**单周期处理器实验报告**

2018302120209 罗宇轩

一、实验结果截图

文件mipstestloop\_sim.asm运行结果如下，其中Mars的寄存器结果如图1-1所示，在Modelsim中的寄存器结果如图1-2所示。

图片包含 游戏机

描述已自动生成

图1-1

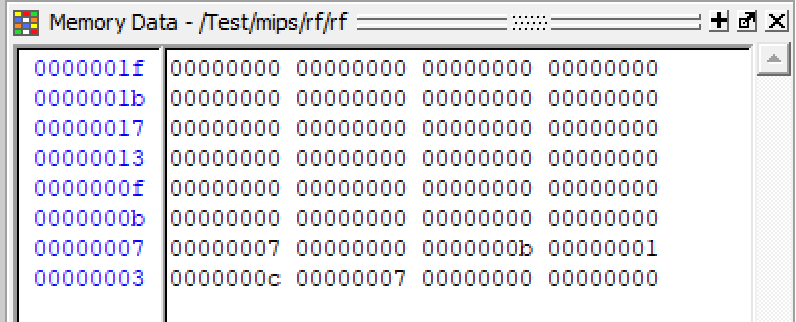


图1-2

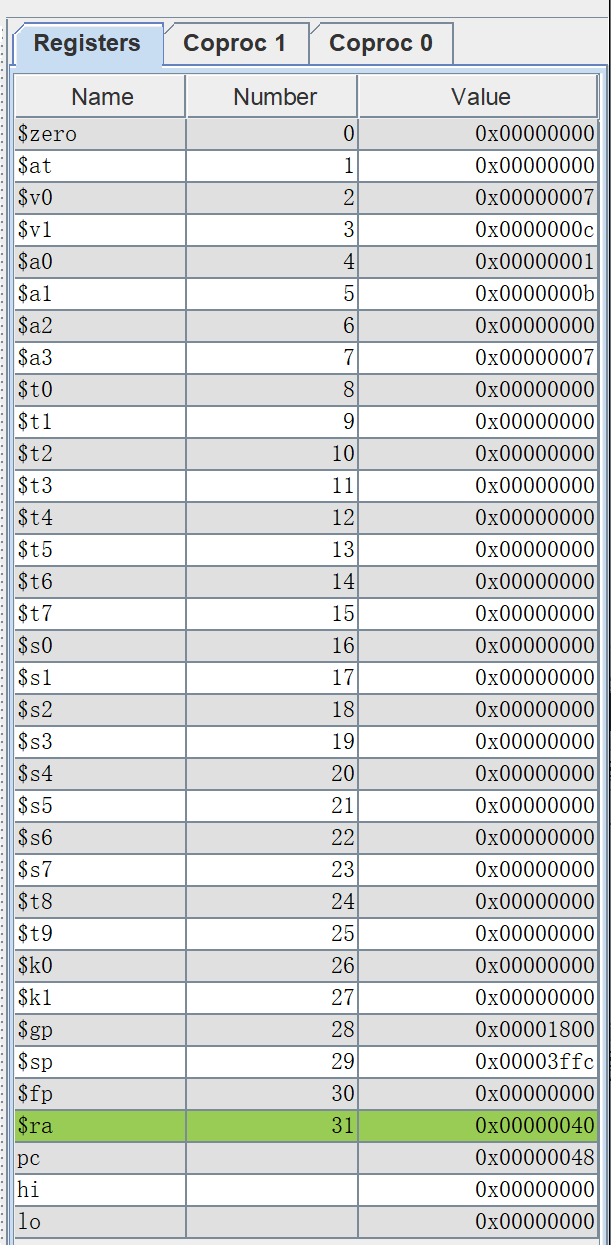
文件mipstestloopjal\_sim．asm运行结果如下，其中Mars的寄存器结果如图1-3所示，在Modelsim中的寄存器结果如图1-4所示。

