据的读入和最终数码管的显示，根据 vivado 对 Dijkstra.mem 汇编指令的仿真，总共有

$$\frac{113800-11000}{100}=1028$$

条指令：

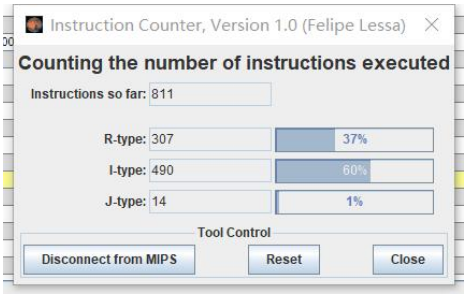


11000ns 对应的 0c1000a3 为跳转到 dijkstra 的 jal 指令。



113800ns 对应的$a0 = 22 为计算完毕的路径之和。

再根据 Mars 的 Instruction Counter，可以算出这段指令数为 811。



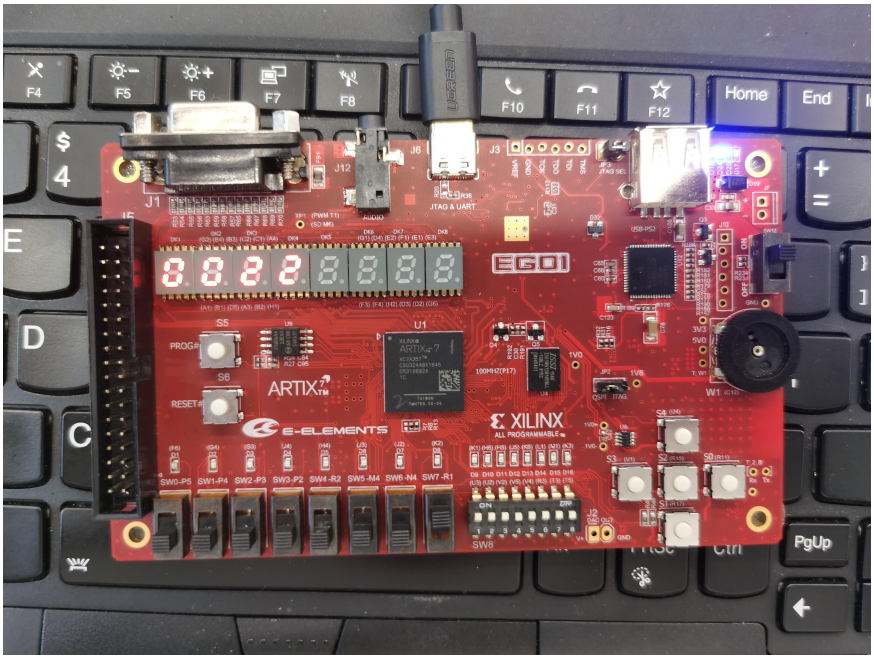
由此，可以算出 CPI 为：

$$CPI = \frac{1028}{811} = 1.2676$$

可见，与理想流水线 CPI 接近于 1 还有不小的差距，原因可能是跳转和 load-use 导致的 flush 和 stall 使流水线中产生了非理想因素。

六．实际硬件验证情况和分析：

按 reset 键（即 U4）之后，待程序运行一定时间，推 readEn 为高电平（即推上 R1）之后，在数码管上显示出稳定的 0x0022，即路径之和为 34，与预期相同（测试数据同单周期处理器）。



七．思想体会：

1. 设计图纸尤为关键，在没有设计图纸时，仅凭感觉写出的流水线很多连输入输出都给错了，有了设计图纸，按照图纸上完成代码则轻松很多。

2. 总体的设计比先实现部分功能，再去完善要更好。起初设计的时候，只是考虑了

Forwarding，而没有考虑控制冒险，准备在之后完善流水线解决控制冒险。但是实现了基本功能转向解决控制冒险的时候，部分流水线原有的结构被打破了，导致有了较大规模的改动，在完成改动之后，事后觉得一开始就考虑到所有要实现的功能再去设计框架并逐渐完善是个更好的选择。

3.善用 Git!!! 在本次流水线的实现过程中，本人有两次不小心误删了关键文件，并且无法找回，导致多做了很多重复的工作，浪费了大量时间，甚至延后了验收。

4.本次的流水线大作业于我而言是对理论课的很好的巩固复习，对于流水线的运转有了更深的体会。觉得一切都自然了很多，原来没有想清楚的问题现在也有了理解。