

**计算机组成·本科实验报告**

|  |  |
| --- | --- |
| 姓 名： | 李欣奇 |
| 学 院： | 信息与电子工程学院 |
| 专 业： | 微电子科学与工程 |
| 学 号： | 3210105139 |
| 指导教师： | 黄科杰 |

2024 年 4 月 21 日

**实验报告**

日期：2024.4.21

地点：

课程名称： 计算机组成 指导老师： 黄科杰

实验名称： 单周期RISC-V仿真

1. **实验目标**
2. **实验环境和工具**
3. **实验内容和过程（总结自己所做的三个阶段工作）**
4. **实验总结（说说踩的坑，总结收获，分析下自己花了多少时间，都用来做什么事情）**
5. **提出改进实验的意见**

一、实验目标

用verilog实现RV32I **单周期CPU**。

二、实验环境和工具

Vivado 2023.2

Ubuntu -64 22.04.3 LTS

三、实验内容和过程

问题1：

问题描述：

lui t0, 0x1 # Load upper immediate (t0 = 0x1000)

addi s1, t0, 4 # s1 = t0 + 4 = 0x1004

无法加载出存储在t0中的值（t0=0x1000），如图1所示。

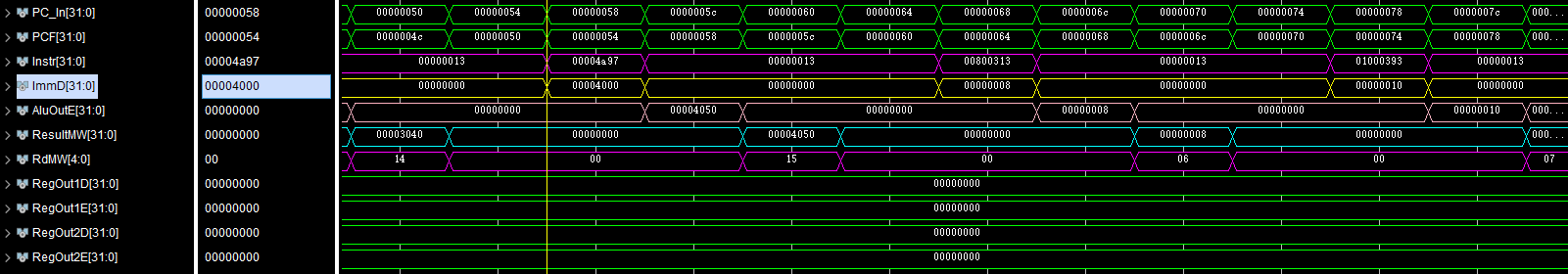


图 1

问题分析：

如图2所示，在RegisterFile信号中，A3地址信号，WD3数据信号和WE3写使能信号都正确，但注意到reset此时还是高电平，因此需要修改系统的复位逻辑。

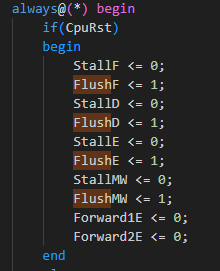


图 3

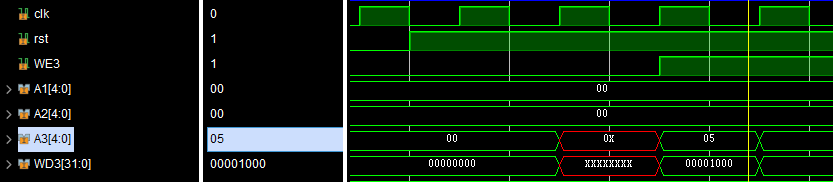


图 2

修改后的代码如右图所示。在Hazard文件的注释中有这一段话：

    //可以最后实现。前期测试CPU正确性时，可以在每两条指令间插入四条空指令，然后直接把本模块输出定为，不forward，不stall，不flush

但这句话是错的，不stall不flush带来的结果就是reset一旦激活程序就立刻开始执行，这导致了RegFile以及其他有reset信号输入的模块本身还在reset的状态就要执行命令。这会导致reset激活状态期间运行错误。

修改后命令运行正常。如图4所示。

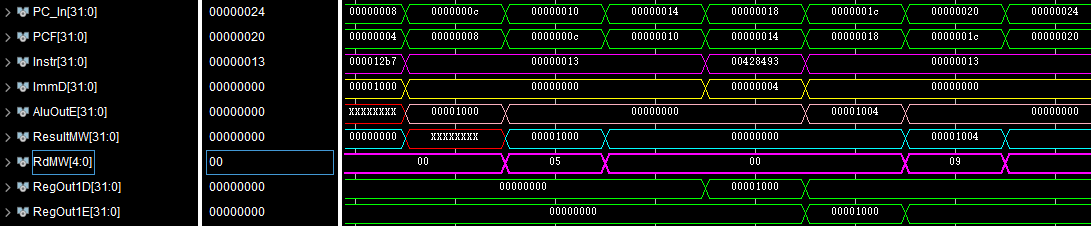


图 4

问题2：

问题描述：

sw t0, 0(t1) # Save word from t0 to [t1]

