MIPS-C3 based on VerilogHDL

Marc Wang

**目录**：

1. 文件组织、模块定义
2. 实现步骤、测试细节
3. 收获感悟、总结问题
4. 具体代码以及表格
5. 文件组织、模块定义
6. 文件组织：

文件组织形式与前两个实验同。

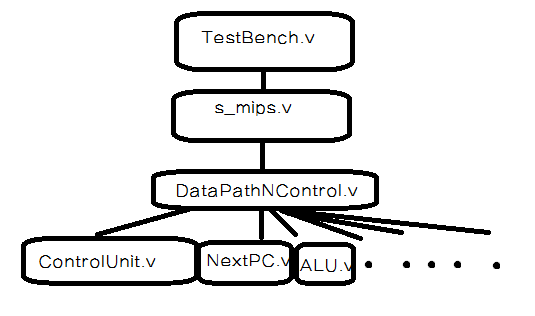


Figure 1

具体文件组成如下图：

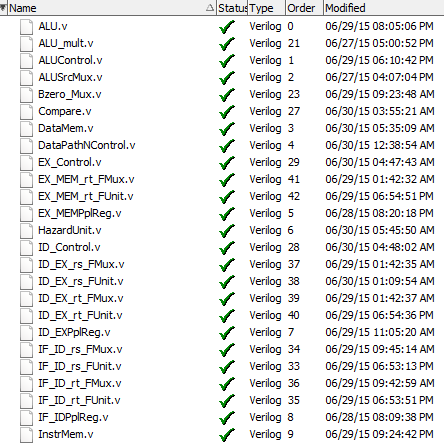


Figure 2

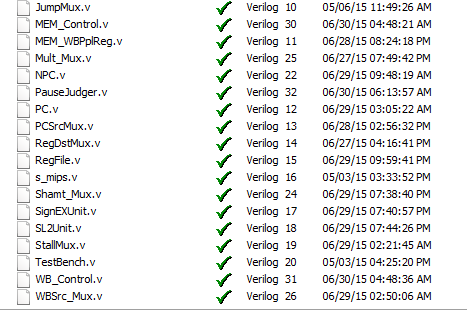


Figure 3

2、模块定义：

|  |  |
| --- | --- |
| 模块名称 | 模块功能 |
| ALU | 基本功能部件，实现CPU各种基本运算、条件分支、跳转、读写内存等基本操作 |
| ALU\_mult |
| Compare |
| DataMem |
| InstrMem |
| NPC |
| RegFile |
| SignEXUnit |
| SL2Unit |
| ALUSrcMux |
| Bzero\_Mux |
| JumpMux |
| Mult\_Mux |
| RegDstMux |
| Shamt\_Mux |
| WBSrc\_Mux |
| EX\_control | 分布形式控制器，分别在ID EX MEM WB阶段译码获得各种控制信号 |
| ID\_Control |
| MEM\_Control |
| WB\_Control |
| EX\_MEMPplReg | 流水线寄存器 |
| ID\_EXPplReg |
| IF\_IDPplReg |
| MEM\_WBPplReg |
| EX\_MEM\_rs\_Funit | 数据冒险、暂停检测单元，通过对各级控制器译码的分析判断是否转发、暂停。其中的HazardUnit配合PauseJudger，分析控制信号，判断是否需要暂停以及分支跳转的特殊处理。 |
| EX\_MEM\_rt\_Funit |
| EX\_MEM\_rs\_Funit |
| ID\_EX\_rs\_FUnit |
| ID\_EX\_rt\_Funit |
| IF\_ID\_rs\_FUnit |
| IF\_ID\_rt\_Funit |
| HazardUnit |
| PauseJudger |
| ID\_EX\_rs\_Fmux | 转发多选器 |
| ID\_EX\_rt\_FMux |
| IF\_ID\_rs\_Fmux |
| IF\_ID\_rt\_Fmux |

1. 实现步骤、测试细节
2. 实现步骤：

（i）参考L12-流水线工程化方法(更新28-29-33-36\_38\_39\_41\_42) .pdf，学习整个实验过程。（4h）

（ii）建立数据通路、数据通路合并、指令分析及分类、暂停、转发、转发与数据通路合并、读寄存器指令分类及各类真值表、写寄存器指令分类及各类真值等表格。（2d\*8h）

（iii）根据（ii）中的表格写verilog代码。在实现上述的各类指令的分类时，采用真值表化简最小项的方式，借助<http://www.codeproject.com/Articles/649849/A-Cplusplus-Karnaugh-Map-Minimizer-Infinte-Variabl>上的开源C++实现的卡诺图化简工具。（12h）

（iv）调试代码。

2、 测试过程：

首先对48条指令逐条调试，再对

两个测试用例测试、调试。具体调试方式为打开mars及modelsim仿真，mars出一条指令结果，modelsim也出一条结果，比较结果，若不同则记录modelsim的当前时间，然后根据这条“问题指令”，调通cpu设计，再回到记录的时间，继续比较下一条。（12h）

1. 收获感悟、总结问题

1、收获：

（i）工程化方法其实是一种mips流水cpu设计的一种科学方法，再保证了效率的同时更保证了设计的正确性。

（ii）对流水cpu的设计特别是转发处理有了更深层的理解，而不仅仅局限于教材第4章的转发分析。

（iii）锻炼了团队协作能力。

（iv）极大的锻炼了调试代码的能力。

2、问题及技巧：

（i）在整理指令分类时，发现书本附录的P.517 add与addu的funct位有误。

（ii）技巧：在调试时，鼠标移到没有定义的变量上不会显示任何数值。根据这点，首先逐个变量检查，是否有未定义变量。

（iii）技巧：在连接个模块之前，可以根据绘制的表格画出数据通路的大概的设计图，再进行连线。

（iv）需要对DataMem进行全清零的初始化，否则仿真可能会出错。（store类指令后有lb、lh等指令时）

（v）技巧：可以使用真值表化简最小项的方式来将已分类的指令进行类之间的区分。

（vi）RegFile采用内部转发的方式解决同周期写读同一寄存器的问题。

（vii）注意需要设置额外的几个控制信号来应对某些指令：是否为有符号运算（符号扩展、ALU运算以及DataMem的取内存值后扩展都会用到）、是否用到shamt位(及是否为sll类指令)等。

（viii）在调试一整段长代码时，建议采取上文二、2所阐述的调试方法，这样效率比较高。

（viiii）在转发分析的表格中，只是对于各数据到寄存器的转发的分析，没有涉及Mem值到Mem单元的转发分析（及lw-sw的情形）。

（x）注意，sll指令的op=0，funct=0，所以nop指令（或者气泡）也被视作一条对$0右移并存入$0的指令，可能在转发判断上有一些问题。（问题情形：在空间顺序上，stall后边跟了需要读$0的指令，以及nop前有需要写$0的指令，解决方法为加stall时把流水线寄存器rs、rt值的位全置为0）

四、具体代码以及表格

(见附录)