The cache in MIPS pipeline timing simulator

实验三 设计MIPS五级流水线模拟器中的Cache

一、实验背景与介绍

在本实验中，你将扩展时序模拟器（用`C`语言编写）以对`指令/数据Cache`进行仿真。 与`RTL`仿真不同，实验提供的模拟器是用`C`语言编写的，是一个更高层次的抽象模型设计用于快速进行架构进行探索。因为更高级别地描述和仿真硬件抽象级别更容易查看不同的设计选择将如何影响性能。

注意模拟器的算法和结构不需要与处理器的算法和结构完全匹配算法和结构，只要结果相同即可。虽然我们不再有底层的实现，只需要知道一个程序要执行多少个周期（我们对周期级别进行仿真）。

我们将为你提供基础的模拟器，该模拟器模拟一个简单的`MIPS`处理器，需要你自行学习`Cache`相关知识，在原有模拟器的基础上拓展`cache`部分，`cache`具体细节要求请查阅实验要求部分。

资料下载：链接: https://pan.baidu.com/s/1SH3i4_HIkSi0I_3uz3_hAw 提取码: 7410

二、实验目的

- `Cache`结构及功能的设计
- 了解指令流水线运行的过程
- 探究`Cache`对计算机性能的影响

三、实验配置与步骤

1.准备工作：

本实验已经提供了支持`MIPS`指令集的流水线计时模拟器，通过这个模拟器你将可以更好的了解计算机是如何运行的。这台计算机主要包括两个部分：`CPU`和内存。计算机要执行的程序（包括代码和数据）都存储在内存中，`CPU`会将指令从内存中取出，进行解码并且执行代码所表示的操作（包括算术运算、逻辑运算以及存储器控制操作）。要注意的是，这里的`CPU`所采用的指令集是`MIPS`。

2.编译和运行

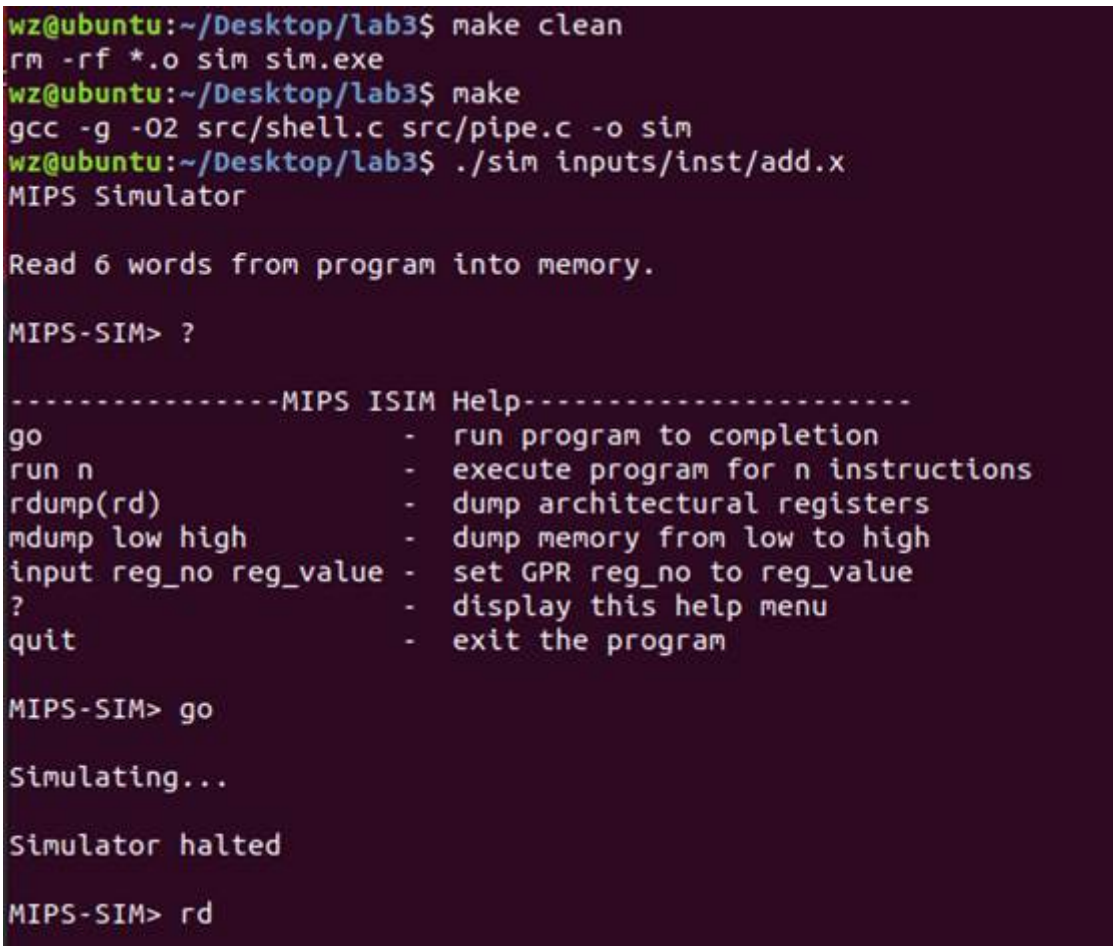
以下命令均在`linux`系统下举例，里面的程序也都是默认是`linux`中运行的，如果在`Windows`中，需要自行安装`make`，默认`Windows`使用`run3`，`Linux`使用`run2`，可参考源码自行配置。