所有的S-type无法正常save，但是I-type中的load可以正常load。

问题分析：

如图5所示。MemWriteM信号传输不正确。

本该在MW阶段执行的Mem写入在Ex阶段就执行了。



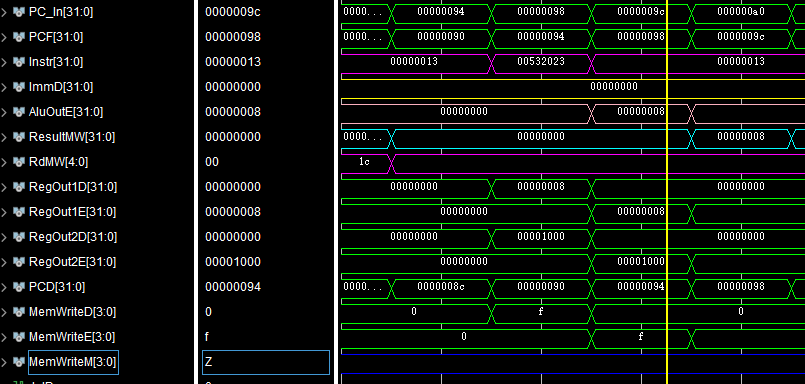


图 5

四、实验总结

（1）代码编写阶段

代码编写阶段总体花费时间不长，在8h以内。基本上就是根据机器码、硬件实现图将其代码化，有一些课堂上没有讲的细节需要花一点时间。例如代码Core中Branch实现的方式和课堂上讲授的连线有所区别，但实际测试发现二者都是正确的。再例如在DataRam的实现的具体细节上，由于其是32位划分的，在处理地址A[1:0]不等于0以及SB，SH的时候需要重新考虑WE，WD的值（由于A直接舍弃了末两位）。总之代码写的时候越小心，debug遇到的问题就越少，写的不仔细，debug就会付出很大的代价。我就是后者。

  wire [3:0] wea;

    wire [31:0] dina;

    assign wea = WE<<A[1:0];

    assign dina = WD<<(A[1:0]\*8);

    DataRam DataRamInst (

        .clk    (clk),                      //请补全

        .wea    (wea),                      //请补全

        .addra  (A[31:2]),                      //请补全

        .dina   (dina),                      //请补全

        .douta  ( RD\_raw         ),

        .web    ( WE2            ),

        .addrb  ( A2[31:2]       ),

        .dinb   ( WD2            ),

        .doutb  ( RD2            )

    );

上图是DataSeg的部分代码。

（2）代码自行调试阶段

对于Part2部分的调试花费了比较多的时间，一部分是编写汇编时编写太多跳转代码比较容易搞混的问题，另一部分则是Debug不知道程序跳到哪里去的问题。总之这部分花费的时间是在意料之中的，带来的收获则是B-type指令和JAL，JALR几乎在后续的debug中没有出问题。

此外，还需要提到的是，在debug期间，由于我改变了Core，加入了新的ImmOperandUnit，在写这部分代码时比较糟糕，导致调试的时候发现许多情况下Imm加起来都不是32位而是31位！好在这样的错误比较容易发现。

此阶段用时6h以内。

（3）官方代码调试阶段

此阶段用时无法统计。

在调试test2时遇到了超级多的bug，而其大部分都是关于SB，LB的。此外还有AUIPC、LUI、BEQ、LH、SLT等命令的。以下是几个耗时比较久的debug测试点：

1test128, 2test274, 2test310, 2test372, 2test373, 2test375, 2test491

其中标红的三个测试点都是SB出了问题。这个问题的难点在于，由于其前后指令就是SB，LB，如果不深入底层的DataRamInst比较难以判断究竟是save的问题还是load的问题。常常在判断bug的位置就花费了很多的时间，而改这个bug又由于有一个\*8的操作太过扎眼想改掉但又难以想到更简单的方案（<<3由于位数限制不可行，除非另外再声明）。

五、实验改进

1. 课堂上讲的端口设计和连接方式可以和代码尽量统一（Branch）

2. 代码具体实现上的细节课上可以提一下，例如比较困难的DataExt和DataRamIn的预处理

3. 单周期完全可以把ImmOperandUnit加上去，方便后面Pipeline的实验。