实验流程

添加指令	（52条）使用lab4工程，导入funcTest_independent下的coe，独立测试6类指令。（通过观察仿真波形图中寄存器的值是否与汇编代码中注释一致） （52条）使用func_test_v0.03\soc_sram_func工程（将mycpu_top封装成sram接口），通过64个测试点。（基于trace调试） （57条）添加异常处理、CP0、剩余5条指令，功能测试通过89个测试点。
axi接口支持	将cpu封装成axi接口。使用func_test_v0.03\soc_axi_func，通过89个测试点。 使用perf_test_v0.01\soc_axi_perf，通过性能测试。
上板测试	使用n4ddr下的func_test和perf_test工程，生成bitstream，上板观察是否成功。 不成功则仍需继续仿真调试。性能测试上板成功则可以计算CPU性能得分，性能分和扩展部分cache的加分有关。
添加基本cache	实现直接映射写直达的cache
扩展部分（加分）	完善cache或实现TLB

上面的每个部分根据完成程度，按照评分标准给分

独立测试前6类指令

此阶段需要用到的文件位于lab4目录下。

刚开始，我们需要添加前6类指令（52条），因此我们提供了单独测试每类指令的测试程序。其汇编代码和coe文件位于`lab4/funcTest_independent`目录下。(测试用例来自《自己动手写CPU》)

我们可以使用计组实验4的工程也可以使用提供的工程`lab4/project_1`(推荐)运行测试。

测试方法

将需要测试的一类指令的obj中的`inst_ram.coe`文件加载到自己的指令ram中，运行仿真。

在每个`inst_rom.s`文件中都有对应的coe的源代码，汇编代码中的注释为**正确执行到当前代码的时候，对应的寄存器的值**，所以对比仿真的波形图，将regfile模块中的32位寄存器添加入波形图中（添加变量方式为在Scope->Objects中找到变量，鼠标右键添加），对比相应的寄存器的值是否相同即可判断指令正确与否。

注意：原本计组实验4通过在testbench里检查CPU sw时的数据来在控制台输出PASS信息。现在我们testbench里没有该部分逻辑，我们是需要自己去对比波形图来检查是否正确的。

功能测试

此阶段需要用到的文件位于func_test目录下。

独立测试程序比较简单，因此在通过了前6类指令的独立测试后，**还不能认为我们的cpu实现正确**，我们现在需要运行更加复杂的**功能测试**程序。该功能测试程序包含89个测试点，测试了指令、延迟槽、异常等情况。为了运行功能测试，我们需要**将我们的CPU接入提供的SoC中去。**

刚开始我们的cpu的顶层模块是自己定义的接口，主要包含指令和数据的访存信号。现在，我们需要将我们的CPU的顶层封装成`mycpu_top`模块，以便被SoC调用。

功能测试涉及到两个SoC，一个是**sram接口**的SoC，对应代码位于`func_test/soc_sram_func/rtl`下。另一个是**axi接口**的SoC，对应代码位于`func_test/soc_axi_func/rtl`下。我们刚开始接入soc_sram进行测试，在将CPU封装成axi接口后，再接入soc_axi进行测试。

接入soc_sram

在独立测试了52条指令后，便可以接入soc_sram。

接入soc_sram需要将我们的mycpu_top封装成sram接口。这个过程比较简单，因为我们原本的指令和数据访存信号和sram信号基本是一一对应的。关于sram信号的定义请参考相关文档。

接入了soc_sram后，如果前52条指令实现正确，则可以通过前64个测试点，否则需要进一步调试。

在成功通过前64个测试点后，我们便可以添加剩余的5条指令。正确实现后可以通过89个测试点。

trace调试

接入soc后，我们引入了trace调试机制，可以**自动化地定位**到我们cpu运行错误的地方。关于trace调试说明请参考***A11_Trace比对机制使用说明_v1.00***文档。为了进行trace调试，我们需要在mycpu_top模块引出相关的**比对信号**——`debug_wb_pc，debug_wb_rf_wen，debug_wb_rf_num，debug_wb_wdata`。信号含义参考相关文档。

运行仿真时，Tcl控制台会每个10000ns输出当前仿真的时间，以及当前`debug_wb_pc`的值。每通过一个测试点还会输出通过的测试点编号。

```
          [30452000 ns] Test is running, debug_wb_pc = 0xbfc46538
          [30462000 ns] Test is running, debug_wb_pc = 0xbfc465e0
          [30472000 ns] Test is running, debug_wb_pc = 0xbfc46684
          [30482000 ns] Test is running, debug_wb_pc = 0xbfc46728
          [30492000 ns] Test is running, debug_wb_pc = 0xbfc467cc
          [30502000 ns] Test is running, debug_wb_pc = 0xbfc46874
          [30512000 ns] Test is running, debug_wb_pc = 0xbfc46918
          [30522000 ns] Test is running, debug_wb_pc = 0xbfc469bc
----[30525045 ns] Number 8'd63 Functional Test Point PASS!!!
```