首先说下文件结构

```
|-your_lab_dir
    |-Makefile
    |-run2        : python2运行脚本
    |-run3        : python3运行脚本
    |-basesim     : 标称对比答案使用
    |-basesim.exe : 标称对比答案使用，兼容windows
    |-inputs/     : 测试文件
    |-src/        : 可以自行添加其他文件，如cache.c，cache.h
        |-pipe.c  ：流水线程序，自行修改
        |-pipe.h：
        |-shell.c
        |-shell.h
        |-mips.h  : MIPS相关的定义
```

代码逻辑写好后执行命令`make`,即可在当前目录下生成名为`sim`的可执行程序，

可以运行`sim`可执行程序,进行调试，见下图



若要运行某个文件，需要执行下面的命令：

```
./sim your_file_path.x
```

具体指令的使用逻辑参考`shell.c`

当模拟器开始运行时，你可以输入`?`来获得命令列表

```
MIPS-SIM> ?

----------------MIPS ISIM Help----------------------
go                      -  run program to completion
run n                   -  execute program for n instructions
rdump                   -  dump architectural registers
mdump low high          -  dump memory from low to high
input reg_no reg_value - set GPR reg_no to reg_value
?                       -  display this help menu
quit                    -  exit the program

MIPS-SIM>
```

相关命令的具体实现在`shell.c`文件中，有兴趣的同学可以去看一下，如下

```c
void get_command() {
  char buffer[20];
  int start, stop, cycles;
  int register_no, register_value;

  printf("MIPS-SIM> ");

  if (scanf("%s", buffer) == EOF)
      exit(0);

  printf("\n");

  switch(buffer[0]) {
  case 'G':
  case 'g':
    go();
    break;
```

这是一个功能十分强大的`MIPS`流水线计时模拟器，它将一条`MIPS`指令的生命周期分为`5`个模块：取指->译码->执行->访存->回写。整个过程为取指令，指令译码，将译码出的指令放到算术逻辑运算部件`ALU`上执行，根据`ALU`算得的结果进行访存和将访存的结果写回寄存器。相关的代码实现在`pipe.c`中。如取指阶段

```c
void pipe_stage_fetch()
{
    /* if pipeline is stalled (our output slot is not empty), return */
    if (pipe.decode_op != NULL)
        return;

    /* Allocate an op and send it down the pipeline. */
    Pipe_Op *op = malloc(sizeof(Pipe_Op));
    memset(op, 0, sizeof(Pipe_Op));
    op->reg_src1 = op->reg_src2 = op->reg_dst = -1;

    op->instruction = mem_read_32(pipe.PC);
    op->pc = pipe.PC;
    pipe.decode_op = op;

    /* update PC */
    pipe.PC += 4;

    stat_inst_fetch++;
}
```

四、实验要求及拓展

在此模拟器的基础上增添 `cache`，并探究 `cache` 对模拟器性能的影响。

`cache` 的具体要求如下，要求设计加 `cache` 版本和不加 `cache`

1. Cache具体要求

	Instruction Cache	Data Cache
Size	8 KB	64 KB
ways	4	8
Block size	32 bytes	32 bytes
Number of sets	64	256
Replacement（替换策略）	LRU	LRU
Sets index（组的索引）	PC的[10:5]	Address的[12:5]
Visit time（访问时期）	取指阶段	访存阶段

2. 延时的设计

基础版的 `Demo` 可以理解成数据和指令全在 `cache` 中，不存在延迟的情况，这里我们添加 `cache` 所带来的的时间开销抽象成消耗的 `cycle`，即一次访存需要 `50` 个 `cycles`。即当 `CPU` 在第 `0` 个 `cycle` 向 `Cache` 索要数据后，主存会在第 `50` 个 `cycle` 送给 `cache`。这里为了简化实验，我们假设 `DCache` **在写回 `cache` 的时间可以忽略。**

3. Cache结构的设计

`Cache`在初始化的时候要求默认是空的。`Cache`中可以包含`tag`、`valid`、`address`等，可以自行学习相关的结构。在完成实验要求的结构后，可以结合现有`cache`升级结构，此点为加分项。

4. 拓展实验

- 4.1 修改`cache`的各项参数来探究`cache`对模拟器性能的影响（如`cache`大小，块大小，组大小，替换策略）
- 4.2 对替换策略进行优化，例如，你可以使用更复杂的哈希函数(或多个)进行映射。来缩短总时间周期
- 4.3 设计`benchmark`测试性能，benchmark中尽可能多使用内存，可以随机存取或连续存取。

五、实验总结

同学们只需要把`cache`的逻辑嵌入到现提供的`MIPS`模拟器中即可，模拟器可以正确处理流水线中数据依赖问题，可以通过增加气泡的方式绕过并正确处理他们。我们的目标是让总的`cycle`尽可能的少，发挥`cache`应有的作用。我们为同学们提供了自检查批处理脚本，以检查编写的正确性，具体使用命令如下（以`Linux`为例）【确保使用前`make`过】

```
make run INPUT=inputs/inst/add.x
make run INPUT=inputs/inst/*.x
# 检查inputs目录下所有的程序
make run
```

最终的成绩会根据同学们完成总体程度来进行打分，鼓励对拓展实验进行深度研究。