图1-3

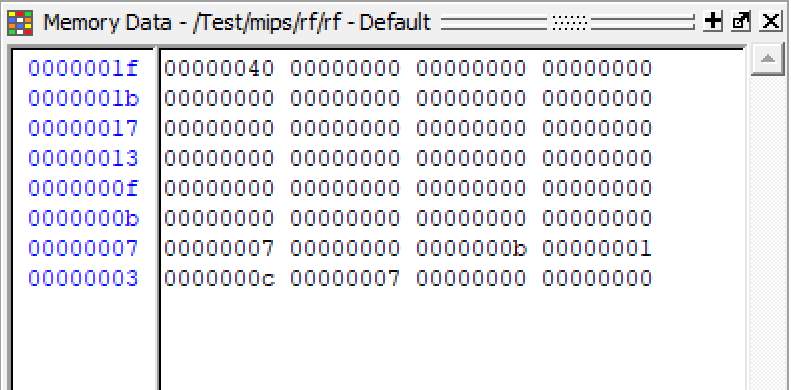
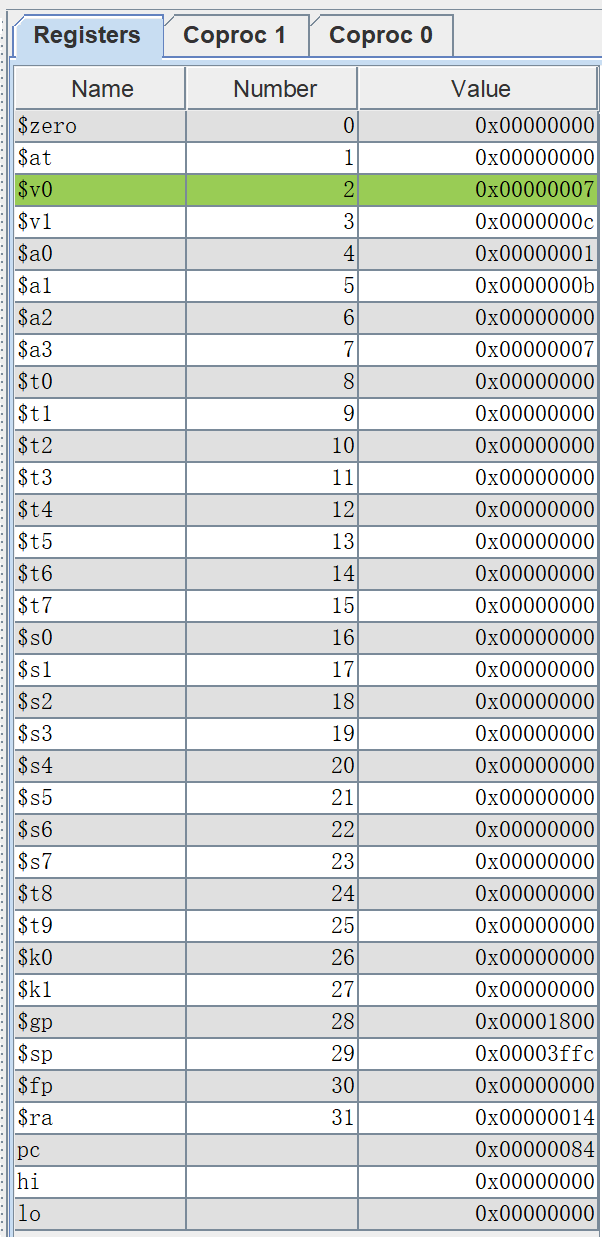