如果发生错误则会打印错误信息，这时就需要观察波形图进行进一步调试了。

```
--------------------------------------------------------------
[31773618 ns] Error!!!
    reference: PC = 0xbfc00384, wb_rf_wnum = 0x1b, wb_rf_wdata = 0x01af5435
    mycpu    : PC = 0xbfc58298, wb_rf_wnum = 0x08, wb_rf_wdata = 0xf6865a84
```

关于soft/func_full, func_part

func_full是所有的57条指令的总测试集，共89个测试点。

func_part目录包含三个obj文件：

- obj_1(对应funt_full中的第1到47条测试)
- obj_2(对应funt_full中的第48到64条测试)
- obj_3(对应funt_full中的第65到89条测试)

从测试点65开始涉及异常。因此在测试异常相关指令时，我们可以使用obj_3直接进行测试，而不用等待前面的测试点通过。

注意

由于功能测试被拆成了三个部分，因此我们的golden_trace也生成了三个部分，所以在使用trace的时候需要做一些修改：如使用`obj_1`测试的时候，需要将`testbench/mycpu_tb.v`中的

```
`define TRACE_REF_FILE "../../../../../../../cpu132_gettrace/golden_trace.txt
```

修改成

```
`define TRACE_REF_FILE "../../../../../../../cpu132_gettrace/golden_trace_1.txt
```

同理使用哪个obj，就需要将testbench文件修改成对应那一条golden_trace。如果遇到`file can't open`的问题可以将路径改为**绝对路径**。

接入soc_axi

在通过了sram接口的功能测试后，我们需要将CPU封装成axi接口。封装正确后，可以通过axi接口的功能测试。

通过了sram接口的功能测试后，说明我们57条指令基本实现正确，我们有了一个基本正确的CPU核。通过了axi接口的功能测试后，说明我们的CPU被正确封装成了axi接口，我们的CPU才可以真正作为一个模块和其它模块对接，用于搭建完整的计算机系统。

想了解更多关于功能测试的详细说明，可以参考***功能测试说明_v0.01***，其中一部分看不懂的细节可以忽略。

性能测试

此阶段需要用到的文件位于perf_test目录下。

将cpu封装成axi接口并且通过功能测试后，我们便可以运行性能测试了。性能测试含10个基准测试程序，以龙芯132的运行时间作为基准，可以测试我们CPU的**性能**。

十个测试程序：

表 1-1 性能测试程序

序号	测试程序	性能测试程序介绍
1	bitcount	来自 Mibench 测试 automotive 集，统计一个整数数组包含的 bit 中 1 的个数
2	bubble_sort	冒泡排序算法
3	coremark	嵌入式系统中 CPU 性能的测试程序，2009 年由 EEMBC 发布。程序包括查找和排序、矩阵操作、状态机和循环冗余操作四部分算法
4	crc32	来自 Mibench 测试 telecomm 集，CRC32 计算工具
5	dhrystone	程序的主要目标是测量处理器的整型运算性能
6	quick_sort	快速排序算法
7	select_sort	选择排序算法
8	sha	来自 Mibench 测试 security 集，SHA 散列算法
9	stream_copy	来自 Stream 测试集的 Copy 操作，访问一个内存单元读出其中的值，再将值写入到另一个内存单元
10	stringsearch	来自 Mibench 测试 office 集，字符串查找工具

仿真时同样会输出信息：

```
Test begin!
…(不同程序有不同打印)

… PASS!... (不同程序有所不同)

…: Total Count =…(不同程序有所不同)
```

bitcount输出：

```
================================================================
Test begin!
bitcount test begin.

Bit counter algorithm benchmark



811

a0c5

4902

================================================================
Test end!
----PASS!!!
```

其它的输出示例可以通过运行`soc_axi_perf_demo`获得。

注意：仿真时不要使用allbench下的coe，因为该coe是根据拨码开关来确定测试哪个程序，用于上板。

详细的性能测试说明可以参考***性能测试说明_v0.01***

调试

一般来说功能测试通过，性能测试还是有可能不通过的。

这个时候还需要继续调试，不过性能测试是没有提供trace的。因此我们可以自己生成trace，自己生成trace的方式为使用`func_test/cpu132_gettrace`下的testbench并进行改写。

上板测试

由于原始的工程使用的是龙芯杯比赛的实验平台，因此需要进行移植。移植后的工程位于n4ddr目录下。

```
.
|-- func_test_v0.01
|   |-- cpu132_gettrace
|   |-- log.txt
|   |-- readme.txt
|   |-- soc_sram_func
|   `-- soft
|-- perf_test_v0.01
|   |-- log.txt
|   |-- readme.txt
|   |-- soc_axi_perf
|   |-- soc_axi_perf_demo
|   `-- soft
|-- readme.txt
`-- score.xls
```

功能测试只包含sram接口的soc。

上板现象参考相关文档。