图1-4

文件mipstest\_extloop.asm运行结果如下，其中Mars的寄存器结果如图1-5所示，在Modelsim中的寄存器结果如图1-6所示。

图1-5

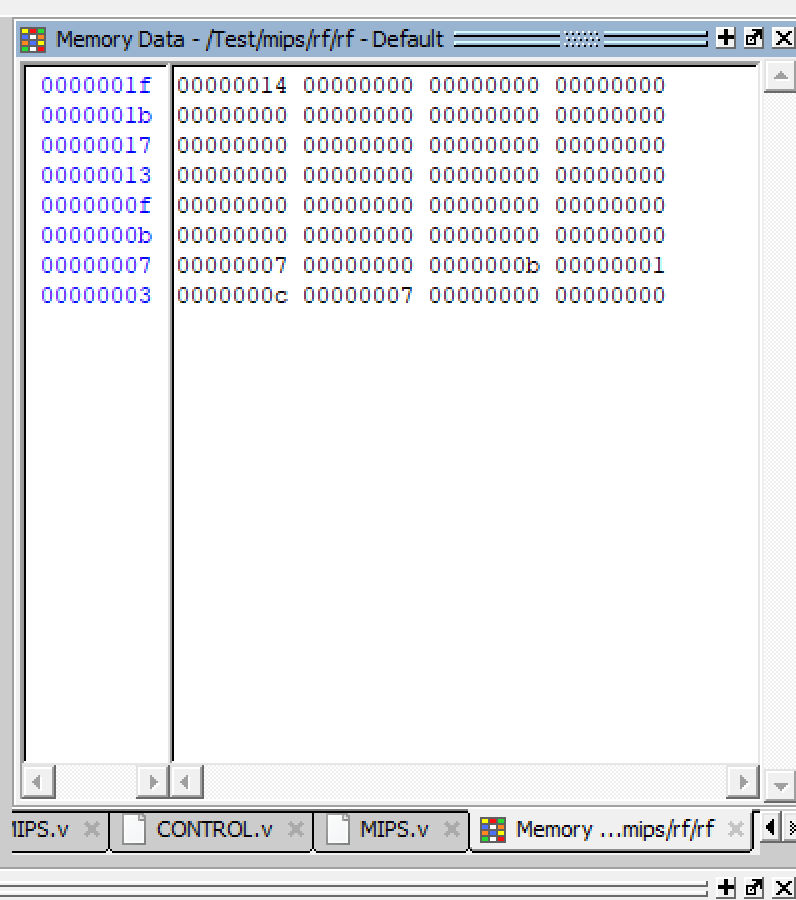


图1-6

二、实验结果分析

该单周期CPU的工程目录如图２－１所示，其中Ｔｅｓｔ＿ＭＩＰＳ．ｖ是ｔｅｓｔｂｅｎｃｈ，ＭＩＰＳ．ｖ是顶层设计，其他ｖｅｒｉｌｏｇ文件都是功能部件。执行完三个测试文件代码后，经比对Ｍａｒｓ中的寄存器结果和Ｍｏｄｅｌｓｉｍ中寄存器的结果完全一致，证明能实现课程目标中要求实现的指令。



图２－１

三、遇到的问题

1. modelsim中的系统函数$display无法使用。在网上搜寻解决方案时发现这种情况存在的比较少，初步诊断原因是系统缺少相关的依赖库，导致系统函数$display无法使用。这个问题涉及modelsim和操作系统交互，不是很好解决。所以只能放弃使用这个函数，删去现有代码中的$display语句，转而将输出结果直接在波形图中查看。
2. 难以区分addu,addi和add指令的区别。在一开始实现add指令的时候感觉非常容易，而在实现addu和addi指令的时候则感觉比较迷茫，不知道控制信号和add指令有何不同。Addu指令和add指令比较类似，控制信号基本相同，所需要做的只是在ALU中添加额外的运算操作，比如对无符号数加法的支持。addi指令则可以说和add区别较大，有着不同的指令格式，实现过程中的数据通路也和add指令不同。
3. Verilog中有符号数和无符号数的操作问题。MIPS体系中数据以补码表示法存储，而实现诸如addu这样的指令的时候又需要实现无符号数的运算，所以就需要对整个MIPS中的数据是signed类型还是unsigned类型比较清楚。同时因为涉及有符号数和无符号数的运算规则不同这一问题，所以在设计ALU时需要提供很多额外的支持。好在仔细学习verilog后发现verilog语言中能通过语言提供的系统函数来实现有符号数和无符号数的转换，从而避免了从底层来解决这一问题。
4. 地址移位的问题。程序计数器PC中的地址是以字节寻址的。但是在设计程序存储器IM的时候选用的是32位的存储器，即以字地址寻址的。故将PC中的指令地址输入到IM的时候需要右移两位（在实际设计的过程中是从第2位开始截取地址实现的）。在数据存储器MEM中也有类似的问题。
5. 关于单周期cpu中的时序问题。根据教材上的设计，单周期cpu中所有的状态单元连接同一时钟信号clk。在进行simulation之前我感觉很困惑：如果单周期cpu是一个同步系统，那么在一个时钟周期内应该无法完成指令中写入寄存器和写入数据存储器的指令内容。后来在进行simulation之后发现之前的困惑是正确的，当RF和MEM是clk上升沿触发的时候的确无法在一个时钟周期内完成写寄存器和写数据内存，实际上是在下一条指令执行的一开始，即下一条指令执行的上升沿完成写入操作的。
6. 算术移位和逻辑移位的问题。一开始主要是不清楚算术移位和逻辑移位的具体运算过程上的差别。其中逻辑移位只用补0，但是算术移位会根据数据的符号有所不同。后来发现这两种移位都在Verilog语言中有实现，得以解决这个问题。
7. 关于跳转指令相关实现的问题。在一开始control部件的设计中沿用了教材上的设计思路，使用了Branch和Jump这些指令。但后来陆续添加了对jal,jr,jalr这些指令的实现后采取这样的控制指令方式未免繁琐。故最终的设计中选用了PCsrc作为跳转类型指令的控制信号（可以看成是多级译码的结果），使整体的设计更为清晰简洁。
8. 关于立即数符号扩展的问题。立即数符号扩展的情况比较复杂，较为简单的有高位置零和有符号数的符号扩展，但也存在lui这样的加载立即数到高位的指令，需要在立即数的地位补充零来进行扩展，需要符号扩展单元提供额外的支持。
9. Modelsim的调试问题。Verilog语言是并行执行的，所以在modelsim的运行过程中单步执行的逻辑非常奇怪，如果用单步调试的话很难控制程序执行的进度。后来摸索出可以在时钟跳变的那一条语句中打上断点，然后放入循环中重复执行，可以较好地控制程序执行的进度。
10. 关于读取测试代码的二进制文件的问题。首先是系统函数$readmemh和$readmemb是不同的，一个用于读取十六进制文件，一个用于读取二进制文件，然而Verilog教材中没有详细介绍两者区别，导致一开始使用了错误的函数。其次是文件路径使用Unix风格的路径而不是Windows风格的路径，一开始错误地使用了Unix风格的路径，导致无法正确读取二进制代码。